Modelling of Electrochemical Promotion in Heterogeneous Catalytic Systems

A Thesis submitted to
The University of Manchester

For the Degree of
Doctor of Philosophy

In the Faculty of
Engineering and Physical Sciences

2014

Ioannis Fragkopoulos

School of Chemical Engineering and Analytical Science (SCEAS)

Table of Contents

List of Figures

Chapter 1

Chapter 2

Chapter 3

Chapter 4

Chapter 5

Appendix

List of Tables

Chapter 3

Chapter 4

Chapter 5

List of Publications, Presentations & Posters

Peer Reviewed Publications

- Fragkopoulos I.S. and C. Theodoropoulos, *Multi-scale modelling of electrochemically promoted systems*. Electrochimica Acta, 2014. *To be submitted*
- Fragkopoulos I.S. and C. Theodoropoulos, *Efficient computational methods for microscopic simulations in multi-scale systems*. Computer Aided Process Engineering, 32, 2014. *To appear*
- Fragkopoulos I.S., I. Bonis and C. Theodoropoulos, *Macroscopic multi-dimensional modelling of electrochemically promoted systems*. Chemical Engineering Science, 104, 647-661, 2013.
- Fragkopoulos I.S. and C. Theodoropoulos,. *Multiscale models of electrochemically-promoted large catalytic surfaces*. Proceedings of the 3rd International Conference on Simulation and Modeling Methodologies, Technologies and Applications, 155-162, 2013.
- Fragkopoulos I.S., I. Bonis and C. Theodoropoulos, *Multiscale modelling of spillover processes in heterogeneous catalytic systems*. Computer Aided Process Engineering, 30, 1013-1017,2012.
- Bonis I., S. Valiño-Pazos, I.S. Fragkopoulos and C. Theodoropoulos, *Modelling of micro- and nano-patterned electrodes for the study and control of spillover processes in catalysis*. Computer Aided Process Engineering, 29, 151-155,2011.

Oral Presentations

- *Fragkopoulos I. S., Theodoropoulos C.,* "Modelling of varying length-scaled catalytic films in electrochemically promoted systems", AIChE Annual Meeting 2013 (San Francisco, USA, November 2013).
- *Fragkopoulos I. S., Theodoropoulos C.,* "Multi-scale models of electrochemically promoted large catalytic surfaces", SCEAS PGR Conference 2013 (Manchester, UK, June 2013).

- *Fragkopoulos I. S., Theodoropoulos C.*, "Modelling of electrochemical promotion in systems of varying nano-metal lengthscales", 9th European Congress of Chemical Engineering (The Hague, The Netherlands, April 2013).
- *Fragkopoulos I. S., Bonis I., Theodoropoulos C.*, "Multi-scale modelling of backspillover processes in electrochemically promoted systems", AIChE Annual Meeting 2012 (Pittsburgh, USA, October 2012).
- *Fragkopoulos I. S., Bonis I., Theodoropoulos C.*, "Multiscale modelling of spillover processes in heterogeneous catalytic systems", 22nd European Symposium on Computer Aided Process Engineering (London, UK, June 2012).
- *Fragkopoulos I. S., Bonis I., Theodoropoulos C.*, "Multiscale modelling of Electrochemically Promoted systems", 20th Annual Conference in Computational Mechanics (Manchester, UK, March 2012).
- *Fragkopoulos I. S., Bonis I., Theodoropoulos C.*, "Modelling of micro- and nano-patterned electrodes for the study and control of spillover processes in catalysis", Research Group Meeting (Newcastle, UK, November 2011).

Poster Presentations

- *Fragkopoulos I. S., Theodoropoulos C.*, "Modelling of electrochemical promotion in nano-patterned catalytic systems", ChemEngDayUK 2013 (London, UK, March 2013).
- *Fragkopoulos I. S., Theodoropoulos C.*, "Multi-scale modelling of backspillover process in CO Electrochemically Promoted oxidation for robust design and control of EPOC systems", Process Integration Research Consortium 2012 (Manchester, UK, October 2012).
- *Fragkopoulos I. S., Theodoropoulos C.*, "Multi-scale modelling of backspillover process in CO Electrochemically Promoted oxidation", SCEAS PGR Conference 2012 (Manchester, UK, June 2012).
- *Fragkopoulos I. S., Bonis I., Theodoropoulos C.*, "Multiscale modelling of spillover processes in heterogeneous catalytic systems", Process Integration Research Consortium 2011 (Manchester, UK, October 2011).
- *Fragkopoulos I. S., Theodoropoulos C.*, "Modelling of micro- and nano-patterned electrodes for the study and control of spillover processes in catalysis", SCEAS PGR Conference 2011 (Manchester, UK, June 2011).

Abstract

The subject of this work is the development of accurate frameworks to describe the electrochemical promotion of catalysis (EPOC) phenomenon. EPOC, also known as non-Faradaic electrochemical modification of catalytic activity (NEMCA), refers to the enhancement of the catalytic performance by application of current or potential in a catalyst/support system. Although this technology is of increasing interest nowadays in the field of modern electrochemistry and exhibits a great industrial potential, there are still just a few commercial applications, partly because the addressed phenomenon is not fully understood and has not been modelled to allow robust system design and control.

For this purpose, a systematic multi-dimensional, isothermal, dynamic model is developed to address the EPOC phenomenon using the electrochemical oxidation of CO over Pt/YSZ as an illustrative system. The formulated model is based on partial differential equations (PDEs) accounting for the simulation of the mass and charge transport as well as the electrochemical phenomena taking place at the triple phase boundaries (TPBs, where the gas phase, the catalyst and the support are all in contact) implemented through a commercial finite elements method (FEM) software (COMSOL Multiphysics). The constructed model is used in conjunction with experimental data for parameter estimation purposes, and a validated model is obtained. The results demonstrate that the effect in such a system is strongly non-Faradaic, with Faradaic rates 3 orders of magnidute lower than the non-Faradaic ones.

The formulated model is extended to describe the various processes taking place in the electrochemically promoted CO combustion system at their characteristic length-scales. The proposed framework couples a macroscopic model simulating charge transport as well as electrochemical phenomena occuring at the TPBs implemented through a FEM-package and an in-house developed efficient implementation of the kinetic Monte Carlo method (kMC) for the simulation of reaction-diffusion micro-processes on the catalyst. Dynamic communication of macro- and micro-scopic models at the TPBs results in the construction of an integrated multi-scale system. Comparison between the multi-scale framework and a fully macroscopic model is carried out for several sets of operating conditions and differences between the two models steady-state outputs are presented and discussed.

A detailed FEM/kMC model, regardless of accurately simulating the several phenomena at their appropriate length-scales, might not be suitable for large system simulations due to the high computational demand. To address this limitation, a computationally efficient coarse-graining methodology, the so-called gap-tooth method, is implemented. In this scheme the catalytic surface is efficiently represented by a small subset of the spatial domain (tooth) separated by gaps. While kMC simulations within each individual tooth (micro-lattice) are used to predict the corresponding evolution of the micro-processes, intelligent interpolation rules are employed to allow for the exchange (diffusion) of species between consecutive micro-lattices. A validated gap-tooth/kMC scheme is obtained and it is exploited for FEM/gap-tooth/kMC electrochemically promoted CO oxidation simulations achieving high computational savings.

Declaration

No portion of the work referred to in the thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Ioannis S. Fragkopoulos

Copyright Statement

Acknowledgements

First, I would like to acknowledge my family for the unconstrained encourangement, support and truthful love they have shown me all these years. A big thanks to my partner, Silvia, for her invaluable support shedding some light on the dark. This thesis is dedicated to them.

Throughout this highly demanding journey I have been privileged to be supervised by an experienced Academic, Dr Constantinos Theodoropoulos. I would like to express my gratitude to him for guiding me, providing me with valuable feedback that helped me deal with many challenging problems as well as for encouraging and supporting me to follow an academic career.

I am also very gratefull to my MEng advisor, Dr Ioannis K. Kookos, who encouraged me to continue my studies. I would not have written this thesis, if I had not experienced his truthfull support.

I would also like to thank Prof Ian S. Metcalfe, Dr Danai Poulidi and Dr Evangellos Papaioannou for broadening my knowledge on Electrochemical Promotion through fruitful discussions we had during our collaboration. Thanks also to Dr Ioannis Bonis for his guidance during the first year of my studies. Furthermore, Prof Constantinos G. Vayenas is also greatly acknowledged for reviewing my first journal publication. The valuable feedback he provided me with, helped me gain a better understanding of the EPOC phenomenon. Last but not least, I would like to thank Prof Claire S. Adjiman and

Dr Alastair D. Martin for reviewing my PhD Thesis and for helping me find answers on complex electrochemical phenomena.

During my studies at SCEAS, I was lucky to be part of the Centre of Process Integration community and especially of the CPI students, who I was very honoured to represent for two years in a row. I would like to thank ALL of them for sharing many moments of cheerfulness together over these years and also for providing advice during my leadership. Many thanks to my friends - Katerina, Elias, Lluvia, Lalo, Liliana, Mona, Mesut - here in Manchester, for sharing many memorable experiences over these years, far from our home countries.

Finally, I would like to acknowledge the Engineering and Physical Sciences Research Council (EPSRC) UK for the financial support provided me throughout my PhD studies (EP/G022933/1) and also for awarding me the EPSRC Doctoral Prize 2013/2014, a prestigious fellowship which will enable me to take my research to the next level.

"A journey of a thousand miles starts with a single step."

Lao Tse

Nomenclature and Abbreviations

Nomenclature

Roman Symbols

A_S	Catalytic surface area, m^2
C_i	Concentration of species i in the gas phase, mol m^{-3}
D_i	Diffusion coefficient of species i, m^2 s^{-1}
$d_{x/y}$	Length of a tooth in x/y-direction, molecules
$D_{x/y}$	Distance between the centres of two adjacent teeth in x/y-direction, molecules
E_A	Activation energy of a reaction, J mol^{-1}
E_{eq}	Equilibrium potential difference at electrochemical reaction interface, V
F	Faraday constant, A s mol^{-1}
F_d	Feed of gas mixture, m^3 s^{-1}
$gap_{x/y}$	Distance between the boundaries of two teeth in x/y-direction, molecules
I	Current density, A m^{-2}
	Flux of ingoing species, molecules
$\mathbf{J_i}$	Current density, A m^{-2}
$J^{A/C}$	Anodic/Cathodic current density, A m^{-2}
k_i	Rate constant for adsorption, m^3 mol^{-1} s^{-1}
	Rate constant for surface reaction, s^{-1}
k_{-i}	Rate constant for desorption, s^{-1}
$k_{o,i}$	Pre-exponential coefficient of adsorption rate constant, m^3 mol^{-1} s^{-1}

$k_{o,-i}$ Pre-exponential coefficient of desorption rate constant, s^{-1}

L Length of electrolyte/anode, m

L_{count} Length of the counter electrode at the cathode, m

M_i Molecular weight of species i, mol m^{-3}

N_{AV} Avogadro constant, mol^{-1}

$\mathbf{N}_i$ Total molecular flux of species i, mol m^{-2} s^{-1}

n_r total number of micro-processes

N_s Concentration of active sites on catalytic surface, mol m^{-2}

n_{wxp} Number of experimental points

O Flux of outgoing species, molecules

P_{A*} One site conditional probability

$P_{A*/B*}$ Two site conditional probability

P_i Partial pressure of species i, atm

P_{ref} Reference (total) pressure, atm

PI Promotional index

Q_j Charge source term of medium j, A m^{-3}

R Ideal gas constant, J mol^{-1} K^{-1}

R_i Volumetric reaction rate of species i, mol s^{-1} m^{-3}

r_i Rate of reaction i, s^{-1}

R_i' Overall reaction rate of species i, s^{-1}

S_i Sticking coefficient of species i

T Absolute temperature, K

t Time, s

V_{OC} Thermodynamic open circuit potential, V

W Width of electrolyte/anode/cathode, m

X_j Mole fraction of gaseous species j

Greek Symbols

α Charge transfer coefficient

Interpolation coefficient

γ Exchange current density pre-exponential coefficient, A m^{-2}

Permanent rate enhancement factor

$\hat{\Gamma}_i$ Transition probability of micro-process i, s^{-1}

η Overpotential, V

θ_i Coverage of species i on catalytic surface

Λ Faradaic efficiency or enhancement factor

$\bar{\mu}_i$ Electrochemical potential of species i, J mol^{-1}

μ_i Chemical potential of species i, J mol^{-1}

π Mathematical constant

ρ Charge density , A m^{-3}

Rate enhancement ratio

σ Charge conductivity, Ω^{-1} m^{-1}

Φ Local electrostatic potential, V

Φ_{pellet} Operating potential in the solid oxide single pellet, V

Ω_{A*} Size of class A*

Ω_T Total number of catalytic lattice sites

Superscripts and Subscripts

A Anode

Au	Gold
BSS	BackSpillover Species
C	Cathode
CO	Carbon monoxide
CO_2	Carbon dioxide
e	Electron
el	Electronic
g	Gas phase
He	Helium
in	Inlet of the reactor
io	Ionic
o	Standard conditions
out	Outlet of the reactor
O_2	Oxygen
O^{2-}	Oxygen anion
$[O^{\delta-}- \delta+]$	Backspillover species
Pt	Platinum
S	Empty active sites
T	Total
YSZ	Yttria Stabilized Zirconia

Abbreviations

BSS	Backspillover species
CFD	Computational fluid dynamics

CSTR	Continuous stirred-tank reactor
CTMC	Continuous time Monte Carlo
E-R	Eley-Rideal
EPOC	Electrochemical promotion of catalysis
FD	Finite differences
FEM	Finite element method
kMC	kinetic Monte Carlo
L-H	Langmuir-Hinshelwood
MEPR	Monolithic electropromoted reactor
MPI	Message passing interface
NEMCA	Non-Faradaic electrochemical modification of catalytic activity
PDE	Partial differential equation
P-EPOC	Permanent electrochemical promotion of catalysis
PEM	Proton exchange membrane
SA	Simulated annealing
SOFC	Solid oxide fuel cell
SQP	Sequential quadratic programming
STM	Scanning tunneling microscopy
TPB	Triple phase boundary
TPD	Temperature programmed desorption
UHV	Ultra hign vacuum
XPS	X-ray photoelectron spectroscopy
YSZ	Yttria stabilized Zirconia

Chapter 1

Introduction

The subject of this thesis is the construction of accurate models to describe and investigate the electrochemical promotion of catalysis (EPOC). The EPOC phenomenon accounts for the alteration of the activity of a catalyst, with a subsequent catalytic performance enhancement, of a metal (or metal oxide) deposited on a solid electrolyte, by means of electrical polarisation (Pritchard, 1990). This phenomenon was for the first time reported in the early 1980s by Vayenas and co-workers (Stoukides and Vayenas, 1981) and has since been investigated extensively by several research groups all around the world. The EPOC phenomenon is very promising in the field of modern electrochemistry due to its interdisciplinary nature involving different scientific fields (Figure 1.1), such as heterogeneous catalysis and electrocatalysis.

A description of the fields electrochemical promotion shares common features with is presented in the following sections.

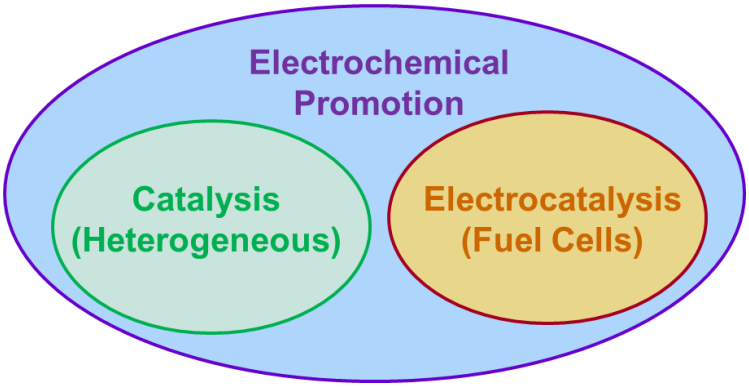


Figure 1.1: Schematic presentation of the scientific fields electrochemical promotion bears similar features with.

1.1 Catalysis

Catalysis refers to a chemical process where a substance, called a catalyst, is utilised decreasing the activation energy barrier of a chemical reaction and consequently, leading to an accelaration of the reaction rate without affecting its equilibrium. A catalyst can also facilitate reactions, i.e. biological reactions, that would not have been possible without its presence. The catalyst takes part in a reaction forming reactant intermediates which, by getting rearranged into products, regenerate the catalyst. A very important feature of catalysis is that since a catalyst is not consumed during a reaction process, it allows for operations over and over again (Gates, 1992).

A chemical reaction takes place either in the gas phase or in the solid phase or in the liquid phase. Moreover, it can occur at the interface between two or three phases. When the reaction occurs at a single phase, the process is termed as homogeneous catalysis, while heterogeneous catalysis stands for reaction processes taking place at an interface. A very significant catalytic process, which is neither heterogeneous nor homogeneous

(but something in between demonstrating some of the features of both), is enzymatic catalysis (or biocatalysis). Biocatalysis refers to a biological process where catalysts in the form of complex protein macromolecules, known as enzymes, are involved. (Bond, 1972).

This thesis focuses only on heterogeneously catalysed systems. Here, a catalyst comprises of 'active' sites where the reactants can be adsorbed and form surface intermediates. Consequently, the surface species can interact forming the final product, which desorbs from the catalytic surface regenerating the active sites. This process is represented by reaction mechanisms, such as the Langmuir-Hinshelwood (L-H) (Langmuir, 1918, 1922; Hinshelwood, 1926) and the Eley-Rideal (E-R) (Rideal, 1939; Eley and Rideal, 1941). In the L-H mechanism all the reactants get adsorbed at the catalyst surface and undergo surface reactions, while in the E-R mechanism some of the reactants get adsorbed and undergo reactions with the gaseous phase mixture.

This work accounts for the L-H mechanism of CO oxidation over Pt (a widely used noble metal catalyst), comprising of adsortion/desorption of O_2 and CO to/from the catalytic surface as well as a surface reaction between adsorbed CO and atomic O_2:

$$O_2(g) + 2 \cdot S \rightleftharpoons 2O \cdot S \tag{1.1}$$

$$CO(g) + \cdot S \rightleftharpoons CO \cdot S \tag{1.2}$$

$$O \cdot S + CO \cdot S \longrightarrow CO_2(g) + 2 \cdot S \tag{1.3}$$

where $\cdot S$ stands for an empty active site on the catalytic surface, $O \cdot S$ and $CO \cdot S$ denote the adsorbed atomic O_2 and CO species, respectively. Figure 1.2 illustrates a schematic representation of such a L-H CO oxidation mechanism.

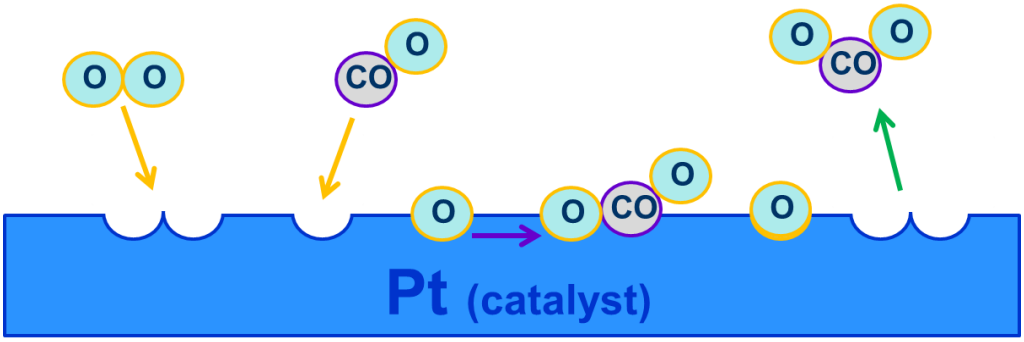


Figure 1.2: Schematic representation of the CO oxidation surface dynamics.

The overall CO oxidation reaction is given by:

$$\frac{1}{2}O_2(g)+CO(g)\xrightarrow{Pt}CO_2(g) \tag{1.4}$$

Detailed macroscopic or microscopic kinetic models should be taken into account to quantify the overall rate of reaction (1.4). The simplest empirical reaction model that was extensively used in early days of modelling heterogeneously catalysed reactions is the power law (or rate law) functional relationship kinetic model (Dornte, 1936; Kwan, 1956). In power law kinetics, the rate expression is described as a function of the gaseous phase species concentrations, i.e. for the CO oxidation system:

$$r_{CO_2} = k_r C_{O_2}^a C_{CO}^b \tag{1.5}$$

where a and b denote the reaction orders, and k_r is the reaction rate constant which is calculated using an Arrhenius expression:

$$k_r = k_0 e^{-\frac{E}{RT}} \tag{1.6}$$

where k_0 is the rate constant pre-exponential coefficient, E is the activation energy of the reaction, R is the ideal gas constant and T denotes the temperature.

Fitting equation (1.5) to experimental data, one can obtain the kinetic parameters, such as a, b, k_0 and E. Although the rate law expressions often lack predictive power, they are still being used when limited experimental data are available, due to the small number of parameters that need to be estimated.

Another commonly used approach is the Langmuir-Hinshelwood-Hougen-Watson (LHHW) model (Hougen and Watson, 1943, 1947). In the L-H-H-W model, one has to take into consideration the detailed L-H surface reaction mechanism, i.e. reactions (1.1) to (1.3), and assume that the adsorption/desortpion reactions (e.g. reactions (1.1) and (1.2)) are in equilibrium and also that one of the surface reactions (here we have only one surface reaction, Rxn.(1.3)) is the rate determining step. Consequently, the rate expression is developed using reaction kinetics (Boudart and Djéga-Mariadassou, 1984) and its predictions are compared against experimental data. If the rate expression cannot capture the experimental data qualitatively, a different set of assumptions (e.g. different rate determining step) is taken into account and an updated rate expression is derived. When the rate expression is sufficient to predict the experimental data qualitatively, the reaction rate constants are estimated using experimental data and a validated rate expression is obtained.

It should be noted here that both the power law and the L-H-H-W kinetic models are used to provide steady state predictions. For the prediction of the reaction rate transients, ordinary differential equations (ODEs) that describe the individual (gaseous/adsorbed) species rates in the system should be taken into account. Here, the adsorption/desorption reactions are not assumed to be in equilibrium and also no reaction is considered as the rate determining step:

$$\left.\frac{dC_i}{dt}\right|_{i=1,N} = \sum_{j=1}^{m} \nu_{i,j} r_j \tag{1.7}$$

where i denotes a gaseous/adsorbed species, N stands for the total number of gaseous and adsorbed species, m is the total number of reactions, $\nu_{i,j}$ is the stoichiometric coefficient for species i in reaction j and r_j is the rate of reaction j.

Furthermore, when both temporal and spatial variations need to be considered, i.e. in pattern formation on catalytic surfaces (Ertl, 1994; Raimondeau and Vlachos 2002a), continuum surface diffusion-reaction partial differential equations (PDEs) are employed:

$$\left.\frac{dC_i}{dt}\right|_{i=1,N} = -\nabla \cdot (-D_i \nabla C_i) + \sum_{j=1}^{m} \nu_{i,j} r_j \tag{1.8}$$

where D_i is the diffusion coefficient of species i.

Although the mean-field (macroscopic) models can be used to describe the spatial variations of the surface species concentrations, they are incapable of simulating spatial localised effects or (pair-wised) interactions between adsorbates (Salciccioli et al., 2011). Such molecular level spatial effects can be accurately treated using microscopic modelling approaches, such as the kinetic Monte Carlo (kMC) method.

The spatial (or lattice) kinetic Monte Carlo method carries out stochastic simulations to provide exact solutions for the probabilistic time-dependent master equation (Fichthorn and Weinberg, 1991):

$$\frac{dP_{a,t}}{dt} = \sum_{b} \left[W_{b\to a} P_{b,t} - W_{a\to b} P_{a,t} \right] \tag{1.9}$$

where $P_{a,t}$ is the probability that the lattice (catalytic surface) is in configuration *a* at time *t*, and $W_{a \to b}$ is the probability per unit time that the lattice will undergo a transition from configuration *a* to configuration *b*.

The solution of the master equation (1.9) is computationally expensive and cannot be achieved analytically for real (large) catalytic system simulations. Consequently, Monte Carlo algorithms have been developed to perform such probabilistic catalytic surface simulations. Bortz et al. (1975) were the first to develop a lattice kMC algorithm to simulate the Ising spin model and since then the lattice kMC method has been extensively used for the investigation of the effect of the local interactions between adsorbates (Silverberg et al., 1985; Myshlyavtsev and Zhdanov, 1989), for the simulation of kinetic phase transitions (Fichthorn et al., 1989; Zhdanov and Kasemo, 1994; Ziff et al., 1986) and for the investigation of the macroscopic (mean field) models effectiveness to demonstate similar system behaviour near critical points of the phase diagram (Dumont et al., 1986; Jensen et al., 1990; Salciccioli et al., 2011).

In this work, we use an in-house developed spatial kMC model based on the continuous time Monte Carlo algorithm proposed by Reese et al. (2001) to simulate the reaction-diffusion processes taking place on the catalytic lattice of interest and subsequently, to compute spatial and temporal homogenised (average) properties, such as reaction rates and species coverages, taking into account local interactions between the adsorbed species.

1.2 Electrocatalysis

Catalytic processes do not involve any net charge transfer. When a net charge transfer is involved in a reaction process accelerating its rate, the process is referred to as electrocatalysis, while the substance that assists such a process is denoted as electrocatalyst. An electrocatalyst serves both as a catalyst and as an electrode accelerating the chemical reaction through conducting the electric field transfer (Vayenas et al., 2001a; Balbuena and Subramanian, 2010).

An electrocatalyst (electrode) is in general covered by or deposited on an electrolyte, forming an electrode/electrolyte interface. Charge is transferred through the electrode by the movement of electrons and through the electrolyte by the movement of ions. An electrocatalyst is typically a solid dense, porous or dispersed metal, a semiconductor (metal oxide) or a liquid metal, while electrolytes can be liquid solutions, solids, fused salts or ionically conductive polymers (Bard and Faulkner, 2001). Electrocatalysis is concerned with processes taking place in electrochemical systems such as electro-organic synthesis (Yoshida et al., 2008), electrochemical sensors (Bakker, 2004), waste water treatment (Chen, 2004) and energy conversion devices (Winter and Brodd, 2004) (e.g. electrolytic cells, i.e. batteries, and fuel cells, i.e. solid oxide fuel cells (SOFC)), amonst others. SOFCs have received increasing attention during the last decades due to their capability to serve important catalytic processes (e.g. steam reforming) while generating electrical power. A schematic presentation of a H_2 fuelled SOFC operation is illustrated in Figure 1.3.

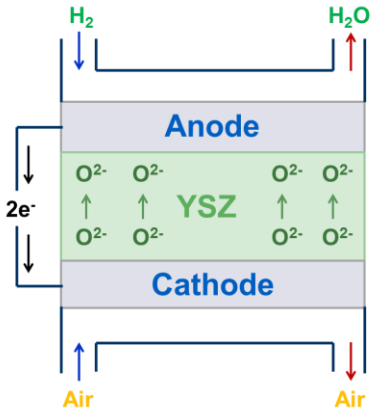


Figure 1.3: Schematic of a H_2 fuelled SOFC operating principle.

The SOFC system comprises of a solid oxide electrolyte, i.e. Yttria-Stabilised Zirconia (YSZ), and two porous metals that act as electrocatalysts -the anode and the cathode electrodes. The charge transfer is served through two half-cell electrocatalytic reactions. The cathode (positive) electrode in such a configuration serves the electrocatalytic reduction of $O_2(g)$ to oxygen anions (O^{2-}), while the anode (negative) electrode acts as an electrocatalyst serving the reaction of H_2 with O^{2-}:

$$\frac{1}{2}O_2(g) + 2e^- \longrightarrow O^{2-} \tag{1.10}$$

$$H_2 + O^{2-} \longrightarrow H_2O + 2e^- \tag{1.11}$$

Based on the two half-cell electrocatalytic reactions (1.10) and (1.11), the overall reaction is given by:

$$H_2 + \frac{1}{2}O(g) \longrightarrow H_2O \tag{1.12}$$

In such an electrocatalytic system, the induced increase in the reaction rate, Δr, is given by Faraday's law (Vayenas et al., 2001a):

$$\Delta r \approx I / 2F \tag{1.13}$$

where *I* is the current of the cell, *F* is the Faraday constant and *2* stands for the number of electrons exchanged between the electrochemical reactions (1.10) and (1.11).

Similar to the H_2 fuelled SOFC system, the electrochemical system utilised in this work comprises of two solid metal electrocatalytic films (e.g. Pt, Au) deposited on both sides of a solid electrolyte (e.g. YSZ). Nevertheless, here the system is of "single pellet" type where both the fuel and the air are in contact with the anode and the cathode sides of the pellet (Figure 1.4).

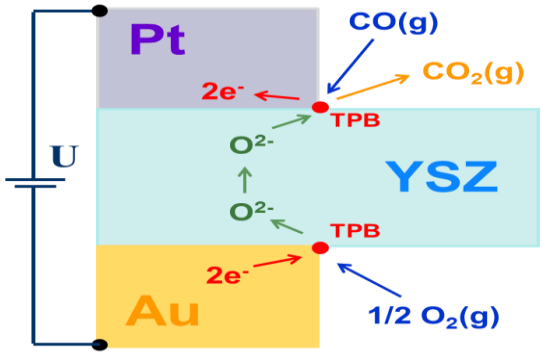


Figure 1.4: Schematic of the anodic (Pt) CO oxidation and the cathodic (Au) O_2 reduction electrocatalytic processes in a solid oxide single pellet.

The charge transfer in the system of interest is achieved through two half-cell electrochemical reactions, i.e. the anodic oxidation of CO and the cathodic reduction of O_2:

$$CO(g) + O^{2-}(YSZ) \xrightarrow{Pt} CO_2(g) + 2e^- \quad (1.14)$$

$$\frac{1}{2}O_2(g) + 2e^- \xrightarrow{Au} O^{2-}(YSZ) \quad (1.15)$$

The electrochemical reactions (1.14) and (1.15) take place at the triple phase boundaries (TPBs), i.e. an interface between the electrolyte, the electrode and the gas phase (as in Figure 1.4). The overall reaction is given by:

$$CO(g)+\frac{1}{2}O(g)\longrightarrow CO_2(g) \quad (1.16)$$

It should be noted that the induced increase in the reaction rate in the electrochemically promoted CO oxidation system of interest is much larger than *I/2F*. This is considered as one of the main differences between electrocatalysis and electrochemical promotion (Vayenas et al., 2001a) and it will be further discussed in the chapters that follow.

1.3 Promotion

Promotion in (electro)catalysis denotes the improvement of the catalytic activity and selectivity due to the presence of small quantities of one or more dopants (promoters) on the catalyst surface (Hegedus et al., 1987). Although by definition the action of a promoter leads to the enhancement of the catalytic performance, a high coverage of it on the catalytic surface can act as a poison impeding the performance of the catalyst (Kiskinova, 1992). The contribution of promoters in the field of industrial catalysis has been extremely valuable, since significant commercial catalytic processes such as the Fischer-Tropsch synthesis and the ammonia synthesis would not have been possible without the use of such substances (Bartholomew and Farrauto, 2005).

Promoters are typically added on the catalytic surface during ‘ex situ’ (before the catalyst operation) catalyst preparation and the subsequent reaction rate enhancement is termed as chemical (or classical) promotion (Ertl et al., 1997). Upon classical promotion, the concentration of the promoter cannot be modified or controlled during

catalytic operation. On the contrary, electrochemically generated promoting species, i.e. species that migrate over the catalytic surface due to electrical potential application, can be directly controlled 'in situ' (during experiment operation) by manipulating the applied potential difference. Such a method for promoting species supply on the catalytic surface is commonly reversible and leads to the effect of electrochemical promotion (Vayenas et al., 2001a).

A promoter is usually not being consumed during a catalyst operation. Nevertheless, promoting species generated by the EPOC effect (e.g. when using O^{2-} conductors as electrolytes), apart from diffusing on the catalyst surface, can also be consumed through interaction with catalyst co-adsorbed species and/or with the gas phase mixture (Vayenas and Pitselis, 2001). In this case, the promoting substances are denoted as 'sacrificial' promoters (Vayenas et al., 2001a).

1.4 Motivation and Objectives

The reduction of environmental pollution has become an issue of great concern, requiring more sustainable and more efficient methods of exhaust emissions conversion. Air pollutants, such as carbon monoxide, nitrogen oxides, hydrocarbons and organic emissions, are very effectively being converted to harmless emissions, using appropriate cost-effective heterogeneous catalytic systems. Nevertheless, the use of heterogeneous catalysis bears some significant bottlenecks, such as the short catalytic life time due to deactivation, the high system preparation cost since most of the catalytic

systems comprise of pricy metals (e.g. Pt) and the incapability of controlling the catalytic performance ‘in situ’ (Katsaounis, 2008).

The EPOC phenomenon is a promising candidate revealing a great potential in the gaseous emissions treatment technology, since it can increase the life time of the catalyst as well as its activity, leading to lower catalyst loading and to subsequent lower operating costs, modify the selectivity to the desired products and control the catalytic performance during an electrocatalytic process (Vayenas et al., 2001a; Katsaounis, 2008).

The exhaust gas treatment has extensively been investigated experimentally using EPOC systems (Alqahtany et al., 1992; Karasali et al., 1994; Marwood et al., 1996; Haller and Kim, 1997; Politova et al., 1997; Kaloyannis and Vayenas, 1999; Vayenas et al., 2001a) and a great commercial potential has been arised with the construction of a Monolithic Electrochemical Promoted Reactor (MEPR) which is considered as a hybrid between a typical monolithic honeycomb reactor and a flat (or ribbed) plate solid oxide fuel cell (Balomenou et al., 2004). However, only few modelling studies have been performed to describe the addressed phenomenon, which were mainly focused on general thermodynamic rules and simple surface kinetics (Metcalfe, 2001a, 2001b; Vayenas and Pitselis, 2001; Foti et al., 2002; Presvytes and Vayenas, 2007; Leiva et al., 2008; Huang et al., 2012), not allowing for the design of commercial applications.

The main objective of this work is the formulation of accurate modelling frameworks to describe the electrochemical promotion phenomenon. Such models aided by a good range of experiments can be used to:

1. Increase the understanding of the EPOC phenomenon and obtain insights on relevant complex phenomena.

2. Compute reliable estimates of parameters such as diffusion coefficients and reaction rate constants, under the effect of polarisation application.

3. Ultimately enable EPOC system robust design and control leading to the incorporation of the addressed effect in scaled-up commercial systems, such as exhaust gas treatment (Vayenas et al., 2001a), hydrogenation of organic compounds (Frenzel et al., 2001), product selectivity modification (Karasali, 1994) and fuel cells (Huang and Goodenough, 2009).

(a)

(b)

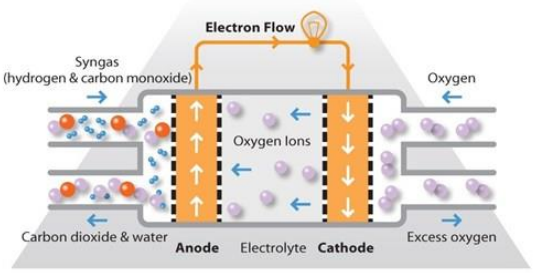


Figure 1.5: Illustration of EPOC systems potential commercial applications, (a) a catalytic converter (Eastern Manufacturing Inc.), (b) a fuel cell technology (Fuel Cell Today).

1.5 Organisation of the Thesis

The organisation of the thesis is as follows:

Chapter 1 introduced the core fields that the electrochemical promotion phenomenon shares common features with, and outlined the motivation and the main objectives of this study.

Chapter 2 summarizes the phenomenology as well as the key aspects of EPOC. A literature review on the most relevant experimental and theoretical studies in the field of electrochemical promotion and a synopsis of the contributions associated with this work are presented.

Chapter 3 provides an account of the construction of a systematic multi-dimensional macroscopic model which employs the finite elements method (FEM) for the simulation of the mass and the charge transport as well as the electrochemical processes in an electrochemically promoted system. The developed model is utilised in conjunction with experimental data for the estimation of parameters, that are of crucial importance for accurate EPOC system simulations, and a validated model is obtained. Sensitivity analysis of the estimated parameters is performed in order to assess the effect of the percentage change of each parameter on the system production rates. Parametric investigation of operating parameters is carried out and the obtained results are discussed.

Chapter 4 presents an extention of the previously developed model to simulate the reaction-diffusion phenomena taking place on the catalytic surface at the molecular

level. The proposed multi-scale framework links a macro-scopic model used for the simulation of the charge transport and the electrochemical phenomena, with a micro-scopic model which employs the kinetic Monte Carlo method for the simulation of the reaction-diffusion micro-processes occuring at the catalyst surface. Comparison between the multi-scale framework and a fully macroscopic model is carried out for several sets of operating conditions and differences between the two models are presented and analysed. The multi-scale framework is further exploited for parametric investigation in order to analyse the effects of operating conditions on the reaction rates.

Chapter 5 deals with the extension of the multi-scale framework to employ a coarse graining methodology, the gap-tooth method, for efficient large-scale EPOC simulations. The gap-tooth method is first implemented in a surface dynamics micro-scopic lattice kMC simulator and the combined gap-tooth/kMC framework is validated under open circuit conditions. The gap-tooth method is consequently implemented in the multi-scale model and the constructed FEM/gap-tooth/kMC framework is employed to perform closed circuit (EPOC) simulations with computational efficiency.

Chapter 6 summarizes the main findings drawn from this work and discusses future research recommendations, which could improve the proposed models, allowing electrochemical promotion to become a major candidate for future challenges in the field of modern electrocatalysis.

Chapter 2

Electrochemical Promotion of Catalysis

2.1 Phenomenon and key aspects

Electrochemical Promotion of Catalysis (EPOC), also known as Non-Faradaic Electrochemical Modification of Catalytic Activity (NEMCA), is the enhancement of catalytic activity and selectivity as a result of an electrochemically controlled migration of an overall neutral dipole of promoting species (ionic species accompanied by their compensating charge in the metal), i.e. $[O^{\delta-} - \delta^{+}]$ termed as 'backspillover' species, from the solid electrolyte to the catalytically active, gas exposed, electrode surface, when current or potential is applied between the two electrodes in a solid electrolyte system (Stoukides and Vayenas, 1981; Vayenas et al., 1988; Pritchard, 1990). More specifically, in the case of using an oxygen conductor as the solid electrolyte, oxygen anions (O^{2-}) are excorporated from the triple phase boundaries (TPBs), i.e. places where the gas phase, the catalytic film and the electrolyte are all in contact, generating backspillover species (BSS) which subsequently spill over the catalytic surface. Due to this migration, an effective double layer is formed over the catalytic surface altering the

work function of the catalyst and consequently, affecting the binding strength of the chemisorbed reactants. Figure 2.1 illustrates the effective double layer for a system of gaseous reactants (CO, O_2)/Metal/Yttria-Stabilised Zirconia (solid electrolyte).

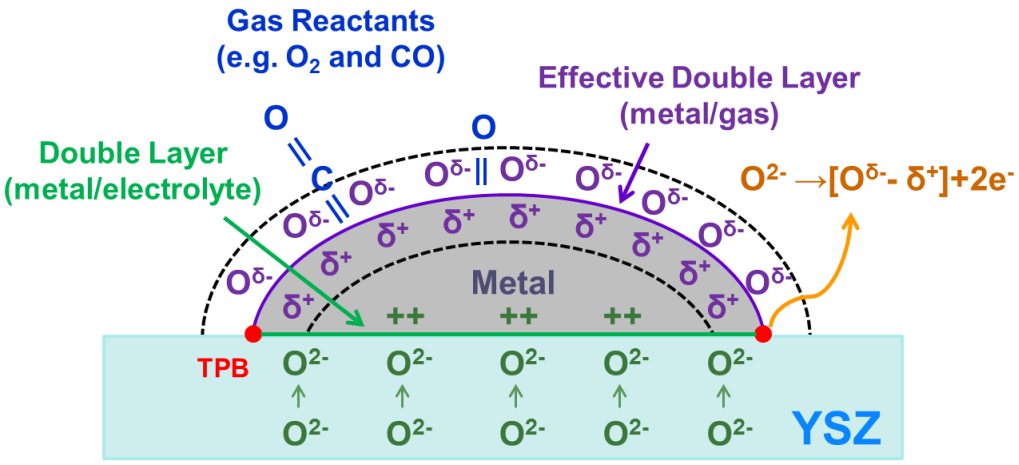


Figure 2.1: Schematic presentation of the effective double layer.

The conversion of oxygen anions to BSS is described by (Vayenas et al., 2001a):

$$O^{2-}_{YSZ} \rightarrow \left[O^{\delta-} - \delta+\right] + 2e^{-} \qquad (2.1)$$

where O^{2-} represents the oxygen anions in Yttria-Stabilised Zirconia (YSZ), $[O^{\delta-} - \delta^{+}]$ is the dipole of oxygen promoting species (BSS) and $2e^{-}$ denotes the number of electrons transfered in the metal.

The electrochemically induced promotion effect on a given catalytic reaction system can be quantified using three parameters (Vayenas et al., 1988, 1991, 1992):

1. The rate enhancement ratio, ρ:

$$\rho = r / r_0 \qquad (2.2)$$

where r is the electrochemically promoted rate and r_0 is the open-circuit (unpromoted) rate. Lower than unity values of ρ represent catalyst poisoning, while greater than this values demonstrate enhancement of catalytic performance.

2. The Faradaic efficiency or enhancement factor, Λ:

$$\Lambda = \Delta r /(I / nF) \tag{2.3}$$

where Δr is the NEMCA induced change $(r-r_0)$ in the catalytic rate, I is the applied current, n is the charge of the promoting species (n=2 for O^{2-}) and F is the Faraday constant. When the absolute value of Λ is greater than 1, the reaction is considered as electrochemically promoted, while lower or equal to 1 values, correspond to electrocatalysis (Faradaic contribution in the system). Furthermore, for oxidation systems (electron supply), Λ>1 and Λ<-1 denote 'electrophobic' and 'electrophylic' behaviours respectively, while the descriptions are reversed for hydrogenation systems (proton supply).

3. The promotional index, PI:

$$PI = \frac{\Delta r / r_0}{\Delta \theta} \tag{2.4}$$

where θ is the promoting species coverage on the catalytic surface. Promotion is demonstrated for PI>0, while PI<0 denotes surface poisoning (the promoting species acts as a poison for the catalytic reaction and not as a promoter).

The EPOC phenomenon was first observed by Stoukides and Vayenas in the early 1980s (Stoukides and Vayenas, 1981) and has since been generally admitted as a completely reversible effect, meaning that the electrochemically promoted catalytic

reaction rate returns to its open circuit (unpromoted) value under current interruption. Figure 2.2 depicts such a reversible system behaviour.

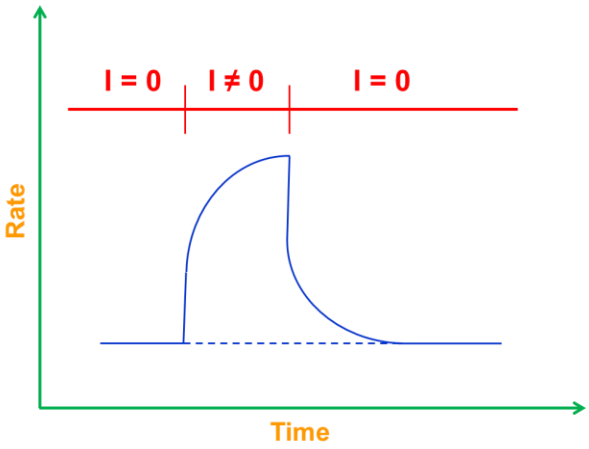


Figure 2.2: Schematic presentation of the reversible EPOC phenomenon.

Nevertheless, under certain conditions and upon current interruption, it was observed by Comninellis and co-workers in the late 1990s that the reaction rate could return to a different steady-state rate than the open circuit one (Varkaraki et al., 1995). Such an irreversible system behaviour, also termed as 'permanent electrochemical promotion' (P-EPOC), is illustrated in Figure 2.3.

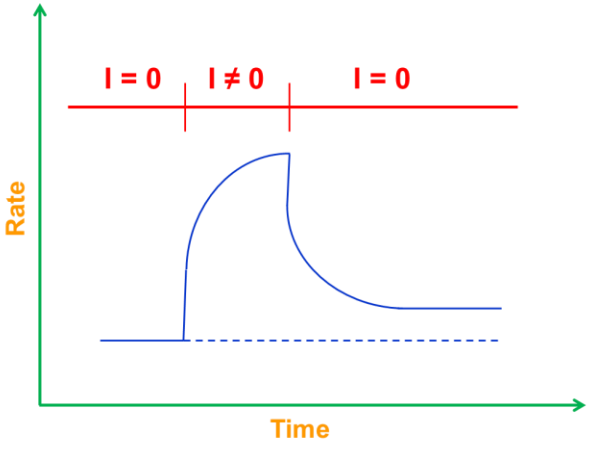


Figure 2.3: Schematic presentation of the permanent EPOC phenomenon.

A permanent rate enhancement factor, γ, was introduced to quantify the irreversible catalytic performance in an electrochemically promoted system (Nicole et al., 1997; Nicole and Comninellis, 1998; Jaccoud, 2007):

$$\gamma = r'/r_0 \tag{2.5}$$

where r_0 and r' stand for the open circuit rate before and after the application of potential, respectively.

As already mentioned, the EPOC phenomenon was first observed by Stoukides and Vayenas (Stoukides and Vayenas, 1981) in the early 1980s and has since been demonstarted in several catalytic systems (Vayenas et al., 1995, 2001a; Vayenas, 2013), using metal, dispersed metal (Marwood and Vayenas, 1998; Constantinou et al., 2007; Xia et al., 2010) or metal oxide (Nicole at el., 1997; Wodiunig and Comninellis, 1999) catalyst-electrodes, deposited on O^{2-} (Hong et al., 1996; Vernoux et al., 2007), Na^{+} (Palermo et al., 1996), H^{+} (Makri et al., 1996), K^{+} (Pitselis et al., 1997) and F^{-} (Yentekakis and Vayenas, 1994) ionic conductors, mixed ionic-electronic (Petrolekas et al., 1998; Poulidi et al., 2011) conductors, under atmospheric and high vacuum (Poppe et al., 1999; Xia et al., 2012) pressure conditions.

Although the electrochemical promotion phenomenon has been extensively investigated, exhibiting a great potential for practical applications, only few scaled-up commercial systems have been constructed so far (Yiokari et al., 2000; Balomenou et al., 2006; Ruiz et al., 2013), mainly due to the absence of systematic and accurate models that could lead to the evaluation of optimal design and operating conditions.

2.2 Experimental and modelling studies

In this section, a selection of the most important studies focusing on the experimental and theoretical aspects of the electrochemical promotion phenomenon will be discussed.

Stoukides and Vayenas (1981) have observed that the selectivity of a catalyst deposited on a solid oxide electrolyte can be altered dramatically, when application of external voltages occur at the cell. The selectivity and yield of oxidation of ethylene (C_2H_4) on polycrystalline silver (Ag) - at atmospheric conditions and temperatures around 400 ^oC - under both open and closed-circuit conditions were investigated. It was observed that O^{2-} *pumping*, from the solid electrolyte to the catalytic surface, was taking place when voltages were applied to the cell. It was also found that due to this electrochemically induced O^{2-} *pumping*, the production rate of ethylene oxide (C_2H_4O) was rised in a considerable and reversible manner.

Michaels and Vayenas (1984) studied the electrochemical oxidative dehydrogenation of ethylbenzene to styrene on a deposited on a solid electrolyte polycrystalline platinum (Pt) catalyst under atmospheric pressure conditions and temperatures near 575 ^oC. A simple kinetic model was constructed to describe the electrocatalytic process of interest under an external application of current. Two types of catalytic active sites were taken into consideration. One type of active sites was taken into account for oxygen dissociative adsorption on the air-side electrode, while the other type ones were used for the conversion of adsorbed oxygen to oxygen oxides on the fuel-side catalytic surface. These oxidised surface species could subsequently react with adsorbed ethylbenzene leading to an enhancement of the dehydrogenation rate to as much as 600%.

Yentekakis and Vayenas (1988) investigated the effect of the electrochemically arised O^{2-} *pumping*, to or from a Pt catalytic film, on the CO oxidation rate at temperatures between 250 and 600 ^{o}C and at atmospheric pressure. Although the Pt catalyst was found to operate typically under open-circuit conditions (Yentekakis et al., 1988), the circuit closure was found to result in a significant increase of 500% in the steady-state CO_2 production rate. Such an enhancement of the catalytic performance could not be justified by the Faradaic contribution on the system (normally altering the rate by a factor of 2). It was found that the O^{2-} *pumping* on the catalytic surface could induce changes in the work function of the catalytic film leading to pronounced alterations in the catalytic activity and consequently, to the observed non-Faradaic increases of the CO_2 production rate.

Vayenas et al. (1988) reported that the non-Faradaic effect was not limited to the ethylene, ethylbenzene and CO oxidation systems, but it was a general effect in closed-circuit heterogeneously catalysed systems. They have studied the CH_3OH oxidation on Pt at temperatures between 400 and 500 ^{o}C and found that a current application can lead to CO_2 production rate increases of the order of 1200%. The observed catalytic performance enhancement was for first time reported as non-Faradaic electrochemical modification of catalytic activity. An enhancement factor, Λ, was introduced to report whether the induced reaction rate increases were due to the Faradaid effect or not. The enhancement factor was defined as the fraction of the polarization-induced alteration in the observed reaction rate, over the rate of supply/removal of oxygen anions (Faradaic rate) to/from the catalytic surface. A value of unity for Λ suggests that the system

behaviour is purely Faradaic, while much greater than unity Λ values suggest that the behaviour of the system is mainly non-Faradaic.

In the next couple of years, the NEMCA effect was demonstrated for several catalytic reactions, i.e. CO, C_2H_4 oxidation on Pt, CO oxidation on Pd and C_2H_4 partial oxidation on Ag (Bebelis and Vayenas, 1989; Vayenas et al., 1989, 1990a). Vayenas et al. (1990a) observed that NEMCA effect is not limited to Yttria-Stabilised Zirconia electrolyte systems (O^{2-} conductors), but it can also been demonstarted in β''-Al_2O_3 electrolyte systems (Na^+ conductors). It was also found that the catalytic activity is exponentially dependent on the catalyst work function. To describe this dependence, the electrochemically catalysed reactions were categorised into two groups, the electrophilic and the electrophobic. The former ones were those whose rate increases exponentially with increasing work function value, while the rate of the latter ones exhibits the opposite effect.

Ladas et al. (1991) performed in situ work function measurements using a Kelvin probe in order to explore the origin of the NEMCA effect in systems where Pt was deposited on both Na^+ and O^{2-} solid electrolyte conductors. The term 'spillover' was proposed to describe the controlled migration of ions from the solid electrolyte to the catalytically active area. It was established that the work function of the catalyst can be altered just with a small coverage of ions (<5% of a monolayer) which was not enough to affect the coverage of the rest chemisorbed species. Moreover, it was found that the work function changes are directly dependent on the changes in the catalyst potential which in turn could be induced either by altering the gaseous phase composition or by application of potential in the system.

Yentekakis and Bebelis (1992) studied the NEMCA effect on the C_2H_4 oxidation over Pt/YSZ using a continuous flow, well mixed (CSTR) reactor at atmospheric conditions. In the single pellet type catalytic reactor utilised in that study, the entire pellet (including working, counter and reference electrodes) was exposed to the reactive gas mixture. Several patterns of possible working and counter electrode positions were taken into consideration. It was found that the NEMCA effect was demonstarted in all investigated cases, suggesting that the addressed effect is independent of the catalyst/counter electrode arrangement and subsequently, that it can be efficiently utilised in systems of conventional-type flow reactors.

Vayenas and co-workers (Vayenas et al., 1993, 1994; Vayenas and Bebelis, 1997) further investigated the NEMCA effect trying to explain its origin through providing evidence for the spillover species presence on the catalyst. Vayenas et al. (1993) have used galvanostatic transients to calculate the initial moment of a Na dipole on Pt and observed that the performed dipole measurements were in great agreement with the ones found in literature for Na/Pt(111). They have also confirmed the existence of spillover ions on the catalytically active surface, through the observation of O 1s signals in in-situ X-ray photoelectron spectroscopy (XPS) investigations of Ag electrodes. Vayenas et al. (1994) provided additional insights for the NEMCA effect through an updated description of the promotion phenomenon. They suggested that an effective electrochemical double layer was formed over the catalytic surface through the migration of spillover species (presented as dipoles of oxygen or sodium ions with their compensating charge in the metal) from the solid electrolyte. The established effective double layer could in turn affect the binding strength of the chemisorbed species leading

to an electrochemically induced promotion of the surface reaction rates. Vayenas and Bebelis (1997) termed the migration of ions from the electrolyte to the catalytic surface as 'backspillover, while the opposite effect was termed as 'spillover'. Using XPS, they provided significant evidence that the backspillover species, which were generated under positive current application, were less reactive than the normally chemisorbed atomic oxygen, i.e. dissociatively adsorbed gas phase oxygen.

Yentekakis et al. (1994) studied the effect of the electrochemically generated Na on the surface reaction rate using the CO oxidation over Pt/β"-Alumina as an illustrative system. It was observed that sodium can act both as a promoter and as a poison for the CO electrocatalytic oxidation. More specifically, for Na coverages lower than 0.04, sodium was acting as a promoter increasing the CO_2 production rate up to 600%, while for greater than 0.04 Na values, sodium was acting as poison decreasing the observed CO_2 production rate up to 90%. They suggested that the poisoning effect was due to the development of a complex between CO, Na and Pt.

Harkness and Lambert (1995) studied the electrochemically promoted ethylene oxidation by NO. The utilised system comprised of a Pt catalyst deposited on a β"-Alumina (Na^+ conductor) solid electrolyte in a CSTR reactor. They observed that the presence of a sodium compound was affecting the adsorption enthalpies and the activation energies of the catalytic surface reactions, acting as a promoter in a rather efficient manner. It was also found through XPS investigation that the Na compound was potentially a carbonate and not an oxide, due to the absence of any measured shifts in the Na 1s emission.

Karavasilis et al. (1996) studied the NEMCA effect on the ethylene epoxidation on Ag catalytic films deposited on a Na^+ solid electrolyte conductor, in the presence of dichloroethane. It was found that the epoxidation rate was enhanced for sodium coverages up to 0.03. Additionally, for Na coverage value of 0.03, the selectivity to ethylene oxide was obtained to be 88% in the presence of chrorine moderator (1ppm). Furthermore, the promotional index, *PI*, was measured to be as high as 40 demonstarting how strong the sodium promoting action was.

Marwood and Vayenas (1998) were the first to investigate the EPOC phenomenon using a dispersed catalyst. The oxidation of ethylene over a deposited on YSZ, dispersed Pt on Au, catalyst was the system of choice. It was shown that the NEMCA effect on the dispersed catalyst was demonstrating similar behaviour to the earlier electrochemical promotion studies where pure (continuous) metal catalysts were of use. Consequently, they suggested that the connection between the active catalyst and the solid electrolyte did not necessarily have to be direct as long as there was an inert metal between them, in order to assist the charge conservation.

Wodiunig and Comninellis (1999) studied the NEMCA effect on the C_2H_4 combustion over a metal oxide catalyst, RuO_2, deposited on YSZ solid electrolyte. Using temperature programmed desorption (TPD) of oxygen from the catalytic surface, they found that the O_2 desorption activation energy was getting reduced by rising the applied potential difference in the cell. The observed desorption activation energy alterations were found to affect the catalytic activity in a reversible and controllable manner.

Poppe et al. (1999) investigated the electrochemically promoted CO combustion on Pt/YSZ under ultra hign vacuum (UHV) conditions. Two differently prepared types of Pt/YSZ samples were used for the investigation. They have found that using one type of samples led to insignificant increases in the surface reaction rate which were only limited to the Faradaic contribution in the system. They observed that in this case, the backspillover species did not diffuse over the catalytic surface but it remained located in the close neighbourhood of the triple phase boundaries (where they were generated from). On the other hand, using another type of samples, they observed that the backspillover species could migrate over the catalytic surface inducing work function alterations. Having observed this, they suggested that the NEMCA effect is strongly dependent on the sample preparation.

Lambert et al. (2000) studied the EPOC phenomenon using alkali ion (Na^+, K^+) solid electrolyte conductors. The catalytic reduction of NO by CO and by propene over Cu, Rh and Pt was used as an illustrative example. They found that the chemical state of an alkali promoter depends on the reactive gas phase composition. Furthermore, it was observed that 100% nitrogen formation selectivity could be exhibited during NO reduction when Rh was used as catalyst and under closed-circuit conditions. It was also found that K^+ promoters induced an increase of the ethylene desorption rate, while Na^+ ones seemed to cause a NO dissociation.

Yiokari et al. (2000) were the first to study the electrochemically induced catalytic performance modification using a commercial scaled-up catalytic system consisting of 24 pellets. The ammonia synthesis over iron (industrial) catalytic films deposited on a proton conductor ($CaIn_{0.1}Zr_{0.9}O_{3-R}$) electrolyte was the reaction system of interest. The

NEMCA effect was demonstarted under low temperature (440 ^{o}C), high pressure (50 atm) and negative polarisation (-1.0 V) conditions, while permanent poisoning was observed in the case of positive potential application. More specifically, it was found that using an optimal gaseous composition (P_{H2}/P_{N2} ratio of the value of 0.5) could lead to an enhancement of the catalytic performance of up to 1300%.

Pliangos et al. (2000) were the first to demonstrate the effect of electrochemical promotion on an already chemically Na-promoted reaction. They investigated the NO reduction over Rh/YSZ via dry impegnation with NaOH and found that the NO reduction rate was further increased by up to a factor of 4 under circuit closure. It was also found that the chemically Na-promoted catalytic films exhibited a decrease in their light-off temperature from 440 to 320 ^{o}C and a further decrease to 260 ^{o}C under positive potential application.

Metcalfe (2001a) developed a model to describe the electrochemistry behind the NEMCA effect. Two types of oxygen, i.e. neutral and ionic, were taken into consideration in the proposed model. Oxygen anions was the only ionic species taken into account and its cleavage on the catalytic surface was considered to be significant in order to serve the double layer operation. Using thermodynamic considerations between the chemical potentials of the oxygen species present on the catalytic surface and neglecting possible interactions between the ionic ones, it was found that the applied overpotential in the system was leading to an equivalent change in the potential of the catalyst. Metcalfe (2001b) continued his study extending the developed model to take into consideration the lateral (ionic oxygen) dipole-to-dipole interactions. It was shown that the surface potential change was lower than the applied overpotential in the case of

strong lateral interactions. Furthermore, raise in the reaction rate was observed under the application of both positive and negative overpotentials.

Vayenas and Pitselis (2001) formulated a steady-state, 1-D surface reaction-diffusion model to elucidate the electrochemical promotion effect on a porous catalyst deposited on a solid oxide electrolyte. The backspillover species was the only species taken into account in this modelling study. It was generated from the triple phase boundary (here assumed as a 0-D interface layer between the catalyst and the support) and subsequently, it was considered either to react with the gas phase species or to diffuse through the catalytic film. The proposed model was used for reaction-diffusion simulations in a system of dispersed supported nanocrystalline catalytic particles. High backspillover species coverage on the nano-particle surface was found, signifying that the dominant mechanism in such a system is the diffusion. The model was further exploited in Presvytes and Vayenas (2007) to investigate the NEMCA effect in a system comprising of semi-spherical Nicket-cermet particles deposited on larger YSZ ones. The whole thickness of the catalytic particle was taken into account here to represent the TPB area. An effectiveness factor was utilised to examine the catalytic performance in systems of various fractions of spherical particles. It was found that the catalytic performance was practically not affected by the geometry of the catalytic particle, on agreement with experiments.

Foti et al. (2002) developed a model to investigate the reaction rate transients of an electrochemically promoted catalytic system. Fast diffusion of backspillover species, which could form on the catalyst surface if and only if there were any active surface sites available, as well as a first order consumption rate for BSS were taken into

consideration. It was found that the coverage of BSS on the catalytic surface was exponentially dependend on the polarisation time and that the BSS steady-state coverage value, as well as its accumulation on the catalytic surface rate were increasing functions of the applied current.

Vayenas et al. (2003) investigated the electrochemically induced surface modifications of single crystal Pt surfaces using scanning tunneling microscopy (STM). It was observed that when the BSS migrated on a clean cayalytic surface, it could rapidly transform to normally chemisorbed atomic oxygen. Furthermore, it was found that the two types of oxygen surface species, i.e. ionic and dissociatively adsorbed oxygen, were completely distinct due to the significant difference in their chemical potentials. It was also shown that the BSS could diffuse on the catalytic surface for greater than mm distances and consequently, that the effect of electrochemical promotion could take place in commercial nm sized catalytic films.

A novel monolithic electropromoted reactor (MEPR) was constructed by the group of Vayenas (Balomenou et al., 2004, 2006; Tsiplakides et al., 2005) comprising of thin porous catalytic films (flat or ribbed) covering both sides of solid electrolyte parallel plates supported in appropriately shaped trenches positioned on the inside surfaces of the walls of the ceramic monolithic reactor. This novel MEPR was utilised to investigate the effect of electrochemical promotion on the hydrocarbon oxidation and the NO reduction by C_2H_4 reaction systems. It was found that MEPRs exhibited non-Fararaid performance even at high conversions of the reactants. Furthermore, MEPRs were found to successfully operate at gas flowrates of the same magnitude as the ones in typical exhaust treatment units enhancing the industrial potential of such systems.

Baranova et al. (2004) studied the methane oxidation on a Rh catalyst deposited on YSZ coated with a thin layer of TiO_2. Two surface states could be observed on the Rodium surface. One state was inactive and corresponded to rhodium oxide and the other one was active and corresponded to rhodium metal. It was found that electrochemical promotion alters the work function of the catalyst by inducing a weakening of the Rh-O chemisorptive bond and subsequently leading to reduction of the rhodium oxide. Baranova et al. (2005) continued their studies employing the same catalyst/support system for the investigation of the ethylene oxidation. It was found that the TiO_2 layer could cause an enhancement in the catalytic activity under both open- and closed-circuit conditions (of the order of 10% in the latter case) and that it could also stabilize the catalyst state under current or potential application.

Koutsodontis et al. (2006) investigated the dependence of the NEMCA effect on the thickness of a porous catalytic layer using the ethylene oxidation over Pt/YSZ as a reaction system. Increasing the catalytic film thickness from 0.2 μm to 1 μm was found to induce a gradual decrease in the rate enhancement ratio from 400 to 50, due to subsequent decrease of the promoting species coverage from the TPB interfaced side to the gas exposed side of the catalytic film.

Poulidi et al. (2007) approached the NEMCA effect using a novel wireless configuration in a dual-chamber membrane reactor. More specifically, they have utilised a mixed protonic-electronic ($Sr_{0.97}Ce_{0.9}Yb_{0.1}O_{3-\delta}$) conductor to supply hydrogen promoting species on the catalytic surface, i.e. the reaction side of the dual chamber. This was accomplished by adjusting the difference of the hydrogen chemical potential throughout the reactor. The manipulation of the hydrogen chemical potential was carried

out using 'sweep' gas, at the sweep side of the dual chamber, consisting of helium, oxygen in helium and 5% hydrogen in nitrogen. It was found that the C_2H_4 oxidation rate was increased by a factor of 1.6 only in the case of using H_2 as a sweep gas (5% H_2 in N_2) and under certain operating conditions. The observed reaction rate enhancement was found to be completely reversible. Poulidi et al. (2011) continued their wireless configuration investigations using a mixed ionic-electronic ($La_{0.6}Sr_{0.4}Co_{0.2}Fe_{0.8}O_{3-\delta}$) conductor in several single pellet systems. Enhancement in the catalytic performance was observed for all utilised sweep gas systems. However, when oxygen was used as sweep gas, the electrochemiclly promoted ethylene combustion rate was observed not to return to its unpromoted state after sweep gas interruption, and thus the system in this case was not completely reversible.

De Lucas-Consuegra et al. (2008) were the first to study the electrochemically promoted catalytic reduction of N_2O by C_3H_6 over Pt/K- βAl_2O_3. It was observed that although the N_2O reduction rate was significantly increased at low temperatures due to the presense of K^+ promoter on the catalytic surface, increasing the gaseous concentration of oxygen, induced surface poisoning (due to relatively high oxygen coverage) and to consequent rate decreases. Furthermore, it was found that the presence of K^+ promoting species, reduced the adsorption of water on the catalyst as a result to increase its performance.

Leiva et al. (2008) performed DFT Monte Carlo simulations in the Grand Canonical ensemble to investigate the modification of the catalytic surface under the effect of the electrochemically induced double layer. It was found that oxygen and sodium promoting species had different effect on the catalytic lattice. The former seemed to induce a

compression of the catalytic lattice due to transport of electrons to the electrolyte, while the latter was found to generate an expansion of the lattice as a result of the opposite effect.

Sapountzi et al. (2009) investigated the EPOC effect on the CO combustion rate in a proton exchange membrane (PEM) fuel cell reactor with Pt and Au anode electrodes and feeding the cathode with either air or hydrogen. It was found that the CO_2 production rate was enhanced when air was fed at the cathode due to direct oxidation of CO with the excorporated from the membrane oxygen. On the contrary, the supply of H_2 at the cathode was found to trigger the water-gas-shift reaction limiting its rate only to the Faradaic contribution. Furthermore, gold was found to achieve more pronounced conversion rate enhancement than Pt. More specifically, it was observed that the electrochemically promoted Au performance was more significant in systems of CO-rich gas phase at the anode and O_2 supply at the cathode.

Jaksic et al. (2010) were the first to investigate the effect of electrochemical promotion on the oxidation of formaldehyde using mixed hypo-d-oxide ($Pt/H_{0.35}WO_3/TiO_2/C$, $Pt/H_xNbO_3/TiO_2/C$) nano-structured catalysts. Cyclic voltametry was employed to confirm the existence of (Pt-OH dipole) promoting species on the electrochemically active catalytic surface. It was found that the NEMCA effect was enhancing the catalytic performance for more than an order of magnitude and that the promotion effect in such systems was fully reversible.

Xia et al. (2010) employed a novel bipolar configuration of nano-dispersed Pt particles deposited on YSZ to study the electrochemically promoted CO combustion.

Current application was found to induce a strong non-Faradaic behaviour enhancing the CO_2 production rate as much as 500. The NEMCA effect in such a configuration was found to be permanent under current interruption.

Mutoro et al. (2010) were the first to employ a thin epitaxial Pt(111)/YSZ(111) system to study the electrochemically induced enhancement in the ethylene oxidation rate. Two types of Pt samples, with and without iron oxide dopants, were utilised to investigate the electrochemical promotion effect. Iron-free catalytic surfaces were found to exhibit a reversible EPOC effect only when the triple phase boundaries length was relatively high. On the contrary, catalytic films covered by FeO_x demonstrated the NEMCA effect in a permanent manner, while the TPB length seemed not to affect the observed promotion effect. The iron oxide dopants on the catalytic surface was suggested that either act as an oxygen reservoir or affect the catalyst microstructure leading to the illustrated differences.

Caravaca et al. (2011) employed a novel configuration of a solid electrolyte single chamber steam electrolysis cell coupling the electrolysis process and the EPOC effect in order to achieve higher H_2 productivities. Negative current application in the system was found to reduce the concentration of water, forming hydrogen and adsorbed on the catalytic surface oxygen anions. The generated oxygen promoting species served catalyst work function alterations leading to (NEMCA) increases in the methane steam reforming rate and to subsequent rises in the H_2 production rate.

Hammad et al. (2011) continued their monolithic electropromoted reactor studies through investigating the NEMCA effect on the oxidation of SO_2. When the MEPR was

loaded with five plates and under certain conditions and upon application of positive current, i.e. supply of promoting species to the catalytic layer, the SO_3 production rate was found to increase as much as 200%, while no enhancement was observed under negative polarisation. Fully occupying the MEPR with 22 plates induced an increase in the SO_2 conversion from 50% (open-circuit conditions) to 60% (positive potential application). Furthermore, it was found that increasing the gas flowrate led to reduced SO_3 production rates, while increasing the applied potential value caused catalytic surface poisoning with promoting species, leading to decreased SO_2 convesrion.

Jiménez-Borja et al. (2012) were the first to investigate the electrochemically induced promotion on the low-temperature natural gas combustion over Pd/YSZ. The NEMCA effect was demonstrated under both positive and negative polarisation. Upon positive potential application and temperatures as low as 340 °C, CO_2 production rate increases up to 250 % were observed. It was shown that the presence of O^{2-} species on the catalytic surface could modify its properties through strengthening the Pd-hydrocarbon bonds and that this modification resulted in the observed enhancement. Furthemore, upon negative potential application, it was found that propane and methane conversion was electrochemically enhanced due to the partial reduction of Pd-O.

Huang et al. (2012) developed a 2-D kinetic model to describe the partial oxidation of n-pentadecane on an electrochemically promoted catalytic system. The proposed model comprised of a model based on Langmuir-Hinshelwood kinetics for the simulation of the typical partial oxidation reforming, coupled with an electrochemical process serving the charge transfer throughout the system. The model was employed in conjunction with experimental data to demonstrate the impact of the temperature on the

conversion of the fuel. It was found that the computational predictions exhibited a very good agreement with the experimental observations for the range of temperature values of 450 to 650 °C, also quantifying the electrochemically induced enhancement of the fuel conversion.

Matei et al. (2013) employed a novel catalyst/support system, comprising of Pd deposited on highly porous YSZ solid electrolyte, for the investigation of the effect of electrochemical promotion on the combustion of methane. Under open-ciruit conditions, the palladium catalysts deposited on highly porous YSZ electrolyte were found to exhibit up to one order of magnitude higher performance than the ones deposited on dense solid electrolyte. Upon positive polarisation, the catalytic performance was further improved leading to a CO_2 production rate increase up to 50%. The observed electrochemical enhancement was found to be permanent under fuel rich conditions due to the presense of surface oxide covering the Pd surface. An enhancement in the methane combustion rate was also observed in the case of negative potential application and temperatures higher than 400 °C.

Ibrahim et al. (2013) investigated the electrochemically promoted oxidation of ethylene over Pt/YSZ under the additional influence of catalytic surface impurities (Na). It was shown that sodium presence on the catalytic film could cause catalytic performance enhancement under open circuit conditions. The NEMCA effect was also demonstrated under closed-circuit conditions but there was no evidence that the observed catalytic enhancement was due to the presense of impurities on the catalytic surface. Nevertheless, it was shown that the sodium coverage could significantly affect the system behaviour. More specifically, the system behaviour could change from

electrophilic to volcano-type for increasing Na coverage, while electrophobic system behaviour was favoured for high coverage of sodium.

Ruiz et al. (2013) investigated the EPOC effect on the CO_2 hydrogenation to renewable fuels over Pt/K-βAl_2O_3 in a tubular bench scale system. Two catalyst configurations with different particle sizes and metal dispersion have been used in that study. It was found that the CO_2 hydrogenation rate was enhanced by as much as 20 times due to the presence of K^+ promoting species on the catalytic surface. Furthermore, using a catalyst with smaller particle size led to an enhancement of the CO_2 conversion to CH_4 by up to 7.6 times, while the use of a less dispersed catalyst seemed to enhance the conversion to C_2H_5OH and CH_3OH by up to 16 and 27 times, respectively.

2.3 Contribution

Although the NEMCA effect has extensively been investigated experimentally and a great commercial potential has been arised with the construction of a monolithic electrochemical promoted reactor (Balomenou et al., 2004), only few modelling studies have been performed to describe and simulate the addressed phenomenon (Metcalfe, 2001a, 2001b; Vayenas and Pitselis, 2001; Foti et al., 2002; Presvytes and Vayenas, 2007; Leiva et al., 2008; Huang et al., 2012).

Metcalfe (2001a) formulated a model comprising of algebraic equations to study the electrochemistry upon the NEMCA effect. In the proposed model only two species, i.e. adsorbed oxygen and oxygen anions, were taken into consideration. The algebraic equations were used to calculate the induced change in the chemical potential of the

adsorbed oxygen species and of the catalyst under an overpotential application in the system. Metcalfe (2001b) extended his previous study to take into account the lateral (ionic oxygen) dipole-to-dipole interactions. The proposed models were capable of predicting the steady state behaviour of the system.

Vayenas and co-workers (Vayenas and Pitselis, 2001; Presvytes and Vayenas, 2007) developed a steady-state 1-D surface reaction-diffusion model to simulate the electrochemical promotion effect on a porous catalyst deposited on a solid oxide electrolyte. The proposed model comprised of an 1-D reaction diffusion partial differential equation (PDE) which was solved only for the backspillover species. For the solution of the PDE, inward flux of BSS was imposed at the triple phase boundary and zero flux was considered elsewhere. Although the proposed model was the first attempt to simulate the NEMCA effect taking into account the spatial variation, it considered BSS as the only surface species. Furthermore, the current in the system was computed by an algebraic equation and not by a charge conservation PDE equation.

Foti et al. (2002) formulated a dynamic model to evaluate the rate of an electrochemically promoted reaction. The proposed model comprised of an ordinary differential equation (ODE) for the calculation of the BSS coverage on the catalytic surface, while first order consumption rate for BSS was considered. Moreover, the BSS production was computed by a Faradaic term. The Faradaic term was considered as a function of the current in the system, which was calculated externally solving the Laplace equation. Consequently, the reaction rate was expressed as a function of the computed BSS coverage and of a proportional factor. The proportional factor was estimated using experimental data under steady state conditions. Although a validated

model was obtained, the diffusional effects of the BSS, which in turn cause the alteration of the catalyst work function and subsequently, the catalytic performance modification, were not taken into account.

Leiva et al. (2008) carried out density functional theory (DFT) calculations and grand canonical Monte Carlo simulations to investigate the modification of the catalytic surface under the effect of the electrochemically generated effective double layer. It was found that oxygen and sodium promoting species had different effects on the catalytic lattice. The former seemed to induce a compression of the catalytic lattice due to transport of electrons to the electrolyte, while the latter was found to generate an expansion of the lattice as a result of the opposite effect.

Huang et al. (2012) constructed a 2-D macroscopic steady-state model to describe the electrochemically promoted partial oxidation of n-pentadecane. The proposed framework comprised of a Langmuir-Hinshelwood kinetic model used to calculate the unpromoted partial oxidation reforming rate coupled with a 2-D model used to simulate the charge conservation in the system and the electrochemically induced n-pentadecane production rate. For the calculation of the electrochemically promoted rate, the promotion rules proposed by Vayenas and Brosda (2002) were utilised. The coupling of the two models was achieved through the summation of the unpromoted and the promoted rates. The L-H and the closed-circuit model parameters were estimated using open and closed circuit experimental data, respectively.

In 2011, we were the first to develop a 2-D macroscopic model to describe the electrochemical promotion phenomenon (Bonis et al., 2011). The proposed framework

was used to perform reaction-diffusion as well as electro-catalytic simulations for the electrochemically promoted CO combustion over Pt/YSZ. Nevertheless, only qualitatively results were obtained in this study, due to unavailability of closed circuit rate kinetic constants in literature.

One year later, a multi-scale approach to investigate the NEMCA effect in a solid oxide single pellet system was developed (Fragkopoulos et al., 2012). The oxidation of CO over Pt deposited on YSZ was used as an illustrative system. The proposed framework was integrated by a macro-scopic model which employed the finite elements method to simulate the electrochemical phenomena throughout the pellet and a micro-scopic model for the simulation of the reaction-diffusion micro-processes using the kinetic Monte Carlo (kMC) method. The multi-scale framework resulting species average coverage profiles were compared against the ones obtained by a fully macro-scopic model and it was found that both modelling studies resulted in similar dynamic trends but with quantitative differences.

In 2013, a macroscopic, multi-dimensional, isothermal, dynamic solid oxide single pellet model was formulated, to analyse the chemical and electrochemical processes taking place in an electrochemically promoted CO combustion system and to demonstrate the EPOC effect in such a system upon potential application (Fragkopoulos et al., 2013). The proposed model coupled mass transport with charge transport processes through the electrochemical phenomena taking place at the TPBs of anode and cathode electrodes. Parameter estimation was undertaken using the proposed model in conjunction with experimental data from the literature and a validated model was obtained. It was found that the effect in such a system was strongly non-Faradaic.

Subsequently, changes in the chemical kinetic parameters were found to lead to great alterations in the CO_2 combustion rate, while modifying the electrochemical kinetic parameters had almost no effect on the catalytic performance.

The same year, we extended our multi-scale framework to perform simulations for electrochemically enhanced catalytic systems with large catalytic surfaces (Fragkopoulos and Theodoropoulos, 2013). The framework here was integrated by a macroscopic model employing the finite elements method (FEM) for the simulation of charge transport in the system and a micro-scopic one using a lattice kMC simulator and the Gap-Tooth method for the simulation of the catalytic surface reaction-diffusion micro-processes. The FEM/kMC/gap-tooth framework was compared against a FEM/kMC one and it was found that the extended model could very accurately capture the surface dynamics with computational efficiency.

We believe that such detailed models aided by high-fidelity experimental data will enhance our understanding on the electrochemical promotion phenomenon and they will ultimately enhance its industrial potential.

Chapter 3

Macroscopic Modelling of EPOC Systems

3.1 Introduction

The subject of this chapter is the development of accurate macroscopic models for electrochemically promoted catalytic systems, i.e. systems where the catalytic performance is enhanced by application of potential between the working and counter electrodes in the solid oxide single pellet. As described in the previous chapters, the EPOC phenomenon was first observed by Stoukides and Vayenas (1981) and has since been broadly investigated experimentally. Nevertheless, only few modelling frameworks have been formulated illustrating the addressed effect (Vayenas and Pitselis, 2001; Foti et al., 2002; Presvytes and Vayenas, 2007; Leiva et al., 2008; Huang et al., 2012).

To the best of our knowledge, Fragkopoulos et al. (2013) were the first to formulate a systematic multi-dimensional modelling framework, to simulate the chemical and electrochemical phenomena taking place in electrochemically catalyzed systems. We believe that such a model can enrich our understanding for EPOC systems and consequently it will allow robust EPOC system design rising the 'scaled-up' industrial potential.

This chapter presents the study performed by Fragkopoulos et al. (2013), where we have developed a macroscopic, multi-dimensional, isothermal, dynamic solid oxide single pellet model, to describe the chemical and electrochemical phenomena taking place in an electrochemically enhanced CO combustion system and to address the NEMCA effect in such a system upon polarisation application. The proposed model couples mass transport (catalytic surface reaction-diffusion) with charge transport (throughout the pellet) as well as with electrochemical reactions taking place at the TPBs of anode and cathode electrodes. Parameter estimation is performed using the proposed framework in conjunction with experimental data from the literature and a validated model is obtained. Subsequent sensitivity analysis of the estimated parameters is carried out to evaluate the effect of the percentage change of each parameter on the reaction rates. Finally, the effects of the gaseous reactants' partial pressures as well as of the temperature on the CO_2 production rate are further investigated and conclusions from the above analysis are discussed.

3.2 The chemical process

The electrochemically promoted CO oxidation over Pt/YSZ will be used to demonstrate the detailed model developed in this chapter, since CO oxidation on Pt is one of the most extensively investigated heterogeneous catalytic systems. In this section, the CO oxidation mechanism as well as the surface mass balances of each species taking part in this catalytic process are presented. Figure 3.1 illustrates a schematic of the CO oxidation surface dynamics.

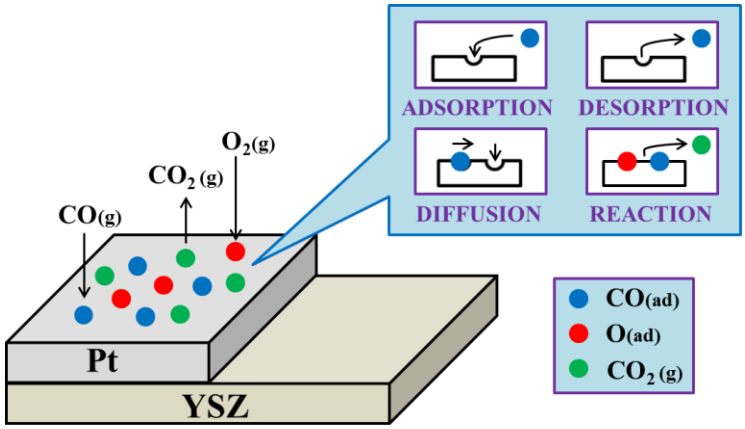


Figure 3.1: Schematic presentation of CO oxidation on Pt surface.

3.2.1 CO oxidation mechanism

Several studies have shown that the CO oxidation on Pt follows the Langmuir-Hinshelwood mechanism (Kaul and Wolf, 1985; Palmer and Smith, 1974). This mechanism comprises of a surface reaction between adsorbed CO and O (Rxn (3.3)) producing CO_2 (see Figure 3.1). Once CO_2 is formed, it desorbs directly in the gas phase and its readsorption on the catalytic (Pt) surface is assumed to be negligible (Matsushima, 1978). Adsorption of O_2 (Rxn (3.1), forward) and of CO (Rxn (3.2), forward) on the catalytic surface takes place through dissociation (Ducros and Merrill,

1976) and without dissociation (Heyne and Tompkins, 1966), respectively. The corresponding mechanism is presented below (Rxns (3.1)-(3.3)) (Kaul et al., 1987):

$$O_{2\,(g)} + 2\cdot S \underset{k_{-1}}{\overset{2k_1}{\rightleftharpoons}} 2O\cdot S \tag{3.1}$$

$$CO_{(g)} + \cdot S \underset{k_{-2}}{\overset{k_2}{\rightleftharpoons}} CO\cdot S \tag{3.2}$$

$$O\cdot S + CO\cdot S \overset{k_3}{\rightarrow} CO_{2\,(g)} + 2\cdot S \tag{3.3}$$

When potential is applied between the anode and the cathode electrodes of the solid oxide single pellet, backspillover species are produced at the triple phase boundaries of the anode and diffuse over the Pt surface (as in Figure 3.2). BSS can either react with the co-adsorbed on the catalytic surface CO forming CO_2 (Rxn (3.4)) or desorb in the gas phase as O_2 (Rxn (3.5)) (Vayenas and Pitselis, 2001):

$$BSS\cdot S + CO\cdot S \overset{k_4}{\rightarrow} CO_{2\,(g)} + 2\cdot S \tag{3.4}$$

$$2BSS\cdot S \overset{k_5}{\rightarrow} O_{2\,(g)} + 2\cdot S \tag{3.5}$$

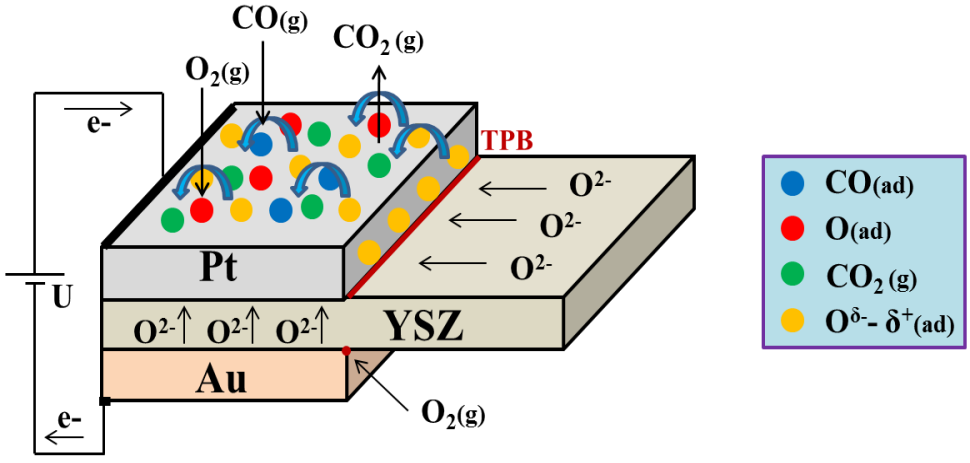


Figure 3.2: Schematic presentation of BSS formation and "migration" on Pt.

3.2.2 Surface mass balances

The surface mass balances for atomic O_2, CO and BSS are described by Eq. (3.6), Eq. (3.7) and Eq. (3.8), respectively. It should be noted that the surface reaction (3.3) is not considered to be the rate determining step and that the adsorption/desorption reactions (3.1) and (3.2) are not assumed to be at equilibrium. The conservation of active catalytic sites can be expressed by Eq. (3.9). Hence,

$$R_O^{'} = \frac{d\theta_O}{dt} = r_1 - r_{-1} - r_3 \tag{3.6}$$

$$R_{CO}^{'} = \frac{\partial\theta_{CO}}{\partial t} = r_2 - r_{-2} - r_3 - r_4 \tag{3.7}$$

$$R_{BSS}^{'} = \frac{d\theta_{BSS}}{dt} = -r_4 - r_5 + r_{elec} \tag{3.8}$$

$$\theta_S + \theta_O + \theta_{CO} + \theta_{BSS} = 1 \tag{3.9}$$

where $R_i^{'}$ is the overall production/consumption rate for species i, θ_j is the coverage of species j on the catalytic surface (dimensionless), $r_{k/-k}$ is the reaction rate for reaction *k*/-*k* (Rxns (3.1) to (3.5), *k* refers to the forward reactions while -*k* refers to the backward ones), and r_{elec} is the BSS Faradaic production rate which is described later in this chapter (see Eq. (3.40)). The individual reaction rates, $r_{k/-k}$, are described as (Kaul et al., 1987):

$$r_1 = 2k_1 C_{O_2} \theta_S^{\,2} \quad \& \quad r_2 = k_2 C_{CO} \theta_S \tag{3.10}$$

$$r_{-1} = k_{-1} \frac{\theta_O^{\,2}}{\left(1-\theta_O\right)^2} \quad , \quad r_{-2} = k_{-2}\theta_{CO} \quad \& \quad r_5 = k_5 \frac{\theta_{BSS}^{\,2}}{\left(1-\theta_{BSS}\right)^2} \tag{3.11}$$

$$r_3 = k_3\theta_O\theta_{CO} \quad \& \quad r_4 = k_4\theta_{CO}\theta_{BSS} \tag{3.12}$$

where $k_{i/-i}$ is the rate constant for *forward/backward* reaction *i*, C_{O_2} and C_{CO} are the gaseous concentrations (in mol/m^3 units) of O_2 and CO respectively. Assuming that the gas phase in the reactor is well-mixed, the gaseous concentrations of O_2 and CO are calculated using the ideal gas equation (Poling et al., 2001):

$$C_j = \frac{P_j}{RT}\ , \quad \mathrm{j} = O_2, CO \tag{3.13}$$

where P_j is the partial pressure for species j in the gas phase, R is the ideal gas constant and T is the temperature in the gas phase.

It should be noted here that the description of r_1 in Eq. (3.10), which has been taken from Kaul et al. (1987), is unconventional due to the 2 factor and corresponds exactly to the rate of coverage of the empty active sites due to reaction (3.1). Hence, for consistency, in reaction (3.1) we have used $2k_1$ as the forward rate constant instead of k_1. The terms $(1 - \theta_O)^2$ and $(1 - \theta_{BSS})^2$ are taken into account in the denominators of atomic O_2 and BSS desorption rates (Eq. 3.11)) respectively, in order to limit the saturation coverage (Gorte and Schmidt, 1978; Kaul et al., 1987). It is worthwhile to mention here that the expressions for r_{-1} and r_5 in Eq. (3.11) (also taken from Kaul et al. (1987)), essentially uses the ratio of the coverage of O and BSS respectively to the coverage of all the other species on the surface (including the empty sites). This ensures that the corresponding rates are finite since only the improbable case of 100% O (or BSS) coverage would lead to an infinite rate (note that the ratio of O and BSS coverage, respectively, to the fraction of empty sites, θ_s, would lead to an infinite rate in the more probable case of a full surface occupied by O, CO and BSS species).

3.2.2.1 Reaction rate constants

The rate constants for adsorption for both oxygen and carbon monoxide are assumed to be sticking coefficient dependent (Eq. (3.14)), while the ones for desorption and for the surface reaction are considered to follow an Arrhenius expression (Eq. (3.15)). Thus, the individual rate constants can be given from:

$$k_i = \frac{S_j}{N_S}\left(\frac{RT_S}{2\pi M_j}\right)^{1/2}, \quad i=1\ (j=O_2) \quad \& \quad i=2\ (j=CO) \tag{3.14}$$

where S_j is the sticking coefficient, M_j is the molecular weight of species j, T_s is the temperature of the catalytic surface and N_s (in mol/m^2 units) is the concentration of active sites on the catalytic surface. Furthermore,

$$k_i = k_{o,i}\exp\left(\frac{-E_{A,i}}{RT_S}\right), \qquad i=\text{-}1,\text{-}2,\ 3 \tag{3.15}$$

where $k_{o,i}$ and $E_{A,i}$ are the pre-exponential factor and the activation energy for each reaction rate, respectively. The sum of the consumption rate constants of backspillover species is constrained and considered to be equal to 10^{-2} s^{-1} (Vayenas and Pitselis, 2001):

$$k_4 + k_5 = 10^{-2}\ \mathrm{s}^{-1} \tag{3.16}$$

3.3 The electrochemical process

3.3.1 Electrochemical reactions

At the TPBs of the cathode, the electrochemical reduction of O_2 occurs:

$$3\times\left[\frac{1}{2}O_{2\ (g)} + 2e^{-} \rightarrow O^{2-}_{YSZ}\right] \tag{3.17}$$

At the anodic TPBs, three electrochemical reactions can take place, i.e. oxygen anions can either be excorporated from YSZ as O_2 (Rxn (3.18)) and as BSS (Rxn (3.20)), or react with CO forming CO_2 (Rxn (3.19)) (Vayenas et al., 2001a):

$$O_{YSZ}^{2-} \rightarrow \frac{1}{2} O_{2\,(g)} + 2e^- \tag{3.18}$$

$$O_{YSZ}^{2-} + CO_{(g)} \rightarrow CO_{2\,(g)} + 2e^- \tag{3.19}$$

$$O_{YSZ}^{2-} \rightarrow BSS + 2e^- \tag{3.20}$$

The rate of the cathodic reduction of O_2 (Rxn (3.17)) is equal to the sum of the rates of the anodic electrochemical reactions (3.18) to (3.20), so that the number of electrons consumed by the electrochemical reactions at the cathode TPBs is equal to the ones produced from the electrochemical reactions at the anode TPBs, hence the factor of 3 in reaction (3.17). Equivalently, we could ensure reactions (3.18) to (3.20) produce a total of 2 electrons by multiplying with the appropriate stoichiometric coefficients. In this case we would be able to omit the factor of 3 in reaction (3.17).

3.3.2 Electrochemical and equilibrium potentials

An electrochemical potential ($\overline{\mu_i} = \mu_i + z_i F \Phi_i$) is defined as the work needed in order to bring 1 mole of an anion from a standard state to a specific electrical potential and concentration (Bard and Faulkner, 2001). The electrochemical potentials of the reacting species have to be balanced at equilibrium (Chen et al., 2010). Hence, for the cathodic electrochemical reaction we get:

$$\frac{1}{2}\mu_{O_2}^C - 2F\Phi_{el}^C = \mu_{O_{YSZ}^{2-}}^C - 2F\Phi_{io} \tag{3.21}$$

where $\mu_{O_2}^{C}$ *and* $\mu_{O_{YSZ}^{2-}}^{C}$ are the chemical potentials (the superscript C stands for the cathode) of O_2 and O_{YSZ}^{2-}, respectively, z_i is the valency (charge) of species i (z is equal to 0 for uncharged species and to -1 for electrons), F is the Faraday constant, Φ_{el}^{C} and Φ_{io} are the local equilibrium potentials of the cathode electrode and the electrolyte phase, respectively.

The equilibrium potential, E_{eq}, is defined as the equilibrium electric potential difference between two phases. Thus, rearranging Eq. (3.21), the equilibrium potential can be given by:

$$E_{eq,C} = \Phi_{el}^{C} - \Phi_{io} = \frac{1}{2F}\left(\frac{1}{2}\mu_{O_2}^{C} - \mu_{O_{YSZ}^{2-}}^{C}\right) \tag{3.22}$$

where Φ_{el}^{C} and Φ_{io} are the local equilibrium potentials of the cathode electrode and the electrolyte phase, respectively (the superscript C denotes the cathode).

Similarly, the equilibrium potentials $E_{eq,A1}$, $E_{eq,A2}$ and $E_{eq,A3}$ related to the anodic electrochemical reactions (3.18), (3.19) and (3.20), take the form:

$$E_{eq,A_1} = \Phi_{io} - \Phi_{el}^{A} = \frac{1}{2F}\left(\mu_{O_{YSZ}^{2-}}^{A} - \frac{1}{2}\mu_{O_2}^{A}\right) \tag{3.23}$$

$$E_{eq,A_2} = \Phi_{io} - \Phi_{el}^{A} = \frac{1}{2F}\left(\mu_{O_{YSZ}^{2-}}^{A} + \mu_{CO}^{A} - \mu_{CO_2}^{A}\right) \tag{3.24}$$

$$E_{eq,A_3} = \Phi_{io} - \Phi_{el}^{A} = \frac{1}{2F}\left(\mu_{O_{YSZ}^{2-}}^{A} - \mu_{BSS}^{A}\right) \tag{3.25}$$

where Φ_{el}^{A} is the local equilibrium potential of the anode electrode and μ_i^A is the chemical potential of species i (the superscript A denotes the anode). The factor of 2 in

the denominator represents the number of electrons, which are produced and/or consumed by each electrochemical reaction.

3.3.3 Thermodynamic open circuit potential

The thermodynamic open circuit potential, V_{OC}, also referred to as *Nernst potential* at open circuit conditions, is expressed by equation (3.26) in an analogous fashion to the one presented in Ho et al. (2009) ($V_{OC} = E_{eq,C} - \sum E_{eq,A_i}$):

$$V_{OC} = 3E_{eq,C} - (-E_{eq,A_1} - E_{eq,A_2} - E_{eq,A_3}) \tag{3.26}$$

where $E_{eq,C}$ and $E_{eq,Ai}$ are the equilibrium potentials related to the cathodic and the anodic electrochemical reactions, respectively, calculated by Eq. (3.22) to Eq. (3.25). The factor of 3 follows the expression in reaction (3.17) and the use of (-) in front of the anodic equilibrium potentials in Eq. (3.26) is consistent with the expressions for the equilibrium potentials given in Eq. (3.23) to Eq. (3.25).

Combining Eq. (3.26) with Eq. ((3.22) to (3.25)) and assuming that the gas phase mixture behaves ideally ($\mu_i = \mu_i^o + RT \ln P_i$) and that the chemical potentials of oxygen anions at the two sides of the electrolyte are equal (Ho et al., 2009), i.e. $\mu^A_{O^{2-}_{YSZ}} = \mu^C_{O^{2-}_{YSZ}}$,we derive the following expression for V_{OC}:

$$V_{OC} = V^o_{OC} - \frac{1}{2F}\mu^A_{BSS} + \frac{RT}{2F}\ln\left(\frac{P^A_{CO}}{P^A_{CO_2}}\right) + \frac{RT}{2F}\ln\left(\frac{\left(P^C_{O_2}\right)^{3/2}}{\left(P^A_{O_2}\right)^{1/2}}\right) \tag{3.27}$$

where V^o_{OC} is the *ideal Nernst potential* (the superscript *o* denotes standard conditions) which is expressed as:

$$V_{OC}^{o} = \frac{1}{2F}\left(\mu_{O_2}^{o} + \mu_{CO}^{o} - \mu_{CO_2}^{o}\right) \tag{3.28}$$

3.3.4 Current density distributions

The charge transfer between the electronic and ionic current, due to the electrochemical reactions at the anodic and cathodic TPBs, can be evaluated by the Butler-Volmer equation. Consequently, the current density distribution due to the electrochemical reduction of oxygen (Rxn (3.17)) taking place at the cathodic TPB is expressed as (Suzuki et al., 2008; Wang et al., 2007):

$$J^{C} = 3\times J_0^{C}\left[\exp\left(\alpha^{C}\frac{n_e F}{RT}\eta^{C}\right) - \exp\left(-(1-\alpha^{C})\frac{n_e F}{RT}\eta^{C}\right)\right] \tag{3.29}$$

where J_0^C is the exchange current density of the cathode, α^C is the cathodic charge transfer coefficient, n_e is the number of electrons transferred in the cathodic electrochemical reaction and η^C is the overpotential of the cathode.

The electrochemical reduction of O_2 (Rxn (3.17)) is considered as three different reactions and the parallel electrical circuit analogy is applied (Achenbach, 1994; Andreassi et al., 2009), hence the factor of 3 in Eq. (3.29).

The exchange current density follows an Arrhenius expression and for the cathodic electrochemical reaction (Rxn (3.17)) it is given by (Andreassi et al., 2007):

$$J_0^{C} = \gamma_C\left(\frac{P_{O_2}^{C}}{P_{ref}}\right)^{0.25}\exp\left(\frac{-E_A^{C}}{RT}\right) \tag{3.30}$$

where γ_C is the exchange current density pre-exponential factor, P_{ref} is the reference pressure in the reactor, i.e. the total gas pressure in the reactor, and E_A^C is the activation energy of the cathode.

The overpotential of the cathode is defined as (Bove and Ubertini, 2006; Tseronis et al., 2008, 2012):

$$\eta^C = V_{OC} - (\Phi_{el}^C - \Phi_{io}) \tag{3.31}$$

At the anodic TPB, three electrochemical reactions take place. Hence, the total current density distribution of the anode can be expressed using the parallel electrical circuit analogy as (Achenbach, 1994; Andreassi et al., 2009):

$$J^A = J_1^A + J_2^A + J_3^A \tag{3.32}$$

where J_1^A, J_2^A and J_3^A are the current densities due to the electrochemical reactions (3.18), (3.19) and (3.20), respectively, given by the following expression:

$$J_i^A = J_{0,i}^A \left[\exp\left(\alpha^A \frac{n_e F}{RT} \eta^A \right) - \exp\left(-(1-\alpha^A) \frac{n_e F}{RT} \eta^A \right) \right], \quad \text{i} = 1,2,3 \tag{3.33}$$

where $J_{0,i}^A$ is the exchange current density of the anode for each electrochemical reaction i, α^A is the anodic charge transfer coefficient and η^A is the overpotential of the anode.

The exchange current density of the electrochemical reaction (3.19) can be described by a similar correlation to the one used for the electrochemical H_2 oxidation and has the following form (Costamagna et al., 2004; Tseronis et al., 2008, 2012):

$$J_{0,2}^A = \gamma_{A,2} \left(\frac{P_{CO}^A}{P_{ref}} \right) \left(\frac{P_{CO_2}^A}{P_{ref}} \right) \exp\left(\frac{-E_A^A}{RT} \right) \tag{3.34}$$

where $\gamma_{A,2}$ is the exchange current density pre-exponential factor of electrochemical reaction (3.19), and E_A^A is the activation energy of the anode.

The expression for the exchange current density for reaction (3.18) follows:

$$J_{0,1}^A = \gamma_{A,1} \left(\frac{P_{O_2}^A}{P_{ref}} \right)^{-0.25} \exp\left(\frac{-E_A^A}{RT} \right) \tag{3.35}$$

The power of -0.25 in Eq. (3.35), comes from the stoichiometric analogy between electrochemical reactions (3.17) and (3.18). The exchange current density for reaction (3.20) is expressed for first time in this study and is considered to be partial pressure independent due to the unknown nature of BSS. Hence,

$$J_{0,3}^{A} = \gamma_{A,3} \exp\left(\frac{-E_A^A}{RT}\right) \tag{3.36}$$

The parameter $\gamma_{A,3}$ in Eq. (3.36) represents an "effective" exchange current density pre-exponential factor. The overpotential of the anode is defined as (Bove and Ubertini, 2006; Tseronis et al., 2008, 2012):

$$\eta^{A} = \Phi_{el}^{A} - \Phi_{io} \tag{3.37}$$

where Φ_{el}^{A} and Φ_{io} are the local electronic and ionic potentials at the anode side, respectively.

3.4 Electrochemically promoted CO oxidation

The well mixed (CSTR) reactor used in this study is of "single-pellet" type and is illustrated in Figure 3.3 along with the 3-D computational domain of the single pellet. The temperature in the reactor is considered constant. The pellet consists of a solid oxide electrolyte (YSZ) and three electrodes, the (anode) working electrode (Pt), and the (cathode) counter and reference electrodes (Au), with surface areas of 2 cm^2, 1.2 cm^2 and 0.1 cm^2, respectively. The thickness, *H*, of the electrolyte is 400 μm. The electrolyte and the electrodes are all exposed to the reacting gas mixture. The gas mixture at the inlet of the reactor consists of O_2 and CO while Helium is used as carrier gas.

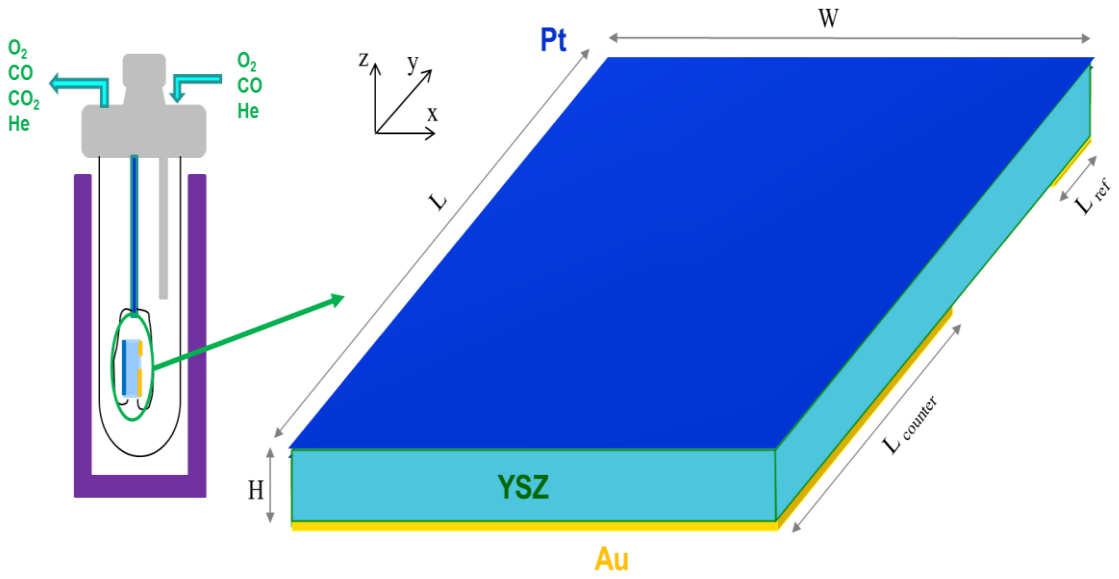


Figure 3.3: Reactor design and 3-D computational domain.

Charge transfer occurs in the single pellet due to the potential difference application between the anode and the cathode electrodes. Consequently, electrochemical reactions take place at the TPBs of the anode and the cathode, leading to a transfer of electronic to ionic current and the vice versa. BSS is formed at the TPBs of the anode and spills over the catalytic surface enhancing its activity.

This solid oxide single pellet framework can be used to simulate the mass and charge conservation as well as the chemical and electrochemical processes taking place on the catalytic surface and the triple phase boundaries respectively. COMSOL Multiphysics 3.5a (COMSOL, 2008) is a commercial computational fluid dynamics (CFD) software which is used here to construct and simulate the proposed model employing the finite element method. The model can compute species coverage on the catalytic surface, electronic and ionic potential throughout the pellet, gas mixture concentration in the reactor and CO_2 production rate.

A 2-D version of the framework is presented in the paragraphs that follow. The 2-D computational domain of the pellet depicted in Figure 3.4, represents an X-Z cross section of the 3-D pellet illustrated in Figure 3.3. The anode (Pt) and cathode (Au) electrodes are considered as boundaries on the electrolyte rather than finite regions.

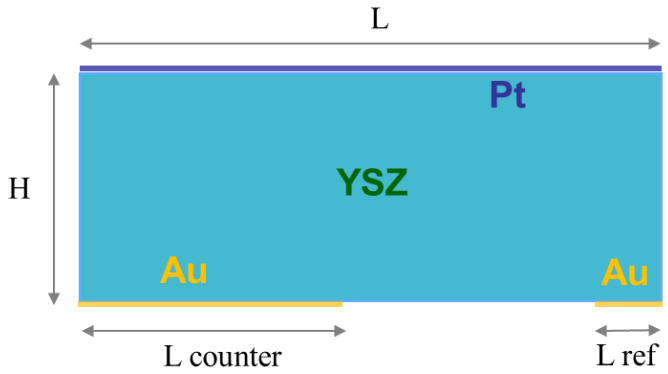


Figure 3.4: The 2-D computational domain of the single pellet.

3.4.1 Model assumptions

The main assumptions that have been made for this 2-D macroscopic model are listed below:

- The thickness of the anode and cathode electrodes is considered to be negligible since it is too small compared with the thickness of the electrolyte. Thus, the electrodes are considered as no thickness (2-D) boundaries of the electrolyte domain.
- The anodic and cathodic TPBs are represented by the perimeters of the anode and cathode electrodes, respectively.
- The gaseous species mixture around the single pellet are assumed as well-mixed. Moreover, the gas mixture is considered to behave as an ideal gas.

- The total pressure of the gas phase mixture is assumed to be constant throughout the reactor.

- The temperature throughout the reactor as well as on the catalytic surface and in the solid oxide single pellet is considered constant.

- The applied potential difference is considered to be constant throughout the pellet, assuming that electrolyte and electrodes are all good charge conductors.

- The electrochemical reactions are considered to occur only at the TPBs of the anode and the cathode.

- Only the catalytic reactions on Pt are taken into account, while the activity of Au (in the presence of Pt), can be neglected at temperatures as high as 600K.

- The model accounts only for the diffusion of BSS for the mass transport over the Pt surface, since for the other species adsorption/desorption and surface reaction are dominant. The BSS diffusion coefficient is considered to be temperature and pressure independent.

- The chemical potential of BSS at the anodic TPBs is considered to be equal to the one of the O^{2-} on the YSZ surface ($\mu_{BSS} = \mu_{O^{2-}_{YSZ}}$). The value of the chemical potential of O^{2-}, $\mu_{O^{2-}_{YSZ}}$, was derived from experimental data (no temperature dependence) and was found to be equal to -236.4 kJ/mol (Bessler et al., 2007a; Vogler et al., 2009).

3.4.2 Governing equations

3.4.2.1 Mass transport equations on the catalytic surface

On the non-porous catalytic surface, the mass conservation equation for species i, has the form (Vayenas and Pitselis, 2001):

$$\frac{\partial C_i}{\partial t} = -\nabla \cdot (-D_i \nabla C_i) + R_i \tag{3.38}$$

Eq. (3.38) can be expressed in terms of the coverage of species i, θ_i, as:

$$\frac{\partial \theta_i}{\partial t} = -\nabla \cdot (-D_i \nabla \theta_i) + R_i^{'} , \qquad i = O, CO, BSS \tag{3.39}$$

where $R_i^{'}$ is the overall reaction rate of species i (expressed in 1/s units) which is described by Eq. ((3.6) to (3.8)).

The BSS Faradaic production rate, r_{elec}, is expressed as (Foti et al., 2002):

$$r_{elec} = \frac{J_3^A}{2FN_S}\left(1-\theta_O+\theta_{CO}+\theta_{BSS}\right) \tag{3.40}$$

where J_3^A is the current density of anode due to BSS formation (Rxn (3.20)).

3.4.2.2 Charge transport equations throughout the pellet

The charge conservation in a non-porous medium *j (j=io (ionic), el (electronic))*, is described by the following equation (Cheddie and Munroe, 2006; Tseronis et al., 2012):

$$\frac{d\rho_j}{dt} = -\nabla \cdot \mathbf{J_j} + \mathrm{Q}_j \tag{3.41}$$

where ρ_j is the charge density, Q_j is the charge source term and $\mathbf{J_j}$ is the current density.

The current density can be described by Ohm's law as:

$$\mathbf{J_j} = -\sigma_j \nabla \Phi_j \tag{3.42}$$

where σ_j is the charge conductivity and Φ_j is the electrostatic potential.

Combining Eq. (3.41) with Eq. (3.42) yields:

$$\frac{d\rho_j}{dt} = -\nabla(-\sigma_j \nabla \Phi_j) + \mathrm{Q}_j \tag{3.43}$$

Electronic and ionic charge transfer occurs in the solid oxide single pellet due to polarization. The governing equations describing the charge transport at the electrolyte and the electrode domains, considering no charge source, are described as follows:

$$\frac{d\rho_{io}}{dt} = -\nabla(-\sigma_{io} \nabla \Phi_{io}) \tag{3.44}$$

$$\frac{d\rho_{el}^{A/C}}{dt} = -\nabla(-\sigma_{el}^{A/C} \nabla \Phi_{el}^{A/C}) \tag{3.45}$$

where ρ_{io} and $\rho_{el}^{A/C}$ are the ionic and electronic charge densities, σ_{io} and $\sigma_{el}^{A/C}$ are the ionic and electronic conductivities of the electrolyte and the electrodes, and, Φ_{io} and $\Phi_{el}^{A/C}$ are the ionic and electronic potentials respectively. The superscripts A/C denote the Anode/Cathode.

3.4.2.3 Gas phase modelling

As mentioned above, the gas phase mixture around the solid oxide single pellet is assumed to be well mixed and to behave as an ideal gas. The partial pressures of the gas phase species in the reactor are described by:

$$P_{CO_2} = P_{CO_2}^{out} = \frac{RT}{F_d}\hat{R}_{CO_2} \tag{3.46}$$

$$P_{CO} = P_{CO}^{out} = P_{CO}^{in} - \frac{RT}{F_d}\hat{R}_{CO} \tag{3.47}$$

$$P_{O_2} = P_{O_2}^{out} = P_{O_2}^{in} - \frac{RT}{F_d}\hat{R}_{O_2} \tag{3.48}$$

$$P_T^{in} = P_T^{out} = P_{O_2}^{out} + P_{CO}^{out} + P_{CO_2}^{out} + P_{He}^{out} \tag{3.49}$$

where P_i is the partial pressure of species i, F_d is the volumetric flowrate of the gas phase mixture in the inlet/outlet of the reactor and $\hat{R}_i$ is the overall gaseous production (i=CO_2) and consumption (i=CO, O_2) rate due to both chemical and electrochemical reactions.

Carbon dioxide is produced by the catalytic surface reactions ((3.3) and (3.4)) and by the electrochemical reaction (3.19). Carbon monoxide is produced/consumed by the catalytic reaction (3.2) and the electrochemical reaction (3.19). Oxygen is produced/consumed by the catalytic reactions (3.1) and (3.5), by the electrochemical excorporation of oxygen anions at the anodic TPBs (Rxn (3.18)) and by the electrochemical oxygen reduction (Rxn (3.17)). These phenomena are summarized in the expressions that follow:

$$\hat{R}_{CO_2} = \int_0^L \left[(r_3 + r_4)N_S + \frac{J_2^A}{2F} \right] W \mathrm{dx} \tag{3.50}$$

$$\hat{R}_{CO} = \int_0^L \left[(r_2 - r_{-2})N_S - \frac{J_2^A}{2F} \right] W \mathrm{dx} \tag{3.51}$$

$$\hat{R}_{O_2} = \int_0^L \left[(r_1 - r_{-1} - r_5) N_S + \frac{J_1^A}{4F} \right] W\mathrm{dx} - \int_0^{L_{count}} \frac{J^C}{4F} W\mathrm{dx} \tag{3.52}$$

where r_i is the rate of reaction i as in Eq. (3.10) to (3.12), N_S is the concentration of the active surface sites on the catalytic surface, W is the width for both the anode and cathode, J_i/n_eF is the Faradaic rate resulting by the electrochemical reactions i, L is the length of the anode electrode and L_{count} is the length of the counter electrode in the cathode.

3.4.3 Boundary conditions

In Figure 3.5, the boundaries (B) of the 2-D computational domain are shown. There are in total 8 boundaries and 8 points, 4 of which, P1, P3, P6 and P8, are considered as TPBs. It should be noted that the anode (working) electrode (Pt) is represented by boundaries B1 and B2 in this computational domain while the cathode (counter) electrode (Au) is represented by boundaries B6 and B7.

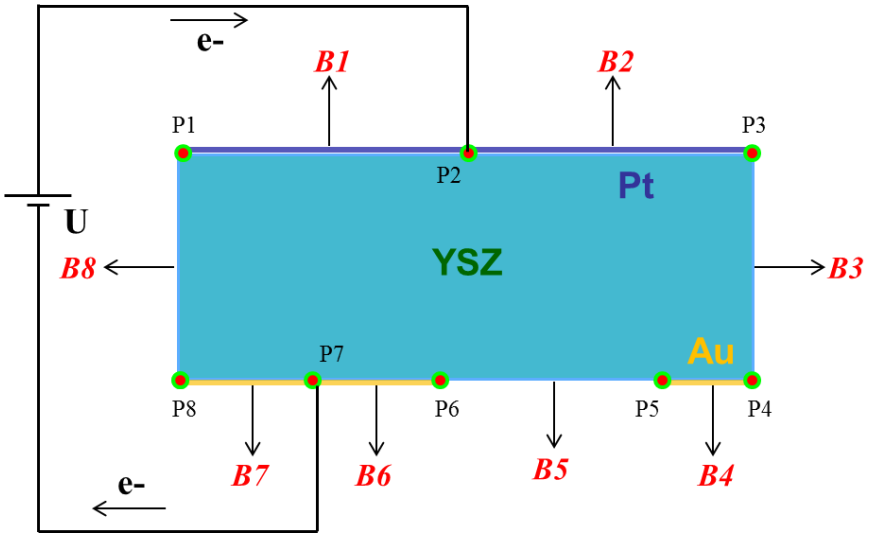


Figure 3.5: 2-D computational domain numbering of boundaries and points.

Mass Transport

At the anodic electrode (boundaries B1 & B2), the CO oxidation electro-catalytic reactions take place. The mass conservation on the catalytic surface is described by partial differential equations (PDEs) of the form of Eq. (3.39). For the solution of the mass conservation PDEs, boundary conditions need to be imposed. Thus, the no flux condition is imposed at the edges of the anode electrode (points P1 and P3) for all species:

P1 & P3: $$\mathbf{n}\cdot\left(-D_i\nabla\theta_i\right)=0\ ,\qquad i=O,CO,BSS \tag{3.53}$$

where **n** is the unit normal vector.

Charge Transport

The circuit is closed at points P7 and P2 (see Figure 3.5). The electronic potential is fixed to the value of the operating potential Φ_{pellet} at P7 and is equal to 0 at P2. At the points P6 and P8, the cathodic reduction of O_2 takes place and leads to a transfer of electronic to ionic current. Thus,

P7: $$\Phi_{el}^{C}=\Phi_{pellet} \tag{3.54}$$

P2: $$\Phi_{el}^{A}=0 \tag{3.55}$$

P6 & P8: $$\begin{aligned}-\mathbf{n}\cdot\left(-\sigma_{el}^{C}\nabla\Phi_{el}^{C}\right)&=J^{C}\\ -\mathbf{n}\cdot\left(-\sigma_{io}\nabla\Phi_{io}\right)&=-J^{C}\end{aligned} \tag{3.56}$$

where J^C is computed from the modified form of the Butler-Volmer (Eq. (3.29)).

At the points P1 and P3, transfer of ionic to electronic current occurs, due to the electrochemical reactions taking place at the anodic TPBs. Hence,

P1 & P3:
$$\begin{aligned} -\mathbf{n}\cdot\left(-\sigma_{io}\nabla\Phi_{io}\right) &= J^{A} \\ -\mathbf{n}\cdot\left(-\sigma_{el}^{A}\nabla\Phi_{el}^{A}\right) &= -J^{A} \end{aligned} \tag{3.57}$$

Here J^A is computed by equations (3.32) and (3.33).

At all the remaining boundaries and points, no flux conditions are imposed.

B3, B5 & B8:
$$\mathbf{n}\cdot\left(-\sigma_{io}\nabla\Phi_{io}\right)=0 \tag{3.58}$$

P4 & P5:
$$\mathbf{n}\cdot\left(-\sigma_{el}^{C}\nabla\Phi_{el}^{C}\right)=0 \tag{3.59}$$

3.5 3-D macroscopic model

The 3-D computational domain of the solid oxide single pellet is presented in Figure 3.3. Similarly to the 2-D framework, the anode and cathode electrodes are considered as boundaries of the electrolyte rather than finite regions. The mass and charge conservation is, here too, described by Eq. (3.39) and (3.44-3.45). The gas phase species production/consumption overall rates are given by the surface integrals:

$$\hat{R}_{CO_2}=\int_0^W\int_0^L\left[(r_3+r_4)N_S+\frac{J_2^A}{2F}\right]\mathrm{dxdz} \tag{3.60}$$

$$\hat{R}_{CO}=\int_0^W\int_0^L\left[(r_2-r_{-2})N_S-\frac{J_2^A}{2F}\right]\mathrm{dxdz} \tag{3.61}$$

$$\hat{R}_{O_2}=\int_0^W\int_0^L\left[(r_1-r_{-1}-r_5)N_S+\frac{J_1^A}{4F}\right]\mathrm{dxdz}-\int_0^W\int_0^{L_{count}}\frac{J^C}{4F}\mathrm{dxdz} \tag{3.62}$$

3.5.1 Boundary conditions

The 3D computational domain boundaries (B), edges (E) and points (P) are illustrated in Figure 3.6. There are in total 8 boundaries, 18 edges (8 of which, E1-4, E6-7 and E13-14 are considered as TPBs) and 14 points. The working electrode (Pt) at the anode is represented by boundary B1 while the counter electrode (Au) at the cathode is represented by boundary B4.

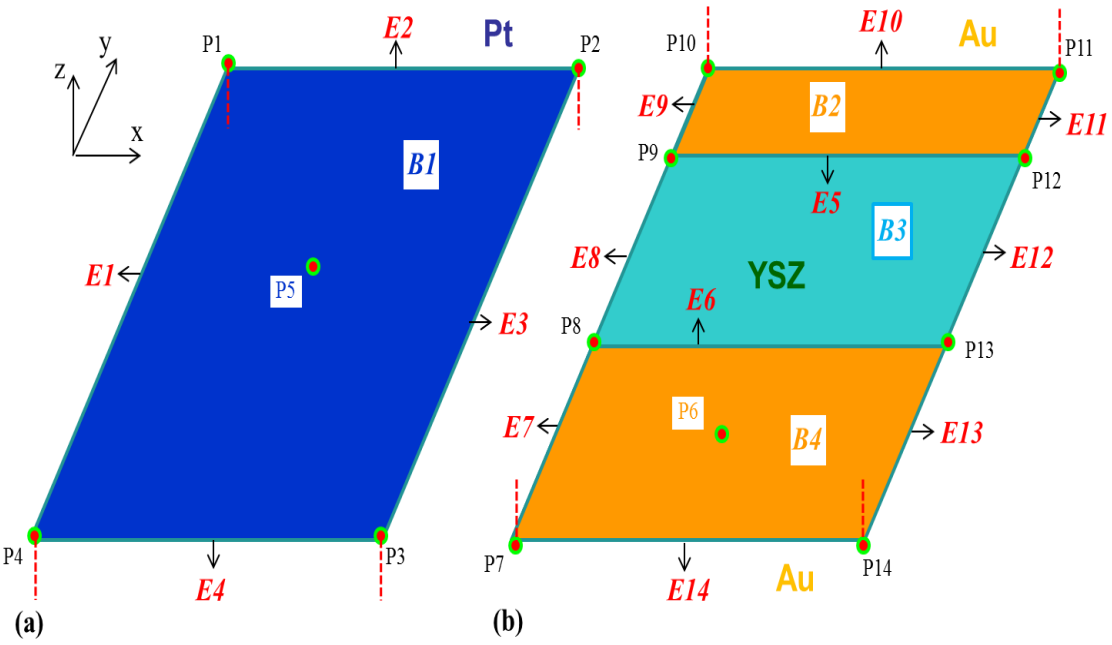


Figure 3.6: 3-D computational domain numbering of boundaries, edges and points, (a) single pellet top view, (b) single pellet bottom view.

Mass transport

No flux is imposed for all species at the edges E1-4:

E1-4: $$\mathbf{n}\cdot\left(-D_i\nabla\theta_i\right)=0\,,\qquad i=O,CO,BSS \tag{3.63}$$

Charge transport

The electronic potential is fixed to the value of the operating potential Φ_{pellet} at point P6 and to 0 at point P5:

P6: $$\Phi_{el}^{C} = \Phi_{pellet} \tag{3.64}$$

P5: $$\Phi_{el}^{A} = 0 \tag{3.65}$$

Transfer of electronic to ionic charge occurs at edges E6-7 and E13-14. Hence:

E6-7 & E13-14: $$\begin{aligned} -\mathbf{n}\cdot\left(-\sigma_{el}^{C}\nabla\Phi_{el}^{C}\right) &= J^{C} \\ -\mathbf{n}\cdot\left(-\sigma_{io}\nabla\Phi_{io}\right) &= -J^{C} \end{aligned} \tag{3.66}$$

Ionic current is transferred to electronic at edges E1-4:

E1-4: $$\begin{aligned} -\mathbf{n}\cdot\left(-\sigma_{io}\nabla\Phi_{io}\right) &= J^{A} \\ -\mathbf{n}\cdot\left(-\sigma_{el}^{A}\nabla\Phi_{el}^{A}\right) &= -J^{A} \end{aligned} \tag{3.67}$$

At all the remaining boundaries and edges the no flux condition is imposed for both ionic and electronic charge transfer.

3.6 Numerical solution approach

The finite element method is used for the simultaneous solution of the set of PDEs ((3.39),(3.44)-(3.45) and (3.53)-(3.59)) implemented through COMSOL Multiphysics 3.5a. The 2- and 3-D computational domains were discretized in 6,568 triangular and 29,438 tetrahedral triangular elements, respectively which were deemed sufficient to produce mesh-independent solutions. The resulting sets of nonlinear partial differential equations were solved using the Direct (UMFPACK) solver in COMSOL.

The mass conservation on the catalytic surface as well as the charge balances in both the cathode and anode electrodes were implemented by utilising the weak form boundary PDEs. Moreover, the COMSOL facility of "integrated coupling variables" was used to implement the expressions for each species partial pressure in the reactor (Eq. (3.46) to (3.48)). The selected model parameters are tabulated in Table 3.1. The electronic charge conductivities are considered to be dependent on the system temperature and are given by (COMSOL, 2011):

$$\sigma_{el}^{Pt} = \frac{1}{2.814022\cdot10^{-17}\mathrm{T}^3 - 1.600249\cdot10^{-13}\mathrm{T}^2 + 5.552497\cdot10^{-10}\mathrm{T} - 4.843579\cdot10^{-8}} \quad (3.68)$$

$$\sigma_{el}^{Au} = \frac{1}{1.275961\cdot10^{-17}\mathrm{T}^3 - 4.720065\cdot10^{-16}\mathrm{T}^2 + 7.877041\cdot10^{-11}\mathrm{T} - 1.145028\cdot10^{-9}} \quad (3.69)$$

The ionic charge conductivity is given by (Ferguson et al., 1996):

$$\sigma_{io}^{YSZ} = 33.4\cdot10^{3}\exp\left(\frac{-10300}{\mathrm{T}}\right) \quad (3.70)$$

Table 3.1: Solid oxide single pellet model parameters.

Parameter	***Symbol***	***Value***
Pt specific site density, mol m^{-2}	N_S	$2.5407x10^{-5}$ (Yentekakis et al., 1994)
Sticking coefficient of O_2 on Pt	S_{O_2}	Estimated in the present work
Sticking coefficient of CO on Pt	S_{CO}	Estimated in the present work
O_2 desorption pre-exponential factor, s^{-1}	$k_{o,-1}$	$2.4x10^{13}$ (Kaul et al., 1987)
O_2 desorption activation energy, J mol^{-1}	$E_{A,-1}$	Estimated in the present work
CO desorption pre- exponential factor , s^{-1}	$k_{o,-2}$	$6.5x10^{13}$ (Kaul et al., 1987)
CO desorption activation energy, J mol^{-1}	$E_{A,-2}$	Estimated in the present work
CO and O_2 surf. reaction's pre-exp factor, s^{-1}	$k_{o,3}$	$2.7x10^{6}$ (Kaul et al., 1987)
CO oxidation activation energy, J mol^{-1}	$E_{A,3}$	Estimated in the present work
BSS and CO surface reaction rate constant, s^{-1}	k_4	Estimated in the present work
BSS desorption rate constant, s^{-1}	k_5	Estimated in the present work
BSS diffusion coefficient, m^2 s^{-1}	D_{BSS}	$4x10^{-15}$ (Vayenas and Pitselis, 2001)
Cathodic charge transfer coefficient	α^C	0.5

Anodic charge transfer coefficient	α^A	0.5
Anode activation energy, J mol^{-1}	$E_A{}^A$	120000 (Chaisantikulwat et al., 2008)
Cathode activation energy, J mol^{-1}	$E_A{}^C$	110000 (Chaisantikulwat et al., 2008)
Cathode pre-exponential coefficient, A m^{-2}	γ_C	6.91×10^8 (Chaisantikulwat et al., 2008)
Anode pre-exponential coefficient, A m^{-2}	$\gamma_{A,1}$	Estimated in the present work
Anode pre-exponential coefficient, A m^{-2}	$\gamma_{A,2}$	Estimated in the present work
Anode pre-exponential coefficient, A m^{-2}	$\gamma_{A,3}$	Estimated in the present work
Anode electronic charge conductivity, Ω^{-1} m^{-1}	$\sigma_{el}{}^A$	as in Eq. (3.68)
Cathode electronic charge conductivity, Ω^{-1} m^{-1}	$\sigma_{el}{}^C$	as in Eq. (3.69)
Electrolyte ionic charge conductivity, Ω^{-1} m^{-1}	σ_{io}	as in Eq. (3.70)
BSS chemical potential, J mol^{-1}	μ_{BSS}	-236.4×10^3 (assumed in this work)
CO standard chemical potential, J mol^{-1}	$\mu_{CO}{}^O$	-137.3×10^3 (Perry et al., 1963)
CO_2 standard chemical potential, J mol^{-1}	$\mu_{CO_2}{}^O$	-394.4×10^3 (Perry et al., 1963)

3.7 Results and discussion

The system operating conditions used in the framework are presented in Table 3.2. The operating potential, Φ_{pellet}, is considered to be constant throughout the pellet. The solid oxide single pellet is assumed to be preheated at the operating temperature value. Extensive parameter estimation studies were performed, using the 2-D model, for parameters, with unknown values, which are of crucial importance for modelling EPOC systems. Moreover, both the 3- and the 2-D model were validated against experimental data from the literature (Yentekakis et al., 1994). The 2-D model was further exploited for sensitivity analysis as well as parametric investigation studies.

Table 3.2: System operating conditions.

Parameter	***Symbol***	***Value***
Operating Pressure, atm	P_{ref}	1
Temperature in the reactor, K	T	645.15
Ideal gas constant, J mol^{-1} K^{-1}	R	8.3145
Faraday constant, A s mol^{-1}	F	96485
Operating Potential, V	Φ_{pellet}	0.7
Electrolyte/Anode length, m	L	10^{-2}
Electrolyte/Anode/Cathode width, m	W	$2x10^{-2}$
Electrolyte height, m	H	$4x10^{-4}$
Cathode counter electrode length, m	L_{count}	$6x10^{-3}$

3.7.1 Parameter estimation

The chemical and electrochemical reaction kinetic constants for such an Electrochemically Promoted system are not available in the literature. Albeit the catalytic system for the oxidation of CO on Pt has been extensively studied (Campbell et al., 1980, 1981a, 1981b), a direct extension of open circuit catalytic kinetic data to the closed-circuit system is not plausible: polarization application leads to alterations in both the steady state and the dynamic system behaviour. This observation has been the backbone of all EPOC studies to date. The principal cause for this alteration is the polarization induced change of all reaction and transport phenomena taking place in the system stemming from (i) the addition of reactions (3.4) and (3.5) to the Langmuir-Hinshelwood mechanism for the CO oxidation (Rxns (3.1)-(3.3)) and (ii) the rate modification for reactions (3.1) to (3.3). In fact, the first of the reasons delineated is less significant, as for this system it has been reported that the BSS is mainly acting as a promoter rather than a reactant and subsequently the rates of reactions ((3.4)-(3.5)) are relatively low and constrained by Eq. (3.16). Consequently, the kinetic parameters of the closed-circuit system are expected to differ significantly from the ones of the open-circuit system. This is especially so since the reported bounds for the open-circuit kinetic values at hand are quite broad (cf. Kaul et al., 1987).

For this reason, a parameter estimation study was undertaken, using our 2-D model in conjunction with reported closed-circuit experimental data of overall CO_2 production rates for a range of CO partial pressures available in literature (Yentekakis et al., 1994). The operating potential was 700mV, the partial pressure of the oxygen at the reactor inlet was 5.8 kPa, and the temperature of the gas mixture in the reactor was 372 ^{o}C.

The geometrical aspects (as in Table 3.3) of our system used for the parameter estimation studies are considered to be *typical* for this kind of experiments (Yentekakis and Vayenas, 1988; Yentekakis et al., 1988; Vayenas et al., 1989, 1990b; Yentekakis and Bebelis, 1992; Yentekakis et al., 1994). The system electrodes were reported to have thickness of 5 μm, which we considered to be negligible compared with the 0.4 mm thickness of the support, hence they were considered as 2-D boundaries of the electrolyte rather than finite regions (see Figures 3.3 and 3.4).

Table 3.3: Geometrical aspects of the system.

Domain	***Length (cm)***	***Width (cm)***	***Other characteristics***
Anode	1	2	Area: 2 cm^2
Electrolyte	1	2	Volume: 0.0008 cm^3
Cathode (counter)	0.6	2	Area: 1.2 cm^2

Our framework was employed for the estimation of kinetic parameters such as: the sticking coefficient of O_2 (S_{O_2}) and CO (S_{CO}) on Pt (Rxns (3.1) and (3.2), Eq. (3.14)), O_2 and CO desorption activation energies ($E_{A,-1}$ and $E_{A,-2}$ respectively, Rxns (3.1) and (3.2), Eq. (3.15)), the CO oxidation activation energy ($E_{A,3}$, Rxn (3.3), Eq. (3.15)), the BSS surface reaction rate constants (k_4 or k_5, Rxns (3.4) and (3.5), Eq. (3.16)) as well as the pre-exponential coefficients for the three electrochemical reactions of the anode ($\gamma_{A,1}$, $\gamma_{A,2}$ and $\gamma_{A,3}$, Rxns (3.18) to (3.20), Eq. (3.34-3.36)) utilized in Butler-Volmer expressions (see Table 3.1). The parameter estimation problem is set up as a nonlinear least squares problem:

$$z = \arg\min \sum_{i=1}^{n_{wxp}} \left(\hat{R}_{CO_2}(P_{CO}^{in}(i)) - R_{CO_2}(P_{CO}^{in}(i)) \right)^2 \quad (3.71)$$

where n_{wxp} is the number of experimental points, $\hat{R}_{CO_2}$ the calculated mean overall rate for CO_2 production, i.e. the average rate due to chemical and electrochemical processes, which is a function of the inlet pressure of CO, $\hat{R}_{CO_2}(P_{CO}^{in})$, as defined in Eq. (3.50), and R_{CO_2} the experimental CO_2 production rate for the same P_{CO}^{in}. The equality constraints for this optimization problem are given by the system Eq. ((3.16), (3.39) and (3.44)-(3.59)).

We assume that the NEMCA effect alters only the activation energies of the catalytic reaction rates (which is reasonable since closing the circuit will in general increase the catalytic activity by affecting the activation barriers), while the values of the reaction rate constants are taken from equivalent open-circuit literature data (Kaul et al., 1987). In contrast, the activation energies for the anodic electrochemical reactions are considered to be equal to a value taken from literature (see Table 3.1) and here we have selected to estimate the anode exchange current density pre-exponential coefficients due to the unknown nature of BSS and the assumption made for the exchange current density of reaction (3.20) (see Eq. (3.36)). Finally, the activation energy of the electrochemical reaction at the cathode is taken from literature (Chaisantikulwat et al., 2008) (see Table 3.1) as the oxygen reduction occuring at the TPBs of the cathode is *typical* for such systems. The parameter estimation task was carried out exploiting the 2-D model (whose computational domain is depicted in Figure 3.4) for computational

efficiency. The estimated parameters were consequently introduced in the 3-D framework for validation purposes.

It should be noted here that the limited number of experimental data to only one polarization potential, do not allow to investigate the parametric effect of voltage and also result in identifiability issues, noisy objective functions and multiple extrema. Parameter estimation studies for electrochemical systems in the context of SOFCs have been undertaken with the use of various optimisation methodologies, i.e. gPROMS (Leah et al., 2005), SQP (Sahibzada et al., 2000), Nelder-Mead method (Gogoi and Das, 2013).

In this work to overcome the problem of convergence to local minima, we have solved the estimation problem using Simulated Annealing (SA) (Kirkpatrick et al., 1983; Otten and van Ginneken, 1989). SA is a stochastic optimization method which can help to avoid convergence to the nearest local optimum point with the use of appropriate restart techniques. On the other hand, this optimization method is expected to probabilistically converge to the neighborhood of the global optimum. To achieve global optimum convergence, a refining step using a gradient-based, deterministic strategy, Sequential Quadratic Programming (SQP) (Gill et al., 1981; Boggs and Tolle, 1995) implemented through the *fmincon* function in MATLAB, is subsequently performed using as initial guess the result from SA. In all cases fmincon was found to converge monotonically within 7-9 iterations. It is possible, although improbable, that the attractor of the global minimum is weak and SQP will converge to a local minimum of higher objective function value. Here the best of the SA/SQP solutions is taken as the result of our estimation problem.

The values of the parameters estimated using the optimization procedure delineated above are tabulated in Table 3.4. As we can seen all estimated sticking coefficients and activation energies are within the lower and upper bounds reported in Kaul et al. (1987). Furthermore, the obtained CO desorption activation energy and the O_2 sticking coefficient are lower than the ones provided for the open-circuit system (Kaul et al., 1987). This finding follows the NEMCA theory suggesting weaker bonds of the adsorbed species due to the polarization effect (Vayenas et al., 2001a).

It is worthwhile to note here that the kinetic parameters computed though the fitting procedure, although they do illustrate the BSS effect, since they deviate from the reported open circuit values (Kaul et. al, 1987), they are by design constant hence independent of the BSS coverage variation. However, it has been reported in literature (Vayenas et al., 2001b; Brosda and Vayenas, 2002) that the kinetic constants are affected by the coverage variations of the BSS species which consequently alter the catalyst work function. Nevertheless, the available experimental data, are limited to one temperature and one operating potential, hence this effect is not currently included in our model.

In Figure 3.7, the good agreement between model predictions using the estimated parameters and experimental data is illustrated. Furthermore, the predictions of the 3-D model, using the obtained parameters (as in Table 3.4) compared with the experimental data are depicted in Figure 3.8, also showing an excellent agreement. This signifies that the current system can be sufficiently (and efficiently) represented by the 2-D model.

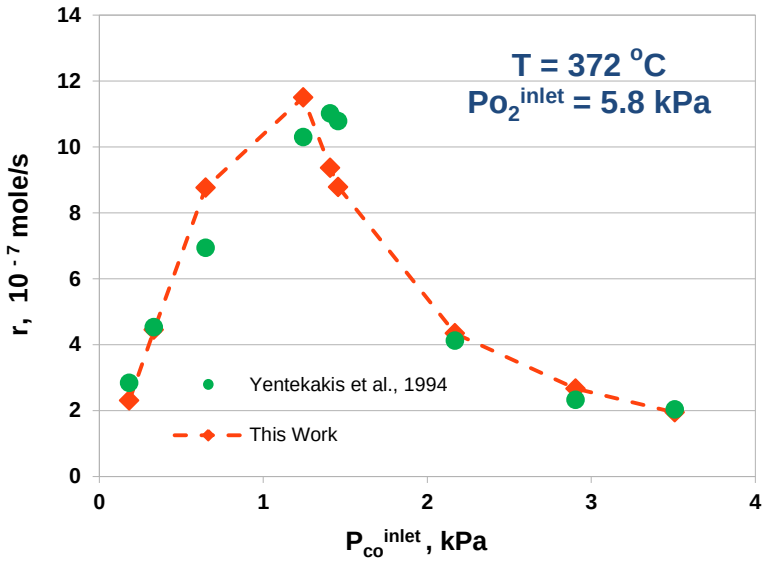


Figure 3.7: Effect of inlet partial pressure of CO on CO_2 production rate. Yentekakis et al., 1994 refers to experimental data provided in Fig. 5a of that study. This work refers to the 2-D macroscopic model developed in the present study.

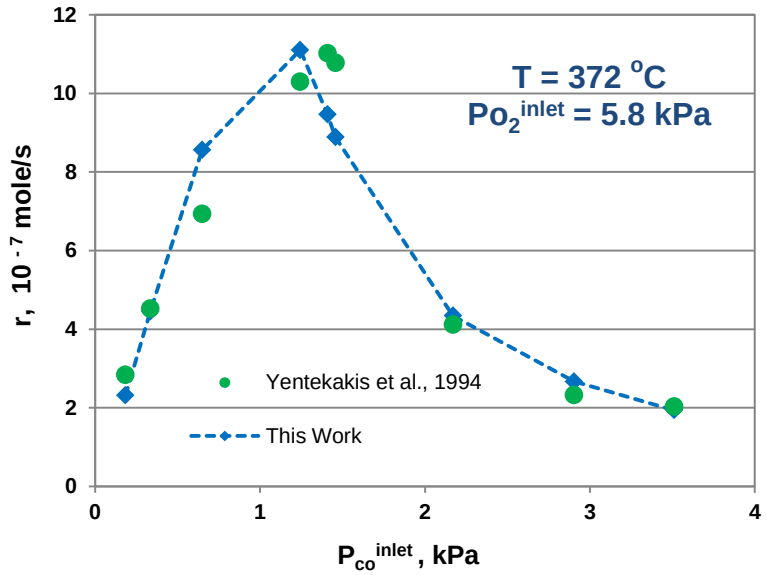


Figure 3.8: Effect of inlet partial pressure of CO on CO_2 production rate. Yentekakis et al., 1994 refers to experimental data provided in Fig. 5a of that study. This work refers to the 3-D macroscopic model formulated in the present study.

Table 3.4: Values of estimated parameters.

Parameter	***Symbol***	***Value***
Sticking coefficient of O_2 on Pt	S_{O_2}	7.69×10^{-5}
Sticking coefficient of CO on Pt	S_{CO}	5.38×10^{-1}
O_2 desorption activation energy (J mol^{-1})	$E_{A,-1}$	243139
CO desorption activation energy (J mol^{-1})	$E_{A,-2}$	99618
CO oxidation activation energy (J mol^{-1})	$E_{A,3}$	35186
BSS surface reaction rate constant (s^{-1})	k_4	5.73×10^{-3}
Anode pre-exponential coefficient (A m^{-2})	$\gamma_{A,1}$	5.01×10^{8}
Anode pre-exponential coefficient (A m^{-2})	$\gamma_{A,2}$	2.92×10^{11}
Anode pre-exponential coefficient (A m^{-2})	$\gamma_{A,3}$	3.42×10^{4}

Uncertainty in the parameter estimation is investigated through sensitivity analysis.

3.7.2 Sensitivity analysis

To quantify the effect of variations in the computed chemical and electrochemical kinetic parameters, a sensitivity analysis was carried out, with the use of the 2-D model, by introducing a percentage change (-50 to +100%) in the values of these parameters.

Table 3.5: Sensitivity analysis of estimated parameters (P_{CO}^{in} =2 kPa).

Parameter (Units)	*Estimated Value*	*% Change in parameter*	*% Resultant change in r_{CO_2}*
S_{O_2} (-)	7.69×10^{-5}	-10	-10.96
		+10	11.21
S_{CO} (-)	5.38×10^{-1}	-10	16.44
		+10	-13.38
$E_{A,-1}$ (J mol^{-1})	243139	-50	-5.68
		+100	0.00
$E_{A,-2}$ (J mol^{-1})	99618	-1	30.03
		+1	-24.64
$E_{A,3}$ (J mol^{-1})	35186	-10	4.76
		+10	-8.02
k_4 (s^{-1})	5.73×10^{-3}	-50	-0.19
		+50	0.07
$\gamma_{A,1}$ (A m^{-2})	5.01×10^{8}	-50	0.00
		+100	0.00
$\gamma_{A,2}$ (A m^{-2})	2.92×10^{11}	-50	0.00
		+100	0.00
$\gamma_{A,3}$ (A m^{-2})	3.42×10^{4}	-50	0.13
		+100	-0.20

As it is shown in Table 3.5, changes in the sticking coefficients and activation energies lead to a strong effect on the CO_2 production rate, while variations in the rate constants of the reactions involving BSS as well as the exchange current density pre-exponential coefficients have almost no effect on the CO_2 production rate. This was expected, since BSS, that is considered as a promoter rather than a reactant (the reaction rate constants of the reactions involving BBS are constrained), forms the effective double layer which subsequently affects the binding strength of the other adsorbates, while the exchange current density is related to the Faradaic contribution (as opposed to NEMCA). This observation is in great agreement with the NEMCA theory suggesting that the enhancement of CO_2 production rate comes mainly from the modification of the catalytic activity leading to an increase in the surface reaction rate. That increase in the surface reaction rate is corroborated in Figure 3.9 which illustrates the effect of P_{CO}^{in} on the enhancement factor Λ. Values of Λ of the order of 10^3 (for P_{CO}^{in}>1) are in good agreement with the ones found in literature (Vayenas and Bebelis, 1999) for this system and demonstrate how strong the Non-Faradaic effect is compared with the Faradaic one.

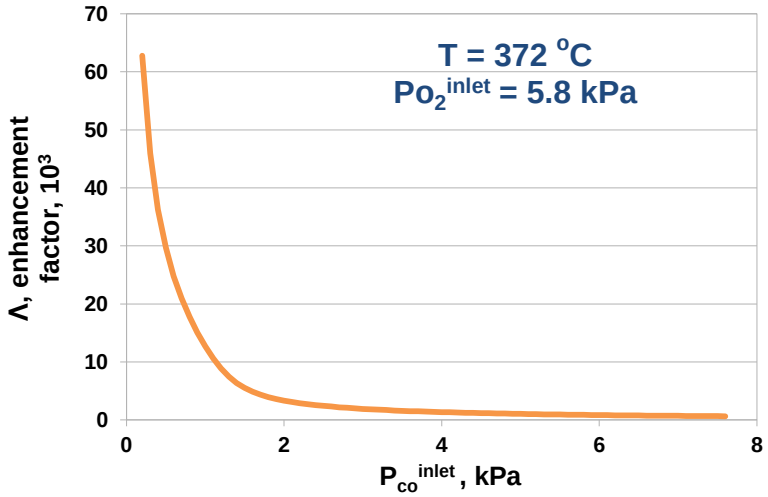


Figure 3.9: Effect of inlet partial pressure of CO on enhancement factor Λ.

It should be noted that the increase of O_2 desorption activation energy has no effect on the CO_2 production rate, while the effect due to its decrease is strong for a decrease higher than 50% (Figure 3.10). This is in good agreement with literature results according to which this parameter is not important for temperatures as low as 300 ºC (Kaul et al., 1987). On the contrary, small variations in the CO desorption activation energy lead to significant alterations in the observed CO_2 production rate (Figure 3.11). Figure 3.12 shows that the CO_2 production rate raises (for P_{CO}^{in}>1) when the sticking coefficient of oxygen is increased. The opposite effect is illustrated in Figure 3.13 where the CO_2 production rate decreases when the sticking coefficient of CO is increased. Furthermore, the activation energy of the surface reaction between adsorbed O and CO affects the CO_2 production rate as expected, i.e. raising its value leads to decrease in the CO_2 production rate (Figure 3.14).

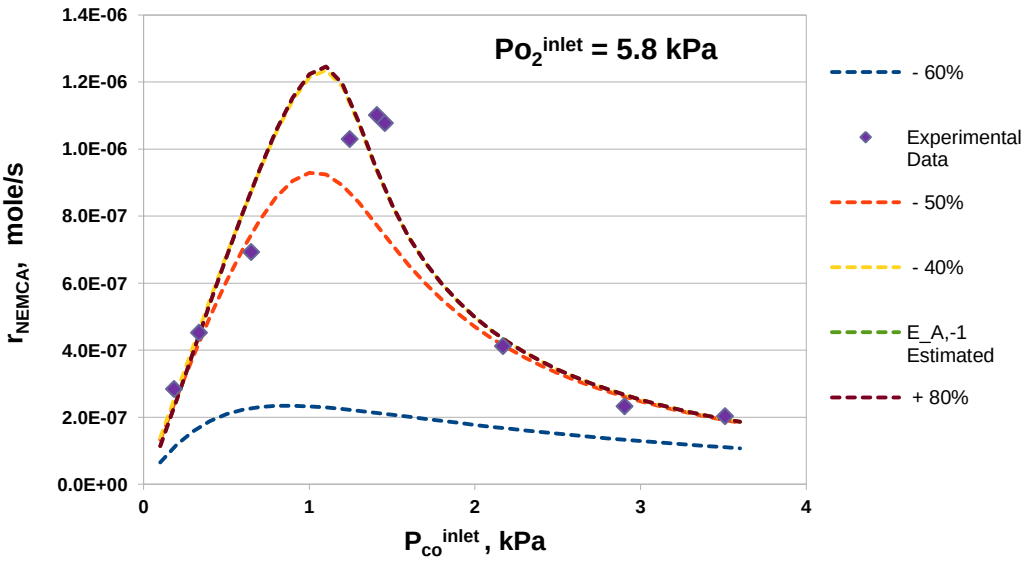


Figure 3.10: Effect of inlet partial pressure of CO on CO_2 production rate for various values of O_2 desorption activation energy, $E_{A,-1}$.

Finally, variations in the rate constant of the surface reaction between BSS and CO (which also affect the BSS desorption rate constant according to Eq. (3.16)) have almost no effect on the CO_2 production rate (see Table 3.5).

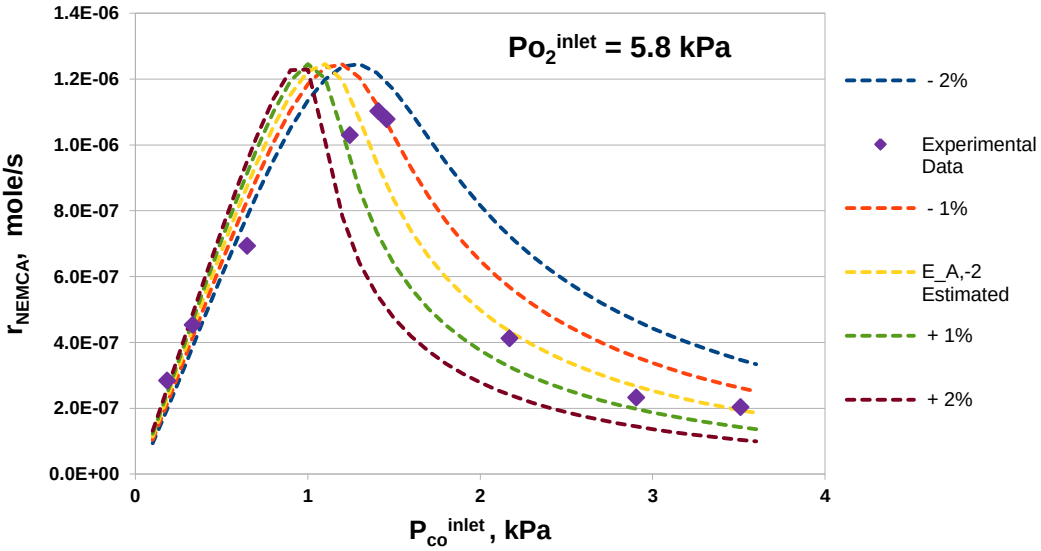


Figure 3.11: Effect of inlet partial pressure of CO on CO_2 production rate for various values of CO desorption activation energy, $E_{A,-2}$.

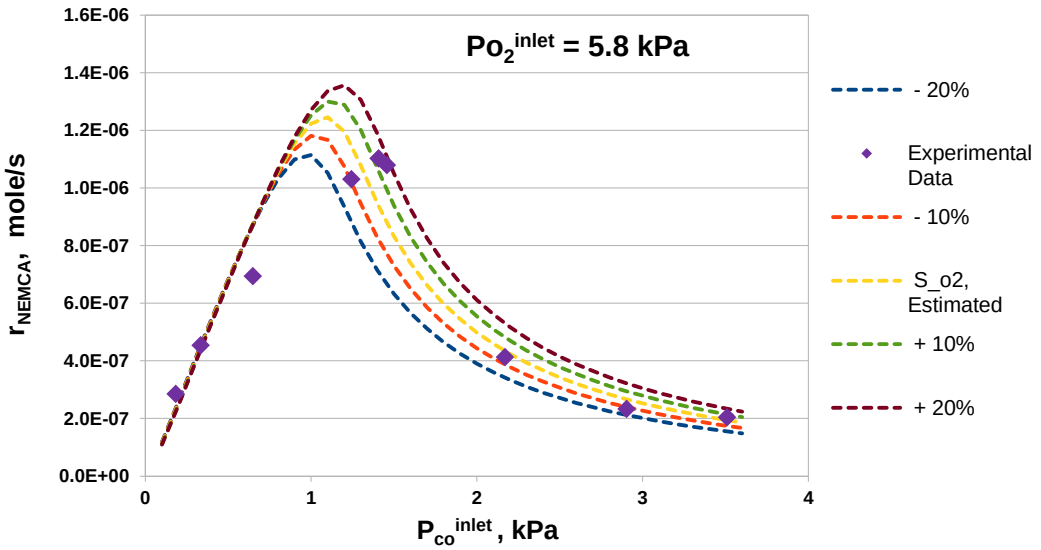


Figure 3.12: Effect of inlet partial pressure of CO on CO_2 production rate for various values of O_2 sticking coefficient, So_2.

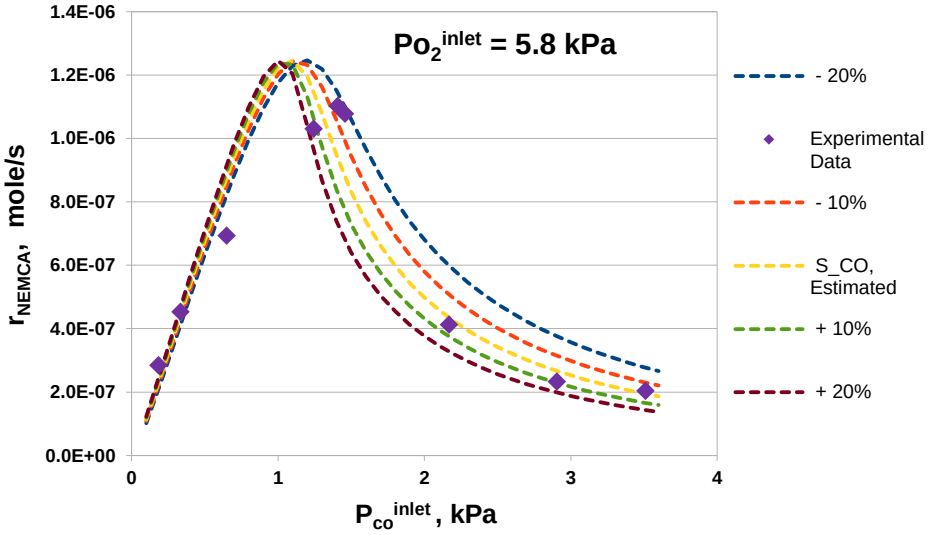


Figure 3.13: Effect of inlet partial pressure of CO on CO_2 production rate for various values of CO sticking coefficient, S_{CO}.

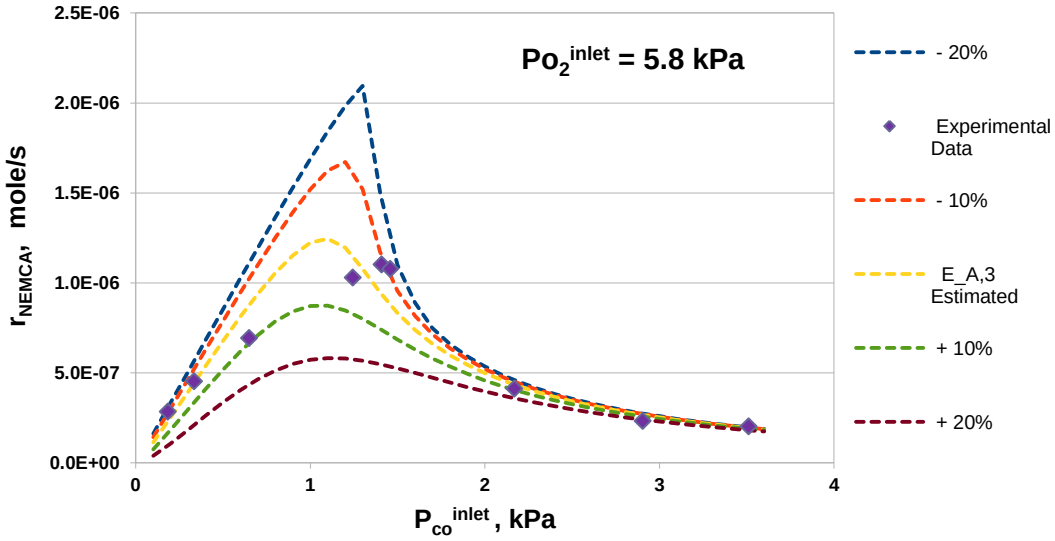


Figure 3.14: Effect of inlet partial pressure of CO on CO_2 production rate for various values of surface reaction between O_2 and CO activation energy, $E_{A,3}$.

3.7.3 Parametric investigation

We have also performed parametric studies, using our 2-D model, in order to investigate the effect of operating conditions on CO_2 production rate.

Effect of P_{CO}^{in}. Figures 3.15 and 3.16 illustrate the effect of P_{CO}^{in} on the CO_2 production rate due to Non-Faradaic and Faradaic effects respectively, at constant values of Φ_{pellet} = 700 mV and $P_{O_2}^{in} = 5.8\,\text{kPa}$. The rate profile depicted in Figure 3.15 exhibits a behaviour characteristic for Langmuir-Hishelwood kinetics. A "volcano-type" behaviour is favoured in the CO_2 production rate at relatively high $P_{O_2}^{in}$ values due to the NEMCA effect which is in good agreement with literature (Yentekakis et al., 1994), while an "S-type" behaviour is observed in the CO_2 production rate due to the Faradaic effect (Figure 3.16). Moreover, we can observe a sharp increase for both Faradaic and Non-Faradaic rates at low P_{CO}^{in}, i.e. values between 1 and 2 kPa.

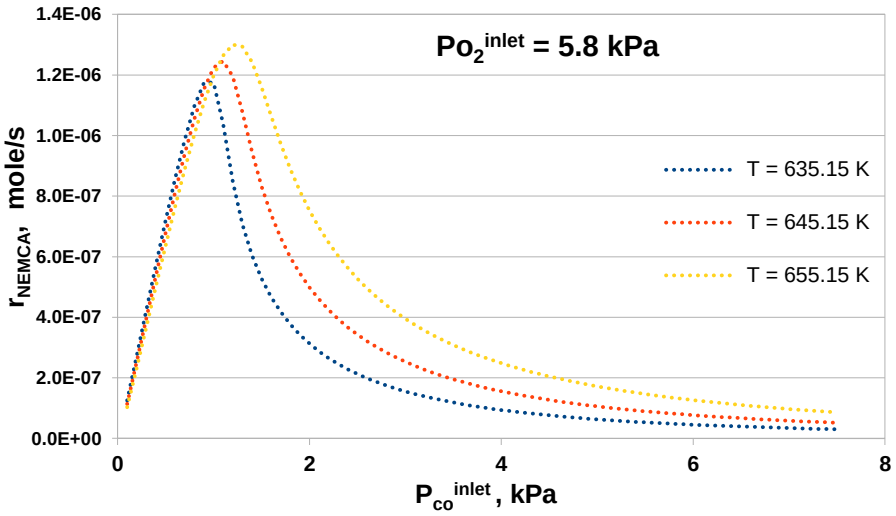


Figure 3.15: Effect of inlet partial pressure of CO on CO_2 production rate under NEMCA effect for various temperature values.

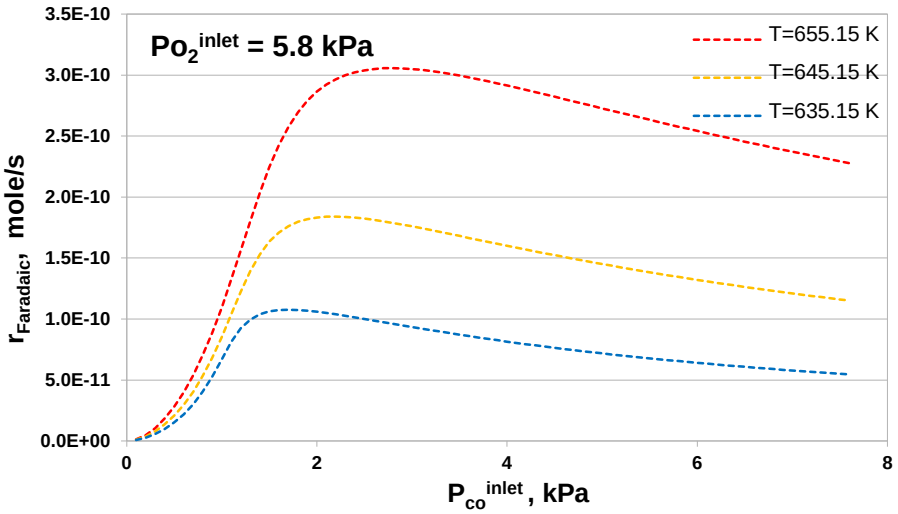


Figure 3.16: Effect of inlet partial pressure of CO on CO_2 Faradaic rate for various temperature values.

Effect of $P_{O_2}^{in}$. The effect of $P_{O_2}^{in}$ on Non-Faradaic and Faradaic reaction rates at constant values of P_{CO}^{in} = 1.5 kPa and Φ_{pellet} = 700 mV is presented in Figures 3.17 and 3.18 respectively. In this case, "S-type" behaviour is observed in the CO_2 production rate due to both NEMCA and Faradaic effects. In Figure 3.17, a CO_2 non-Faradaic rate raise is found for values of $P_{O_2}^{in}$ in the range of 5 to 10 kPa, while the rate seems to be reaching a plateau for relatively high $P_{O_2}^{in}$. In contrast, a relatively small rate rise due to the Faradaic effect is found even for higher values of $P_{O_2}^{in}$ (Figure 3.18).

Effect of Temperature. In the case of constant $P_{O_2}^{in}$ = 5.8 kPa and varying P_{CO}^{in}, raising the temperature leads to a minor increase of the electrochemically promoted rate for values of P_{CO}^{in} lower than 1.5 kPa, while the opposite effect is favoured for higher P_{CO}^{in}. Likewise, in the case of constant P_{CO}^{in} = 1.5 kPa and variable $P_{O_2}^{in}$, an increase in temperature leads to a raise of the CO_2 non-Faradaic production rate for $P_{O_2}^{in} / P_{CO}^{in} < 6$

(excess of CO), while a plateau is reached for $P_{O_2}^{in} / P_{CO}^{in} > 6$. Raising the temperature in all cases leads to an increase of the closed-circuit CO_2 production rate due to the Faradaic effect.

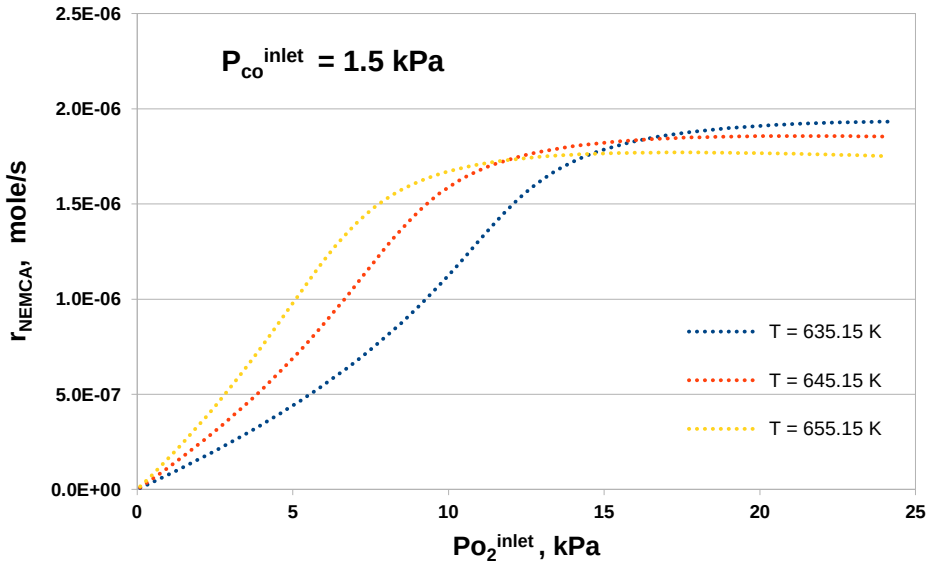


Figure 3.17: Effect of inlet partial pressure of O_2 on CO_2 production rate under NEMCA effect for various temperature values.

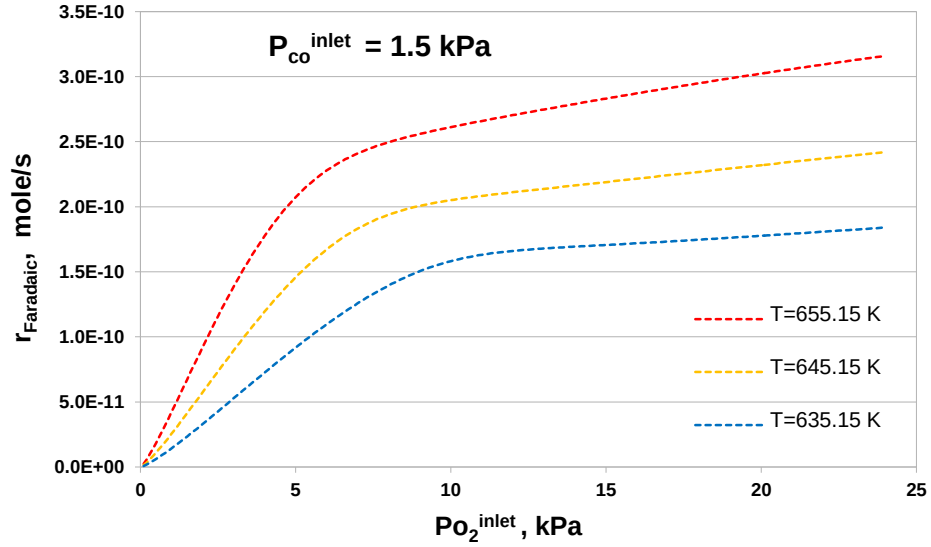


Figure 3.18: Effect of inlet partial pressure of O_2 on CO_2 Faradaic rate for various temperature values.

3.8 Conclusions

The aim of this chapter was the construction of a multi-dimensional, isothermal, dynamic solid oxide single pellet macroscopic model to describe the chemical and electrochemical processes taking place in a solid oxide pellet under the effect of electrochemical promotion. The proposed model comprises of PDEs for mass and charge conservation describing the chemical and electrochemical phenomena taking place at the catalytic surface and at the TPBs of the anode and cathode respectively. The simulation of the model was carried out with the use of COMSOL Multiphysics, a commercial CFD software which employs the FEM for the simultaneous solution of the resulting set of PDEs. In addition, the facility of "integrated coupling variables" incorporated in this CFD software was used in order to implement each gas species partial pressure expression. The proposed framework allows the prediction of species coverage on the catalytic surface, electronic and ionic potential profiles throughout the pellet, gas mixture concentration within the reactor and CO_2 production rates.

Since kinetic parameters such as activation energies and sticking coefficients for the closed-circuit catalytic reactions are expected to differ from the open-circuit ones and electrochemical parameters such as the exchange current density pre-exponential coefficients are not available, we have undertaken a parameter estimation study. To achieve this, a combination of stochastic and deterministic optimization techniques were performed using a tailored 2-D version of our model (through the use of the scripting facility of COMSOL Multiphysics) to fit model predictions with experimental data available in the literature (Yentekakis et al., 1994). Potential identifiability issues due to

the limited number of available data were addressed through multiple restarts for the stochastic optimizer. The computed parameters were within literature reported lower and upper limits. The resulting 2- and 3-D models predict physically realistic trends.

It should be noted here that an increase in P_{CO}^{in} induces a decrease in the BSS coverage as expected from the exchange current density expressions. Nevertheless, the effect of $P_{O_2}^{in}$ is not explicitly quantifiable from the available experimental data, hence further experimental information would be needed to quantify the effect of BSS coverage alterations on the catalyst work functions and hence on the kinetic constants.

Sensitivity analysis for the computed parameters was undertaken to obtain insights on the reliability of the parameter estimation problem and to quantify the effect of each parameter on the reaction rate. As a result, we found that variations in the chemical kinetic constants lead to significant changes in the CO_2 production rate, while the ones in the reaction rate constants of reactions where the BSS was participating in as well as in the exchange current density pre-exponential factors, have minor or almost no effect on the CO_2 production rate.

Reaction rate profiles have been generated to study the parametric effect of various operating parameters on the catalytic performance. It was observed that a rise in the P_{CO}^{in} favoured "volcano-type" behaviour in the CO_2 non-Faradaic production rate, while "S-type" behaviour was observed in the CO_2 Faradaic production rate. At low P_{CO}^{in} values, a sharp rise was observed in both Faradaic and non-Faradaic rates. On the other hand, increasing the $P_{O_2}^{in}$ led to "S-type" behaviour in the CO_2 production rate due to both non-Faradaic and Faradaic effects. Furthermore, the performance of the pellet was enhanced

increasing the temperature in the case of variable $P_{O_2}^{in}$ for $P_{O_2}^{in} / P_{CO}^{in} < 6$ and the opposite effect was favoured in the case of variable P_{CO}^{in} for higher than 1.5 kPa P_{CO}^{in} values.

Chapter 4

Multi-scale Modelling of EPOC Systems

4.1 Introduction

The main objective of this chapter is the development of an accurate multi-scale framework to describe the effect of electrochemical promotion and to obtain insights for the chemical micro-processes taking place on the catalytic film under closed-circuit conditions.

In the previous chapter we have presented a recently published work (Fragkopoulos et al., 2013) which accounts for the development of a multi-dimensional macroscopic model to describe the chemical and electrochemical processes taking place in an electrochemically promoted CO combustion system and to quantify the non-Faradaic catalytic performance enhancement upon polarisation. The proposed model was utilised in conjunction with literature data for the estimation of parameters that are of great importance for EPOC system simulations. Sensitivity analysis for the estimated parameters was carried out to quantify the effect of each parameter on the CO_2

production rate. We found that varying the electrochemical kinetic parameters has practically no impact on the predicted CO_2 production rate, while altering the chemical ones can lead to vast changes of the obtained CO_2 production rate.

This study established that the dominant effect in such a system is the non-Faradaic one. Consequently, since the NEMCA and Faradaic contributions to the system are taking place at different length-scales (the NEMCA effect is limited to the catalyst surface, while the Faradaic effect depends on the entire computational domain of the system), we have extended our macroscopic study to the formulation of a multi-scale framework to investigate the NEMCA (catalytic processes) and Faradaic (charge transport and TPB electrochemical processes) phenomena taking place in the system at their appropriate length-scales.

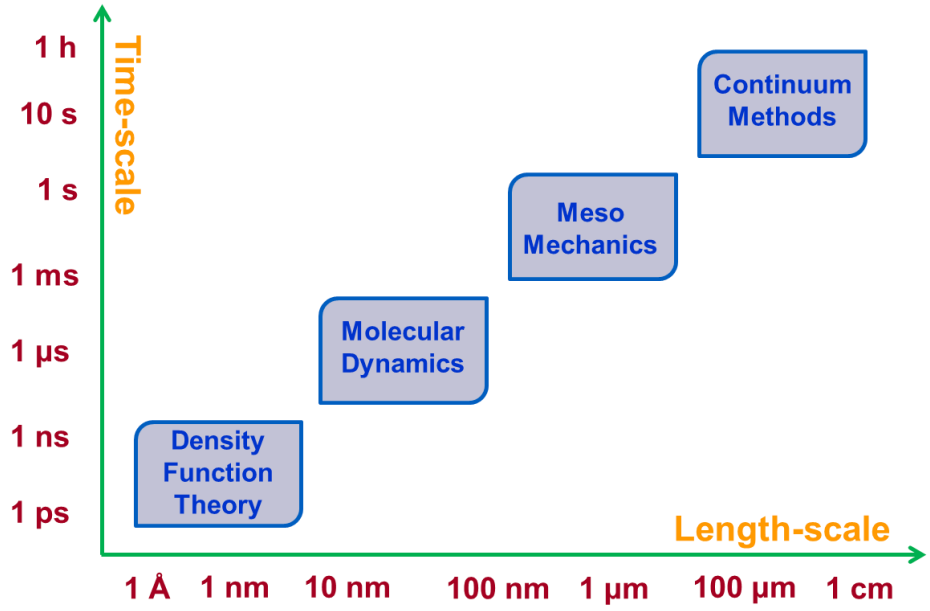


Figure 4.1: Characteristic length- and time-scales for various modelling methods.

Physical phenomena taken into consideration in engineering systems are by nature occuring at different length- and time-scales with varying degrees of complexity. Multi-scale modelling accounts for the construction of a composite mathematical framework that links two or more computational models describing such diverse and complex phenomena (Weinan, 2011). Figure 4.1 illustrates a relation between various levels of length- and time-scales with macro-, meso- and micro-scale computational approaches. The macro-scale approach corresponds to the entire flow domain (continuum methods), the micro-scale approach to the molecular level (molecular dynamics, density function theory) and the meso-scale approach to the level between the macro- and micro-scale ones (meso- mechanics).

Communication between the different scales (multi-scale modelling) can be attained through the use of hierarchical or hybrid (concurrent) methods (Andersson et al., 2010). In the hierarchical method, the simulation of the physical phenomena begins at a smaller scale and the calculated properties are delivered to a larger level, while the hybrid method accounts for more complex simulations where three (or more) different methods are interacting.

Furthermore, the classification of multi-scale models also relies on the technique that the submodels are coupled (Maroudas, 2000; Pantelides, 2001; Ingram et al., 2004). A multi-scale approach can consequently be considered as multidomain, embedded, parallel, serial and simultaneous (Ingram et al., 2004). Multidomain linking refers to the communication of two differently scaled (macro- and micro-scale) non-overlapping regions through an interface. In embedded communication, the smaller scale model is embedded in the larger scale model through interconnecting its properties. In parallel

communication, the submodels simulate phenomena occuring in the same domain, in a complementary manner. Serial coupling stands for systems where the transmission of information is strictly one-way, i.e. when the results of a small scale model are used in a larger scale model without a reverse influence. Simultaneous coupling accounts for the representation of the whole system at the micro-scale and the consequent conversion of the microscopic results into macroscopic variables (Ingram et al., 2004).

The multi-scale approach is very promising nowadays in the field of electrochemistry and especially in solid oxide fuel cell (SOFC) systems, where state-of-the-art frameworks (Khaleel et al., 2005; Bessler et al., 2007b; Kim et al., 2009; Lee and Hong, 2010) have been developed to link the transport phenomena taking place at the macro-scale, with electrochemical reactions and material properties accurately described at the meso- and micro-scales.

Khaleel et al. (2005) presented a multi-scale approach where the microscopic model employs the Lattice-Boltzmann method to investigate the performance of a porous electrode, calculating the overall SOFC current-voltage relation and taking into consideration the structure of the electrode at the micro-structure scale, transportation of oxygen ions as well as reaction surfaces distribution. The evaluated current-voltage relation is subsequently fed to the macroscopic model which performs cell voltage, current density and heat production calculations employing the finite elements method.

Bessler et al. (2007b) performed multi-scale simulations to describe transport in an internal-reforming SOFC operated on CH_4/H_2O mixtures. The framework is integrated by a macroscopic model which is used for simulating the transport of the gas-phase in

the SOFC channel, the charge transport as well as continuum mass in the porous electrodes, and a microscopic model which is employed for the simulation of mass transport over the triple phase boundaries.

Kim et al. (2009) formulated a multi-scale framework to predict the performance of a SOFC when micro-structure evolution takes place at the anode. In that study, the framework integrates a macroscopic model which simulates the charge and mass transport as well as the electrochemical phenomena, and local material properties such as triple phase boundary density, electrical conductivity and gas diffusivity are calculated through evolving the micro-structure.

Lee and Hong (2010) developed a multi-scale technique for the design of a novel intermediate-temperature planar-type micro SOFC stack system. The proposed multi-scale approach, couples a computational fluid dynamics (macroscopic) model which is employed for the simulation of the fuel and air flows, with molecular dynamics (microscopic) simulations which are employed for determining the optimal composition of an electrolyte used in intermediate temperature simulations.

Despite the increasing use of multi-scale frameworks in SOFC systems, such a modelling approach, has not been formulated yet for the simulation of electrochemically promoted systems.

This chapter presents an under submission work (Fragkopoulos and Theodoropoulos, 2014), where we have constructed a multi-scale framework to describe the effect of electrochemical promotion in a solid oxide single pellet system. The proposed framework links a 3-D macroscopic model used to perform charge transport simulations

throughout the solid oxide single pellet employing CFD software (COMSOL Multiphysics), with a 2-D microscopic model used to simulate the reaction-diffusion processes taking place at the anode working electrode (catalytic surface) employing the kinetic Monte Carlo method. As in the previous chapter, the electrochemically promoted CO oxidation over Pt/YSZ is chosen as an illustrative scheme. To examine the added value of such a multi-scale model, comparisons between the proposed framework and the macroscopic model presented in the previous chapter dynamic and steady state outputs are performed for various sets of operating conditions. Subsequent parametric investigation using the multi-scale framework is carried out to analyse the effects of the partial pressure and temperature on the CO_2 production rate and finally, outcomes from the above study are discussed.

4.2 Electrochemically promoted multi-scale CO combustion

The electrochemically promoted multi-scale CO combustion over Pt/YSZ is used here as an extension to the work presented in the previous chapter. The reactor utilised in the proposed system is considered as well mixed and is of single-pellet type meaning that the entire pellet is exposed to the reacting gas mixture. The reactor design and the 3D computational domain of the single pellet are illustrated in Figure 4.2, where Yttria Stabilized Zirconia is utilised as electrolyte, while Pt and Au are used as anode and cathode electrodes respectively. The physical dimensions of the single pellet are tabulated in Table 4.1 and it should be noted that both Pt and Au electrodes are considered as 2-dimensional (i.e. negligible thickness) layers on YSZ.

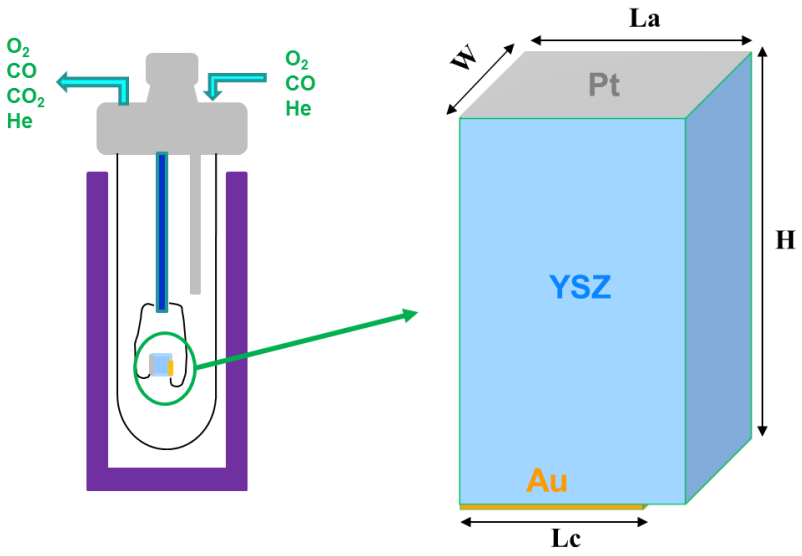


Figure 4.2: Reactor design and 3-D computational domain.

Table 4.1: The physical dimensions of the solid oxide single pellet.

Parameter	*Symbol*	*Value*
Electrolyte/Anode length, m	L_a	$4.6x10^{-7}$
Cathode length, m	L_c	$4x10^{-7}$
Electrolyte/Anode/Cathode width, m	W	$1.023x10^{-7}$
Electrolyte height, m	H	$5x10^{-6}$

The proposed multi-scale framework integrates the reaction-diffusion phenomena taking place on the catalytic surface (Pt) and the charge transport throughout the pellet as well as the electrochemical processes taking place at the triple phase boundaries. The catalytic surface processes for this system are represented in Figure 4.3, while the mechanisms of the chemical (CO oxidation reaction mechanism as proposed by Kaul et al. (1987) augmented by reactions with backspillover species (BSS) due to the circuit

closure) and of the electrochemical processes taken into consideration in this study are tabulated in Tables 4.2 and 4.3 respectively.

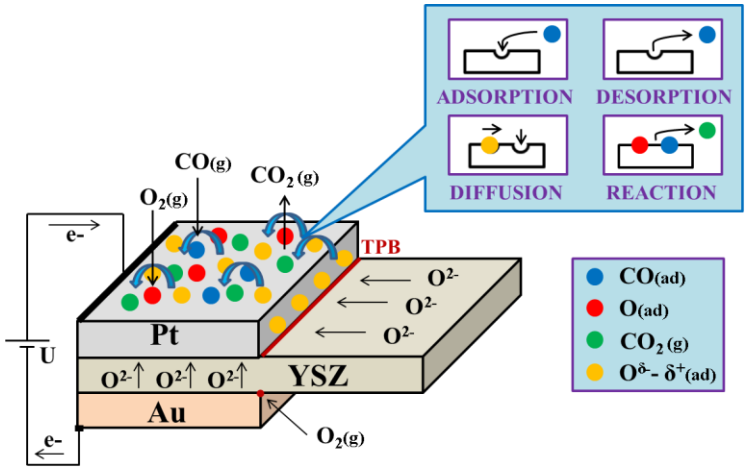


Figure 4.3: Schematic presentation of the electro-catalytic surface dynamics.

Table 4.2: The scheme of catalytic surface micro-processes.

Open-circuit CO oxidation micro-processes		
Adsorption/desorption of O_2	$O_{2\,(g)} + 2\cdot S \underset{k_{-1}^{d}}{\overset{k_{1}^{a}}{\rightleftharpoons}} 2O\cdot S$	(4.1)
Adsorption/desorption of CO	$CO_{(g)} + \cdot S \underset{k_{-2}^{d}}{\overset{k_{2}^{a}}{\rightleftharpoons}} CO\cdot S$	(4.2)
Surface reaction between adsorbed CO and O	$O\cdot S + CO\cdot S \xrightarrow{k_{3}^{r}} CO_{2\,(g)} + 2\cdot S$	(4.3)
Closed-circuit additional micro-processes		
Surface reaction between adsorbed CO and BSS	$BSS\cdot S + CO\cdot S \xrightarrow{k_{4}^{r}} CO_{2\,(g)} + 2\cdot S$	(4.4)
Desorption of BSS	$2BSS\cdot S \xrightarrow{k_{5}^{d}} O_{2\,(g)} + 2\cdot S$	(4.5)
BSS Surface diffusion	$BSS\cdot S + \cdot S \xrightarrow{k_{diff}^{BSS*}} \cdot S + BSS\cdot S$	(4.6)

Table 4.3: The scheme of electrochemical reactions.

Cathodic TPB:	$3\times\left[\frac{1}{2}O_{2\,(g)} + 2e^{-} \rightarrow O_{YSZ}^{2-}\right]$	(4.7)

	$O^{2-}_{YSZ} + CO_{(g)} \rightarrow CO_{2\,(g)} + 2e^-$	(4.8)
Anodic TPB:	$O^{2-}_{YSZ} \rightarrow \frac{1}{2} O_{2\,(g)} + 2e^-$	(4.9)
	$O^{2-}_{YSZ} \rightarrow BSS + 2e^-$	(4.10)

COMSOL Multiphysics 3.5a (COMSOL, 2008), is employed for the simulation of the charge conservation in the pellet at the macroscopic level, while an in-house developed lattice kinetic Monte Carlo model (Hari and Theodoropoulos, 2009; Fragkopoulos et al., 2012; Fragkopoulos and Theodoropoulos, 2013) is utilised for the simulation of the reaction-diffusion micro-processes taking place on the catalytic surface at the microscopic level. The proposed framework can be used to obtain electronic and ionic potential curves throughout the pellet as well as species' coverage profiles at the micro-catalytic surface. It can also provide transients of the gas mixture concentration in the reactor and of the CO_2 production rate.

4.3 Model assumptions

Similar to chapter 3, the main assumptions that have been made for this 3-D multi-scale framework are listed below:

- The gas phase pressure in the reactor as well as the temperature on the catalytic surface and throughout the reactor are considered constant.
- The gaseous mixture in the reactor is considered to be well-mixed and to behave as an ideal gas.

- The framework accounts only for the catalytic micro-processes on Pt while the catalytic effect of Au (in presence of Pt), can be neglected (Yentekakis et al., 1994).
- Only the diffusion micro-process of BSS is taken into account as a simplification to the mass transport on the Pt, since for the other species the dominant micro-processes are the adsorption and desorption ones.
- Assuming that both electrolyte and electrodes are good charge conductors, the operating potential difference throughout the pellet is considered to be constant.
- The electrodes of the cathode and the anode are modelled as 'flat' (2-D) surfaces deposited on the electrolyte domain.
- The electrochemical reactions are considered to take place only at the anodic and cathodic TPBs, represented here by the edges of the respective electrodes.
- The chemical potential of BSS is assumed to be equal with the one of the O^{2-} on the surface of YSZ ($\mu_{BSS} = \mu_{O^{2-}_{YSZ}}$) (Fragkopoulos et al., 2013).

4.4 The microscopic model

The microscopic model used here, employs a stochastic methodology for the simulation of the catalytic surface dynamics. Gillespie (1976, 1977) was one of the first authors to use a stochastic approach for simulations of simple homogeneous reacting systems. This probabilistic approach (an extension of the well-known Monte Carlo method proposed by Metropolis and Ulam (1949)) has since been extensively improved and employed for complex heterogeneously catalysed reaction system kinetic

simulations where the catalytic surface is either represented by a lattice taking into account the spatial distribution and the local micro-kinetics (Reese et al., 2001; Raimondeau and Vlachos, 2002b) or not taking into consideration any spatial representation (Dooling and Broadbelt, 2001; Zhdanov and Kasemo, 2002, 2003). In the cases where the catalytic surface is represented by a lattice, this method is referred to as the lattice kinetic Monte Carlo.

The kinetic Monte Carlo method is used to perform stochastic simulations for the probabilistic master equation (Fichthorn and Weinberg, 1991):

$$\frac{\partial P_{n,t}}{\partial t} = \sum_{m} \left[W_{m \to n} P_{m,t} - W_{n \to m} P_{n,t} \right] \tag{4.11}$$

where n and m are successive configurations of the catalytic surface (represented by a lattice as in Figure 4.4a), $P_{n,t}$ is the probability that the lattice is in configuration n at time t, and $W_{m \to n}$ is the probability per unit time that the lattice will undergo a transition from configuration m to configuration n. As an example, considering that the catalyst lattice is initially in a configuration m depicted in Figure 4.4b (where lattice positions - boxes- are occupied by species of the type A), then at a time t it can be in configuration n or p illustrated in Figures 4.4c and 4.4d respectively, among several others.

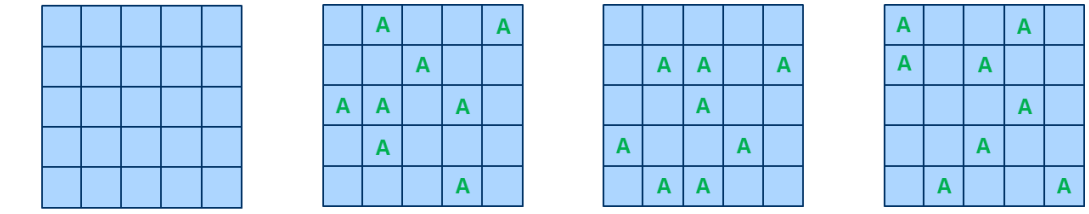


Figure 4.4: Lattice representation of the catalytic surface and possible lattice configurations.

Due to the large number of possible configurations, the solution of the master equation (4.11) cannot be achieved analytically for real system simulations. For this reason, Monte Carlo algorithms are often employed to perform the computational intensive simulations efficiently.

In this work, we have employed an in-house developed lattice kMC model based on the Continuous Time Monte Carlo (CTMC) algorithm proposed by Reese et al. (2001), for the simulation of the reaction-diffusion micro-processes taking place on the catalytic lattice. The main features of the CTMC algorithm exploited here are summarised in the following 5 steps:

1. *Initialisation of the lattice.* The surface species, i.e. A* (O*), B* (CO*), C* (BSS*), are randomly positioned on the micro-lattice (as in Figure 4.5) according to the given initial respective number of molecules.

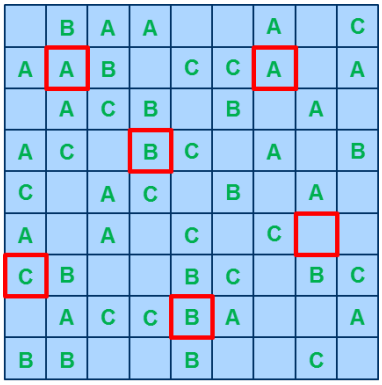


Figure 4.5: Schematic of a random distribution of species on the lattice.

2. *Creation of classes.* After distributing the species on the catalytic lattice, classes (of species) are formed and the size of each class is computed. Classes represent combinations (or pairs) of reactive surface species. The size (total number) of each class is used in the calculation of conditional probabilities (see Eq. (4.23)). The

conditional probabilities are necessary for obtaining the micro-processes rates (termed as transition probabilities) which will be described later in this chapter (Eq. (4.14) to (4.19)).

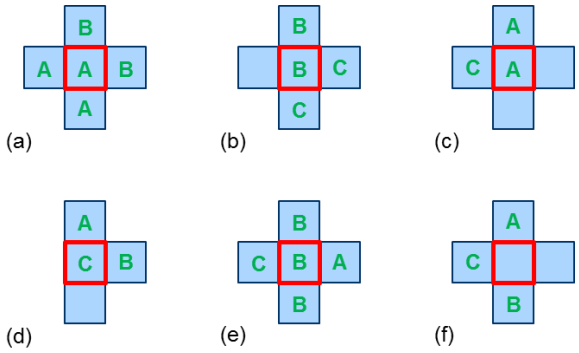


Figure 4.6: Schematic presentation of classes of six selected positions of species.

Figure 4.6a-f illustrates six selected lattice positions occupied by different species (red highlighted -boxes- species positions in Figure 4.5) along with their neighbours, i.e. species occupying adjacent lattice positions. Taking into account the distribution of species illustrated in Figure 4.6a as an example to describe the size of the associated classes we have:

- 1 class of A*,A*,2 (meaning that adsorbed species A has two adsorbed neighbours of the type A).
- 1 class of A*,B*,2 (meaning that adsorbed species A has two adsorbed neighbours of the type B).

Similarly, for the other selected species (Figures 4.6b-f) we have:

- *Figure 4.6b:* 1 class of B*,B*,1, 1 class of B*,C*,2 and 1 class of B*,*,1 (where a single * represents an empty lattice site).
- *Figure 4.6c:* 1 class of A*,A*,1, 1 class of A*,C*,1 and 1 class of A*,*,2.

- *Figure 4.6d:* 1 class of C*,A*,1, 1 class of C*,B*,1 and 1 class of C*,*,1 (it is worthwhile to note that only three adjacent sites are taken into consideration here since the 4th neighbour is positioned outside the lattice).
- *Figure 4.6e:* 1 class of B*,B*,2, 1 class of B*,A*,1 and 1 class of B*,C*,1.
- *Figure 4.6f:* 1 class of *,A*,1, 1 class of *,*,1, 1 class of *,B*,1 and 1 class of *,C*,1.

To evaluate the size of each class (total number of each *A**,*B**,*n* pair of species), we follow the procedure described above selecting all the lattice sites one by one and counting the types of all classes and subsequently summing the numbers that correspond to classes of the same type.

3. *Selection of a micro-process.* The selection of a micro-process to take place comes next. The selection of the *k*th micro-process is performed probabilistically using a random number $R_1 \epsilon (0,1)$ in the inequality that follows:

$$\sum_{i=1}^{k-1} \hat{\Gamma}_i < R_1 \hat{\Gamma}_T < \sum_{i=1}^{k} \hat{\Gamma}_i, \qquad \text{with } R_1 \in (0,1) \quad \& \quad (1 < k < n_r) \tag{4.12}$$

where $\hat{\Gamma}_i$ is the transition probability of micro-process i, n_r is the total number of micro-processes and $\hat{\Gamma}_T$ is the total transition probability which is expressed as the summation of the individual transition probabilities:

$$\hat{\Gamma}_T = \sum_{i=1}^{n_r} \hat{\Gamma}_i \tag{4.13}$$

As an example, taking into account 3 micro-processes with transition probabilities 5 s^{-1}, 10 s^{-1} and 20 s^{-1} gives a total transition probability of 35 s^{-1} (as in Figure 4.7) and consequently, multiplying it by a random number equal to 0.65 gives

22.75 s^{-1}. Such a value for the transition probability falls into the area of the 3rd micro-process (see also Figure 4.7) which is subsequently selected.

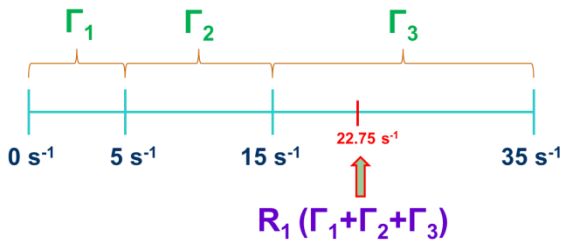


Figure 4.7: Schematic of micro-process selection process using Eq. (4.12).

The transition probability for each micro-process *i* (as in Table 4.2) is given by:

$$\hat{\Gamma}_1 = k_1^a P_* P_{*/*} \qquad \& \qquad \hat{\Gamma}_{-1} = k_{-1}^d P_{O^*} P_{O^*/O^*} \tag{4.14}$$

$$\hat{\Gamma}_2 = k_2^a P_* \qquad \& \qquad \hat{\Gamma}_{-2} = k_{-2}^d P_{CO^*} \tag{4.15}$$

$$\hat{\Gamma}_3 = k_3^r \left(P_{O^*} P_{CO^*/O^*} + P_{CO^*} P_{O^*/CO^*} \right) \tag{4.16}$$

$$\hat{\Gamma}_4 = k_4^r \left(P_{BSS^*} P_{CO^*/BSS^*} + P_{CO^*} P_{BSS^*/CO^*} \right) \tag{4.17}$$

$$\hat{\Gamma}_5 = k_5^d P_{BSS^*} P_{BSS^*/BSS^*} \tag{4.18}$$

$$\hat{\Gamma}_{diff} = k_{diff}^{BSS} \left(P_{BSS^*} P_{*/BSS^*} + P_* P_{BSS^*/*} \right) \tag{4.19}$$

where k_i^a, k_i^d and k_i^r are the adsorption, desorption and surface reaction rate constants respectively, k_{diff} is the diffusion micro-process rate constant, P_{A^*} and P_{A^*/B^*} are one and two site conditional probabilities, respectively.

The individual adsorption rate constants, k_i^a, are expressed using the gas collision theory as (Raimondeau and Vlachos, 2002b; Guerrero and Wolf, 2011):

$$k_i^a = \frac{S_j P_T X_j}{N_S} \left(\frac{1}{2\pi M_j R T_S} \right)^{1/2}, \quad i = 1 \; (j = O_2) \quad \& \quad i = 2 \; (j = CO) \tag{4.20}$$

where S_j is the sticking coefficient of gaseous species j, P_T is the system operating pressure, X_j is the mole fraction of gas species j, N_s is the concentration of active sites on the catalytic surface, M_j is the molecular weight of species j and T_s is the temperature of the catalytic surface.

The individual desorption, k_i^d , and surface reaction , k_i^r , rate constants follow the Arrhenius expression:

$$k_i^{d,r} = k_{o,i}^{d,r} \exp\left(\frac{-E_{A,i}^{d,r}}{RT_S}\right), \qquad i = \text{-}1, \text{-}2,\ 3 \tag{4.21}$$

where $k_{o,i}$ and $E_{A,i}$ are the pre-exponential factor and the activation energy for each micro-process i, respectively.

The diffusion micro-process rate constant, k_{diff} , can be related with the continuum diffusion coefficient, D_i, using a similar correlation to the one presented in (Phillips, 1998; Tello and Curtin, 2005):

$$k_{diff} = D_i \frac{1}{A_S} = D_i \frac{N_S N_{AV}}{\Omega_T} \tag{4.22}$$

where N_{AV} is the Avogadro constant, N_s is the catalyst site density, Ω_T is the total number of lattice sites and A_S is the catalytic surface area.

The one site conditional probability, P_{A*}, expresses the probability of selecting a site occupied by A*, while the two site one, $P_{A*/B*}$, expresses the probability of picking a site occupied by B* after choosing an adjacent site occupied by A*. The conditional probabilities P_{A*} and $P_{A*/B*}$ are given by (Reese et al., 2001):

$$P_{A^*} = \frac{\Omega_{A^*}}{\Omega_T} \qquad \& \qquad P_{A^*/B^*} = \frac{\sum_{j=1}^{4} j \cdot \left(\Omega_{B^*A^*j}\right)}{4 \cdot \Omega_{B^*}} \tag{4.23}$$

where Ω_{A*} is the number of sites occupied by A*, Ω_T is the total number of lattice sites, Ω_{B*A*j} is the number of sites of identity B* that have *j* adjacent sites of identity A* (also referred to as size of class B*,A*,j). Number 4 in the denominator of Eq. (4.23) is the maximum number of the adjacent sites that a selected site can have (the diagonally adjacent sites are not taken into account in this scheme).

4. *Selection of lattice sites and reaction.* Having found probabilistically (using Eq. (4.12)) which micro-process is to occur, we randomly select a class associated with the chosen micro-process (i.e. if the adsorption of O_2 is chosen as the microprocess to take place, we randomly select one of the (*,*,*1)*, (*,*,*2)*, (*,*,*3)* and (*,*,*4)* classes which are related to this micro-process). Consequently, lattice sites that belong to the selected class are randomly picked (i.e. if the *,*,2 class is picked, then we randomly select one of the lattice empty sites that has two empty sites as neighbours) and the micro-process takes place. Finally, update of the number of surface and gaseous species, of the size of classes and subsequently of the conditional and of the transitional probabilities follows.

5. *Update of time variable.* After the chosen micro-prosses takes place, the time interval, $\Delta t_{\text{micro-process}}$, due to this kMC event is calculated using a random number R_2: (Gillespie, 1977; Guerrero and Wolf, 2011):

$$\Delta t_{micro-process} = \ln\left(\frac{1}{R_2}\right)\frac{1}{\Omega_T \hat{\Gamma}_T}, \qquad \text{with } R_2 \in (0,1) \tag{4.24}$$

Consequently, the simulation time is updated using the following expression:

$$t_j = t_{j-1} + \Delta t_{micro-process} \tag{4.25}$$

where t_{j-1} is the value for the time before each micro-process takes place and t_j is the updated value for the time (after each micro-process takes place).

After updating the time interval, we start again from step 3 (selection of micro-process) and we follow the same process. This iterative process is continued until the given time reporting horizon, t_{rep}^{kMC}, is reached.

The gaseous phase species' rates due to the catalytic micro-processes taken into consideration in the kMC algorithm are given by:

$$\hat{R}_i^{kMC} = \frac{molecules\ of\ i}{N_{AV} t_{rep}^{kMC}} \tag{4.26}$$

where N_{AV} is the Avogadro constant, t_{rep}^{kMC} is the time reporting horizon used for every kMC run and *molecules of i* is the number of molecules of species i produced (positive sign) or consumed (negative sign) within the kMC time reporting horizon.

4.5 The macroscopic model

The macroscopic model utilised here consists of a set of PDEs to describe the charge conservation in the solid oxide single pellet. The charge transport in a non-porous media *i*, is described by the Poisson equation as follows (Cheddie and Munroe, 2006; Tseronis, 2009):

$$\frac{d\rho_i}{dt} = -\nabla(-\sigma_i \nabla \Phi_i) + Q_i \tag{4.27}$$

where ρ_i is the charge density, σ_i is the charge conductivity, Φ_i is the local electrostatic potential and Q_i is the charge source term.

Potential application between the anode and cathode electrodes in the solid oxide single pellet leads to transportation of charge throughout the pellet, and to exchanges of ionic to electronic charge and the vice versa, at the anodic and cathodic TPBs due to the electrochemical reactions taking place there (see Table 4.3). Considering no charge source, the ionic and electronic charge conservation in the electrolyte and at the two electrodes respectively, can be described as follows:

$$\frac{d\rho_{io}}{dt} = -\nabla(-\sigma_{io}\nabla\Phi_{io}) \tag{4.28}$$

$$\frac{d\rho_{el}^{A/C}}{dt} = -\nabla(-\sigma_{el}^{A/C}\nabla\Phi_{el}^{A/C}) \tag{4.29}$$

where ρ_{io} and $\rho_{el}^{A/C}$ are the ionic and electronic charge densities, σ_{io} and $\sigma_{el}^{A/C}$ are the ionic and electronic conductivities, and, Φ_{io} and $\Phi_{el}^{A/C}$ are the ionic and electronic potentials respectively. The superscripts A/C stand for the Anode/Cathode electrode domains.

The gaseous phase species' Faradaic rates and the BSS Faradaic generation rate due to the electrochemical reactions (4.7) to (4.10) occuring at the TPBs of the cathode and the anode are described by the following expressions:

$$\hat{R}_{CO_2}^{FEM} = \int_0^W \int_0^{L_a} \frac{I_8^A}{2F}\,\mathrm{dxdz} \tag{4.30}$$

$$\hat{R}_{CO}^{FEM} = -\int_0^W \int_0^{L_a} \frac{I_8^A}{2F}\,\mathrm{dxdz} \tag{4.31}$$

$$\hat{R}_{O_2}^{FEM} = \int_0^W \int_0^{L_a} \frac{I_9^A}{4F}\,\mathrm{dxdz} - \int_0^W \int_0^{L_c} \frac{I^C}{4F}\,\mathrm{dxdz} \tag{4.32}$$

$$\hat{R}_{BSS}^{FEM} = \int_0^W \int_0^{L_a} \frac{I_{10}^A}{2F}\,\mathrm{dxdz} \tag{4.33}$$

where W is the width of both anode and cathode electrodes, $L_{a/c}$ is the length of anode and cathode electrodes respectively, F is the Faraday constant, $I_i^{A/C}$ denotes the current density distribution of electrochemical reactions i for the Anode/Cathode computed by Eq. (4.37) and Eq. (4.43), respectively.

4.5.1 Boundary conditions

The numbering of boundaries (B) and edges (E) of the 3D computational domain is presented in Figure 4.8. There are in total 7 boundaries and 15 edges, 6 of which, E2-4, E6 and E10-11 represent TPBs. It should be noted that boundaries B1 and B3 represent the anodic (Pt) and cathodic (Au) electrodes respectively.

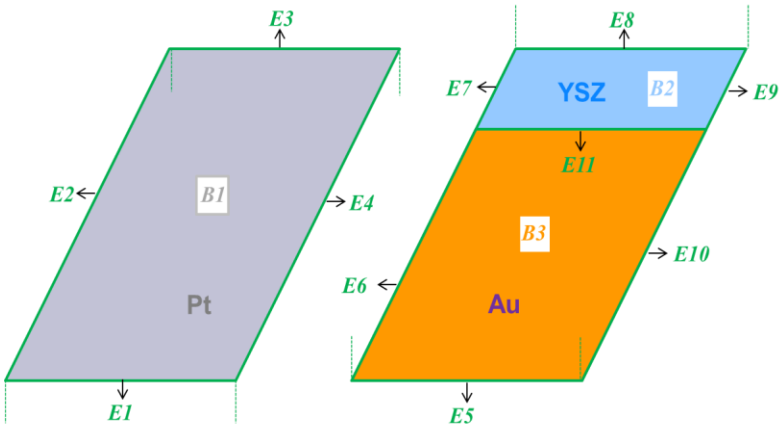


Figure 4.8: The 3-D Computational domain numbering of boundaries and edges.

For the solution of the set of PDEs that describe the charge transfer in the pellet, boundary conditions need to be imposed. The electronic potential is fixed to the value of the operating potential Φ_{op} at edge E5 and to 0 at E1.

E5: $$\Phi_{el}^{C} = \Phi_{op} \tag{4.34}$$

E1: $$\Phi_{el}^{A} = 0 \tag{4.35}$$

Transfer of electronic to ionic current occurs at the TPBs of the cathode that are represented by E6 and E10-11. Hence:

E6 & E10-11: $$\begin{aligned} -\mathbf{n}\cdot\left(-\sigma_{el}^{C}\nabla\Phi_{el}^{C}\right) &= I^{C} \\ -\mathbf{n}\cdot\left(-\sigma_{io}\nabla\Phi_{io}\right) &= -I^{C} \end{aligned} \tag{4.36}$$

where I^C is the current density distribution of the cathode which is due to the electrochemical reaction (4.7) taking place at the cathodic TPBs. The current density of the cathode can be expressed using the Butler-Volmer equation as:

$$I^{C} = 3\times I_{0}^{C}\left[\exp\left(\alpha^{C}\frac{n_e F}{RT}\eta^{C}\right) - \exp\left(-(1-\alpha^{C})\frac{n_e F}{RT}\eta^{C}\right)\right] \tag{4.37}$$

where I_0^C is the exchange current density of the cathode the expression of which can be found in Fragkopoulos et al. (2013), n_e is the number of electrons transferred in the cathodic electrochemical reaction, α^C is the cathodic charge transfer coefficient and η^C is the overpotential of the cathode. The factor of 3 in the above expression is due to the parallel electrical circuit analogy (Achenbach, 1994; Fragkopoulos et al., 2013).

The overpotential of the cathode is defined as (Bove and Ubertini, 2006):

$$\eta^{C} = V_{OC} - (\Phi_{el}^{C} - \Phi_{io}) \tag{4.38}$$

where Φ_{el}^{C} and Φ_{io} are the local equilibrium potentials of the cathode and the electrolyte mediums respectively, and V_{OC} is the thermodynamic open circuit potential given by (Fragkopoulos et al., 2013):

$$V_{OC} = V_{OC}^{o} - \frac{1}{2F}\mu_{BSS}^{A} + \frac{RT}{2F}\ln\left(\frac{P_{CO}^{A}}{P_{CO_2}^{A}}\right) + \frac{RT}{2F}\ln\left(\frac{\left(P_{O_2}^{C}\right)^{3/2}}{\left(P_{O_2}^{A}\right)^{1/2}}\right) \tag{4.39}$$

where μ_{BSS}^{A} is the chemical potential of the BSS, F is the Faraday constant, P_i is the partial pressure for species i in the gas phase (subscripts A and C stand for anode and cathode respectively) and V_{OC}^{o} is the *ideal Nernst potential* expressed as (Fragkopoulos et al., 2013):

$$V_{OC}^{o} = \frac{1}{2F}\left(\mu_{O_2}^{o} + \mu_{CO}^{o} - \mu_{CO_2}^{o}\right) \tag{4.40}$$

where μ_i^o is the chemical potential of species i at standard conditions.

Ionic charge is transferred to electronic at edges E2-4 which represent the anodic TPBs. Hence:

E2-4:
$$\begin{aligned} -\mathbf{n}\cdot\left(-\sigma_{io}\nabla\Phi_{io}\right) &= I^{A} \\ -\mathbf{n}\cdot\left(-\sigma_{el}^{A}\nabla\Phi_{el}^{A}\right) &= -I^{A} \end{aligned} \tag{4.41}$$

where I^A is the total current density of the anode which is due to the electrochemical reactions (4.8) to (4.10) taking place at the anodic TPBs. The total current density of the anode can be expressed using the parallel electrical circuit analogy as (Achenbach, 1994):

$$I^{A} = I_8^{A} + I_9^{A} + I_{10}^{A} \tag{4.42}$$

where I_8^A, I_9^A and I_{10}^A are the current density distributions due to the electrochemical reactions (4.8), (4.9) and (4.10), respectively, given by the following expression (Aguiar et al., 2004; Tseronis et al., 2008):

$$I_i^{A} = I_{0,i}^{A}\left[\exp\left(\alpha^{A}\frac{n_e F}{RT}\eta^{A}\right) - \exp\left(-(1-\alpha^{A})\frac{n_e F}{RT}\eta^{A}\right)\right], \quad \mathrm{i} = 8,9,10 \tag{4.43}$$

where $I_{0,i}^{A}$ is the exchange current density of the anode for each electrochemical reaction i, the expression of which can be found in Fragkopoulos et al. (2013), α^{A} is the anodic charge transfer coefficient and η^{A} the overpotential of the anode.

The overpotential of the anode is given by:

$$\eta^{A} = \Phi_{el}^{A} - \Phi_{io} \tag{4.44}$$

where Φ_{el}^{A} and Φ_{io} are the electronic and ionic potentials at the anode side respectively.

At all the remaining boundaries and edges the no flux condition is imposed for both ionic and electronic charge transfer.

4.6 The FEM/kMC multi-scale framework coupling

A schematic presentation of the multi-scale framework algorithm is illustrated in Figure 4.9.

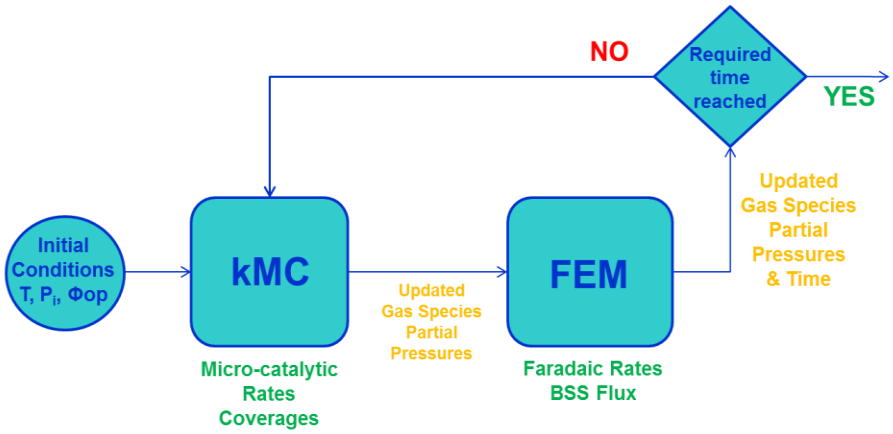


Figure 4.9: Schematic of the multi-scale framework algorithm.

Initial conditions such as temperature, T, gaseous species' partial pressures, P_i, and operating potential, Φ_{op}, are fed into the microscopic simulator and at the end of a time

reporting horizon (chosen here as t_{rep}^{kMC} =10^{-5} s), the model provides us with each gaseous species' production/consumption rates, $\hat{R}_i^{kMC}$, computed by Eq. (4.26). The partial pressures of the gaseous species are then updated using Eq. (4.45) to (4.48) taking into account the calculated micro-catalytic rates ($\hat{R}_i^{kMC}$). It is worthwhile to note here that for the first update of the partial pressures, the Faradaic rates computed by Eq. (4.30) to Eq. (4.32) are not taken into account ($\hat{R}_i^{FEM} = 0$, as the macroscopic run has not yet been performed), while the use of the previous iteration's values is necessary for the rest of the updates. The computed partial pressures are subsequently fed into the macroscopic simulator and at the end of the same time reporting horizon, the model provides us with the individual Faradaic rates (Eq. (4.30) to (4.32)) and a Faradaic generation term for BSS (Eq. (4.33)). The partial pressures of the gaseous species are updated again, now taking into account the values of the current $\hat{R}_i^{FEM}$ and of the previously calculated $\hat{R}_i^{kMC}$, and are fed back to the microscopic simulator also providing a flux for BSS. This process is continued until the total given time is reached.

The partial pressures of the gaseous species are updated using the following expressions:

$$P_{CO_2}^{new} = \frac{RT}{F_d}\left(\hat{R}_{CO_2}^{FEM} + \hat{R}_{CO_2}^{kMC}\right) \tag{4.45}$$

$$P_{CO}^{new} = P_{CO}^{in} + \frac{RT}{F_d}\left(\hat{R}_{CO}^{FEM} + \hat{R}_{CO}^{kMC}\right) \tag{4.46}$$

$$P_{O_2}^{new} = P_{O_2}^{in} + \frac{RT}{F_d}\left(\hat{R}_{O_2}^{FEM} + \hat{R}_{O_2}^{kMC}\right) \tag{4.47}$$

$$P_{He}^{new} = P_T - (P_{O_2}^{new} + P_{CO}^{new} + P_{CO_2}^{new}) \quad (4.48)$$

where P_i^{new} is the updated partial pressure of species i, P_i^{in} is the partial pressure of species i at the inlet of the reactor, F_d is the volumetric flowrate of the gas mixture, R is the ideal gas constant, $\hat{R}_i^{FEM}$ and $\hat{R}_i^{kMC}$ are each species i Faradaic and kMC rates (in mol/s units) computed by Eq. (4.30) to (4.32) and Eq. (4.26), respectively.

The macroscopic generation of oxygen BSS ([O^{2-}- δ^{2+}]), using a similar correlation to the one presented in Karavasilis et al. (1996) for sodium backspillover species ([Na^+ - δ^-]), is given by:

$$\frac{d\theta_{BSS}}{dt} = \frac{\hat{R}_{BSS}^{FEM}}{N_s A_s} \quad (4.49)$$

where $\hat{R}_{BSS}^{FEM}$ is the Faradaic generation of BSS (Eq.(4.33)), N_S is the catalyst site density and A_S is the catalytic surface area.

In terms of number of lattice sites occupied by BSS, Ω_{BSS^*}, taking into account that $\theta_{BSS} = \Omega_{BSS^*} / \Omega_T$ and $A_s = \Omega_T / N_s N_{AV}$, Eq. (4.49) takes the form:

$$\frac{d\Omega_{BSS^*}}{dt} = \hat{R}_{BSS}^{FEM} \; N_{AV} \quad (4.50)$$

and subsequently (for relatively small utilised time reporting horizons, t_{rep}^{kMC}):

$$\frac{\Omega_{BSS^*}\left(t + t_{rep}^{kMC}\right) - \Omega_{BSS^*}\left(t\right)}{t_{rep}^{kMC}} = \hat{R}_{BSS}^{FEM} \; N_{AV} \quad (4.51)$$

where N_{AV} is the Avogadro constant, $\Omega_{BSS^*}\left(t + t_{rep}^{kMC}\right)$ is the number of BSS molecules at the end of the time reporing horizon and $\Omega_{BSS^*}\left(t\right)$ is the number of BSS molecules at the beginning of the time reporing horizon.

Thus, the total number of BSS molecules, n_{BSS}, that need to be introduced into the kMC simulator during every time reporting horizon, taking into consideration that $n_{BSS} = \Omega_{BSS^*}\left(t + t_{rep}^{kMC}\right) - \Omega_{BSS^*}\left(t\right)$, is given by:

$$n_{BSS} = \hat{R}_{BSS}^{FEM} \; N_{AV} \; t_{rep}^{kMC} \tag{4.52}$$

The calculated BSS molecules that need to be introduced into the microscopic simulator are represented by a continuous variable. The integer part of it, represents full molecules that are fed into the system, while the decimal part is considered as a fractional molecule. When the fractional part of the continuous variable exceeds the unit value, one BSS molecule is introduced into the system and the continuous variable is updated. As an example, in the case of a number of BSS molecules need to be introduced, $n_{BSS} = 1.2$, and for a utilised time reporting, $t_{rep}^{kMC} = 10^{-3}\,\mathrm{s}$, we introduce 1 molecule of BSS every 1 ms, and every 5 ms we introduce one more BSS molecule, i.e. in total 6 molecules per 5 ms.

Szymczak and Ladd (2004) suggest that in a continuous process, it is not enough to just introduce the correct number of molecules in a domain but also the technique for this introduction is important. They found that is more accurate to release the molecules continuously over a time horizon rather than at the beginning of it. Taking into consideration this, the molecules (calculated using Eq.(4.52)) are introduced into the kMC simulator using a timestep randomly chosen from a uniform distribution in the range [t, $t+t_{rep}^{kMC}$] through being randomly placed at positions of empty sites available at the lattice boundaries.

4.7 Numerical solution approach

COMSOL Multiphysics 3.5a is utilised as the partial differential equation solver where the finite elements method is employed for the simultaneous solution of the set of PDEs described by equations (4.28) and (4.29) at the macroscopic level. The 3-D computational domain is discretised in 38,876 tetrahedral triangular elements and the GMRES system solver (Incomplete LU preconditioner, 0.01 drop tolerance) in COMSOL is chosen (achieving fast convergence of such a highly complex system) for the solution of the set of equations. The charge conservation in the anode and cathode electrodes is implemented using the weak form boundary PDEs incorporated in COMSOL. The kinetic Monte Carlo method is employed for the simulation of the reaction-diffusion micro-processes (as in Table 4.1) taking place on the catalytic surface. The 2D catalytic surface is represented by a 1800x400 sites micro-lattice (respective to the L_axW macro-lattice). The macro- and micro-scopic models interact at the anodic triple phase boundaries via BSS generation and also through the gaseous species partial pressures. MATLAB R2007a (MATLAB, 2007) is used as the interface between the macro- and micro-scopic models the codes of which are written in COMSOL scripts and in standard FORTRAN 90 (Ellis et al., 1994), respectively (Appendix A.2).

The selected framework parameters are tabulated in Table 4.4 along with with their corresponding sources. The electronic charge conductivities, σ_{el}^{A} and σ_{el}^{C}, can be found in COMSOL 4.2a Material Library (COMSOL, 2011), while the ionic charge conductivity is given by (Ferguson *et al.*, 1996):

$$\sigma_{io}^{YSZ} = 33.4 \cdot 10^{3} \exp\left(\frac{-10300}{T}\right) \quad (4.53)$$

Table 4.4: Multi-scale framework utilised parameters.

Parameter	***Symbol***	***Value***
Pt specific site density, mol m^{-2}	N_S	2.5407×10^{-5} (Yentekakis et al., 1994)
Sticking coefficient of O_2 on Pt	S_{O_2}	7.69×10^{-5} (Fragkopoulos et al., 2013)
Sticking coefficient of CO on Pt	S_{CO}	5.38×10^{-1} (Fragkopoulos et al., 2013)
O_2 desorption pre-exponential factor, s^{-1}	$k^{d}_{o,-1}$	2.4×10^{13} (Kaul et al., 1987)
O_2 desorption activation energy, J mol^{-1}	$E^{d}_{A,-1}$	243139 (Fragkopoulos et al., 2013)
CO desorption pre- exponential factor, s^{-1}	$k^{d}_{o,-2}$	6.5×10^{13} (Kaul et al., 1987)
CO desorption activation energy, J mol^{-1}	$E^{d}_{A,-2}$	99618 (Fragkopoulos et al., 2013)
CO and O_2 surface reaction pre-exp factor, s^{-1}	$k^{r}_{o,3}$	2.7×10^{6} (Kaul et al., 1987)
CO oxidation activation energy, J mol^{-1}	$E^{r}_{A,3}$	35186 (Fragkopoulos et al., 2013)

CO and BSS surface reaction rate constant, s^{-1}	k_4^r	5.73×10^{-3} (Fragkopoulos et al., 2013)
BSS desorption rate constant, s^{-1}	k_5^d	4.27×10^{-3} (Fragkopoulos et al., 2013)
BSS continuum diffusion coefficient, $m^2\ s^{-1}$	D_{BSS}	4×10^{-15} (Vayenas and Pitselis, 2001)
BSS diffusion rate constant (kMC), s^{-1}	k_{diff}^{BSS}	8.4×10^{-2} (using Eq. (4.22))
Cathodic charge transfer coefficient	α^C	0.5
Anodic charge transfer coefficient	α^A	0.5
Anode activation energy, $J\ mol^{-1}$	E_A^A	120000 (Fragkopoulos et al., 2013)
Cathode activation energy, $J\ mol^{-1}$	E_A^C	110000 (Fragkopoulos et al., 2013)
Cathode pre-exponential coefficient, $A\ m^{-2}$	γ_C	6.91×10^8 (Fragkopoulos et al., 2013)
Anode pre-exponential coefficient, $A\ m^{-2}$	$\gamma_{A,1}$	5.01×10^8 (Fragkopoulos et al., 2013)
Anode pre-exponential coefficient, $A\ m^{-2}$	$\gamma_{A,2}$	2.92×10^{11} (Fragkopoulos et al., 2013)
Anode pre-exponential coefficient, $A\ m^{-2}$	$\gamma_{A,3}$	3.42×10^4 (Fragkopoulos et al., 2013)
Electrolyte (YSZ) electrical conductivity, $\Omega^{-1}\ m^{-1}$	σ_{io}	as in Eq. (4.53)

Cathode (Au) electrical conductivity, Ω^{-1} m^{-1}	σ_{el}^{C}	as in COMSOL (2011)
Anode (Pt) electrical conductivity, Ω^{-1} m^{-1}	σ_{el}^{A}	as in COMSOL (2011)
O_{YSZ}^{2-} chemical potential, J mol^{-1}	$\mu_{O_{YSZ}^{2-}}$	-236.4×10^3 (Bessler et al., 2007a; Vogler et al., 2009)
CO standard chemical potential, J mol^{-1}	μ_{CO}^{o}	-137.3×10^3 (Perry et al., 1963)
CO_2 standard chemical potential, J mol^{-1}	$\mu_{CO_2}^{o}$	-394.4×10^3 (Perry et al., 1963)

4.8 Results and discussion

The operating conditions utilised in the multi-scale framework are listed in Table 4.5. The operating potential, Φ_{op}, is fixed to 700mV and is considered to be constant throughout the solid oxide single pellet. Also, the pellet is assumed as preheated at the system operating temperature. Comparison between the proposed multi-scale framework and our macroscopic model is carried out under atmospheric conditions, through investigating the transient and the steady state behaviours resulting from each modelling study. Furthermore, the multi-scale framework was exploited for the investigation of the effect of the gaseous species partial pressures and of the operating temperature on the CO_2 production rate.

Table 4.5: Multi-scale framework operating conditions.

Parameter	***Symbol***	***Value***
Operating (reference) pressure, atm	P_T	1
Faraday's constant, A s mol^{-1}	F	96485
Volumetric flowrate of gas mixture, m^3 s^{-1}	F_d	$2.5x10^{-6}$
Operating potential, V	Φ_{op}	0.7

4.8.1 Transient results

A comparison between the the CO_2 production rate dynamic profiles resulting from the macroscopic and the multi-scale frameworks at temperature T = 623.15K, for an inlet partial pressure of O_2, $P_{O_2}^{in} = 3.5\text{kPa}$, and different CO partial pressures is depicted in Figures 4.10a-c.

We can observe that although the dynamics of the models are similar, quantitative differences are exhibited in their steady state outputs. More specifically, the multi-scale framework results in greater CO_2 production rates with a difference of about 70% with the macroscopic output for the set of $P_{O_2}^{in} = 3.5\text{kPa}$ and $P_{CO}^{in} = 50\text{Pa}$ (Figure 4.10a). Increasing the CO partial pressure to $P_{CO}^{in} = 300\text{Pa}$ (Figure 4.10b) leads to a decreased difference of 20% and to almost identical responses in the case of $P_{CO}^{in} = 660\text{Pa}$ (Figure 4.10c). Consequently, the macroscopic model seems to underestimate the multi-scale

output for low CO partial pressures, while the systems tend to the same quantitative behaviour for relatively greater CO partial pressures.

Figure 4.10a-c shows that the macroscopic model simulation can accurately replace the multi-scale system one for high CO partial pressures, while the direct multi-scale simulations are necessary for accurate CO_2 production rate estimations at low CO partial pressures as expected, since high CO partial pressures result in CO high coverages and to subsequent reduced interactions with adsorbed O molecules.

The CO_2 production rate transients were further investigated using different sets of operating conditions. Figures 4.11a-c and 4.12a-c illustrate the CO_2 production rate differences between the two modelling frameworks for $P_{O_2}^{in} = 5.0\text{kPa}$ and $P_{O_2}^{in} = 6.5\text{kPa}$ operating O_2 partial pressures respectively. The dynamic trends of the models are again here similar with decreasing differences between the models' steady state outputs when the corresponding P_{CO}^{in} were increased.

Furthermore, in Figures 4.11a and 4.12a we can see that the models demonstrate the same transient behaviours even in cases where the curves of the CO_2 production rate initially reach a maximum point and subsequently change direction reaching a plateau in lower CO_2 production rate values. Moreover, as we can observe in Figures 4.10a-c, 4.11a-c and 4.12a-c, both modelling frameworks reach steady state at the same time for all utilised operating conditions as expected due to the low BSS diffusion probability compared with the adsorption/desorption ones (same as in Raimondeau and Vlachos (2002b) for low CO diffusion probability).

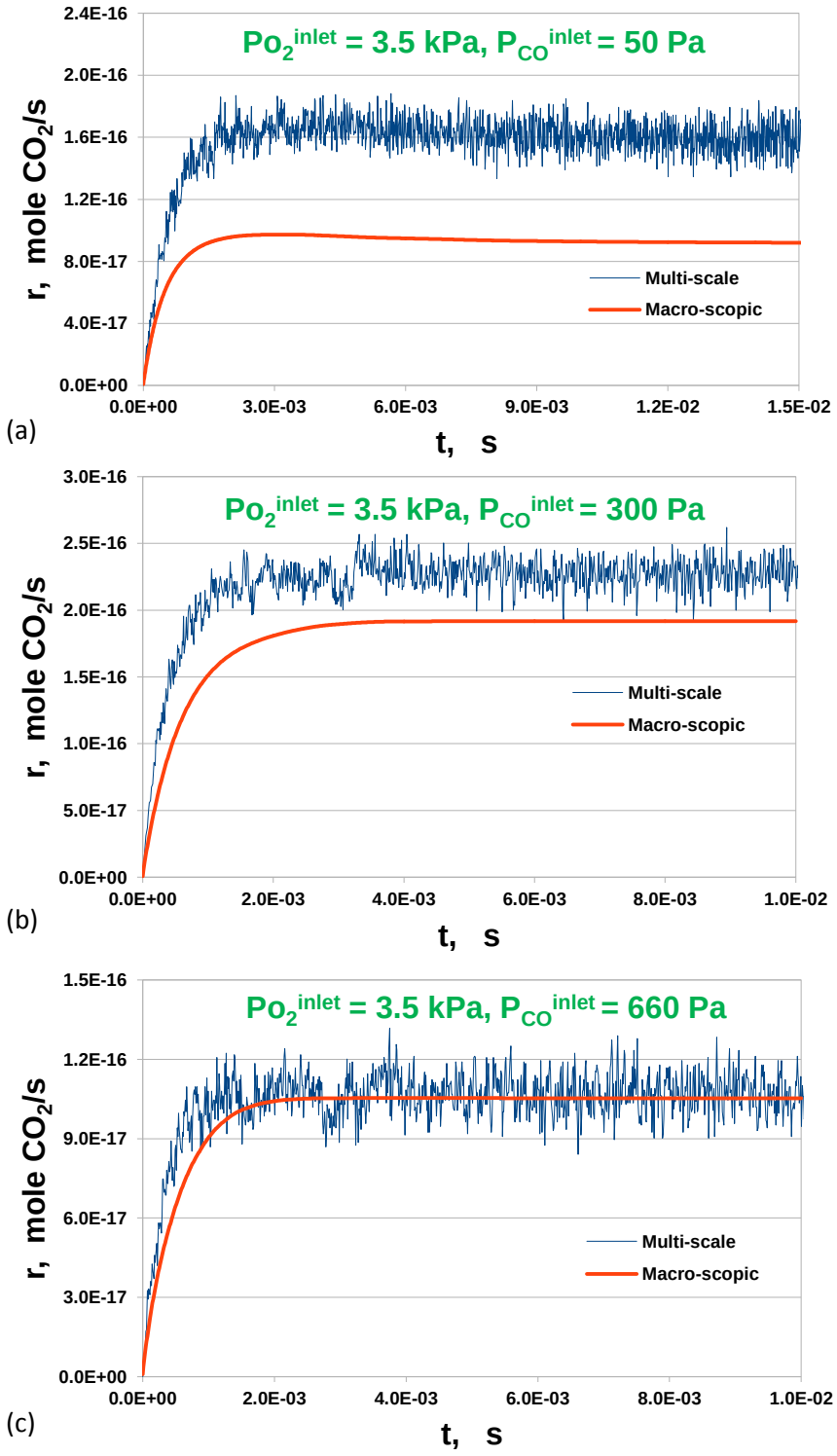


Figure 4.10: Transients of CO_2 production rate, at T = 350°C, for an inlet partial pressure of O_2, $P_{O_2}^{in}$ = 3.5kPa, and an inlet partial pressure of CO: (a) P_{CO}^{in} = 50Pa, (b) P_{CO}^{in} = 300Pa, (c) P_{CO}^{in} = 660Pa.

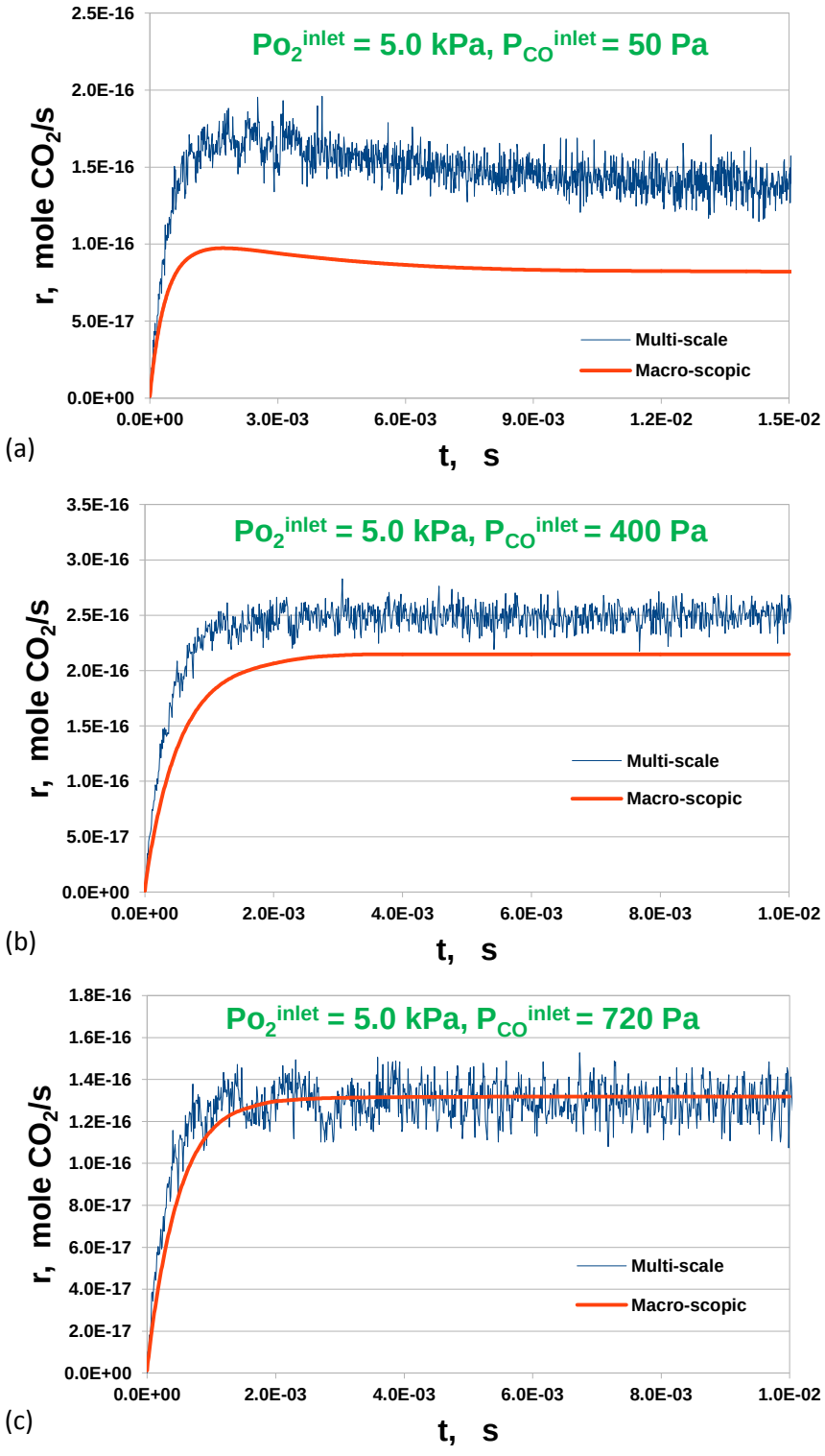


Figure 4.11: Transients of CO_2 production rate, at T = 350°C, for an inlet partial pressure of O_2, $P_{O_2}^{in}$ = 5.0kPa, and an inlet partial pressure of CO: (a) P_{CO}^{in} = 50Pa, (b) P_{CO}^{in} = 400Pa, (c) P_{CO}^{in} = 720Pa.

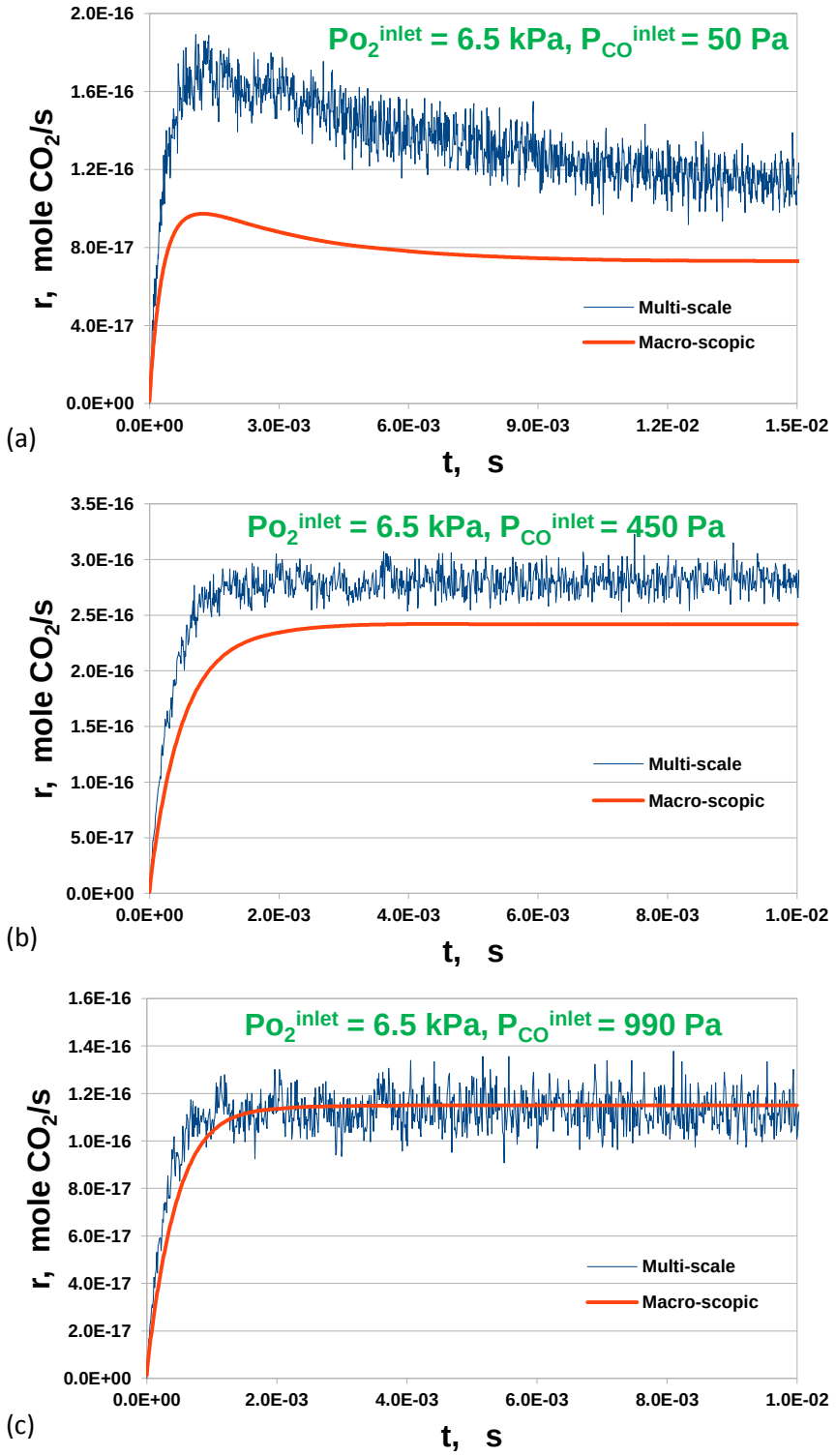


Figure 4.12: Transients of CO_2 production rate, at T = 350°C, for an inlet partial pressure of O_2, $P_{O_2}^{in}$ = 6.5kPa, and an inlet partial pressure of CO: (a) P_{CO}^{in} = 50Pa, (b) P_{CO}^{in} = 450Pa, (c) P_{CO}^{in} = 990Pa.

4.8.2 Steady state results

The steady state effect of inlet partial pressure of CO on CO_2 production rate for both macroscopic and multi-scale models, at temperature T = 623.15K and for inlet partial pressures of O_2, $P_{O_2}^{in} = 3.5\text{kPa}, 5\text{kPa} \,\&\, 6.5\text{kPa}$ is summarised in Figures 4.13a-c respectively. Steady state is reached through evaluating the CO_2 formation rate over sufficiently long time lengths reaching low statistical precision.

A typical "volcano-type" behaviour (Vayenas et al., 2001a) is observed in the CO_2 production rate for both frameworks. Greater differences are observed between the models' corresponding outputs for relatively low CO inlet partial pressures.

The produced CO_2 is predicted to reach a maximum value at $P_{O_2}^{in} / P_{CO}^{in} \approx 30$ through the multi-scale framework and at $P_{O_2}^{in} / P_{CO}^{in} \approx 20$ when the macroscopic model is utilised. Moreover, the macroscopic model predicts a lower maximum CO_2 production rate than the multi-scale system, demonstrating the importance of accurate simulations in such a system where maximum CO_2 production rate is desired.

For $P_{O_2}^{in} / P_{CO}^{in} < 30$, the differences between the models' steady state outputs are getting reduced when increasing the CO inlet partial pressure and for $P_{O_2}^{in} / P_{CO}^{in} < 7$ minor or no differences are observed. Reducing discrepancies between the mean field and the kMC rates for increasing CO partial pressures has been reported in the literature (Raimondeau and Vlachos, 2002b). Raising the CO partial pressures leads to increasing CO adsorption rates and to subsequent CO higher coverages. The higher the CO coverage the lower the interactions between adsorbed CO and O surface species.

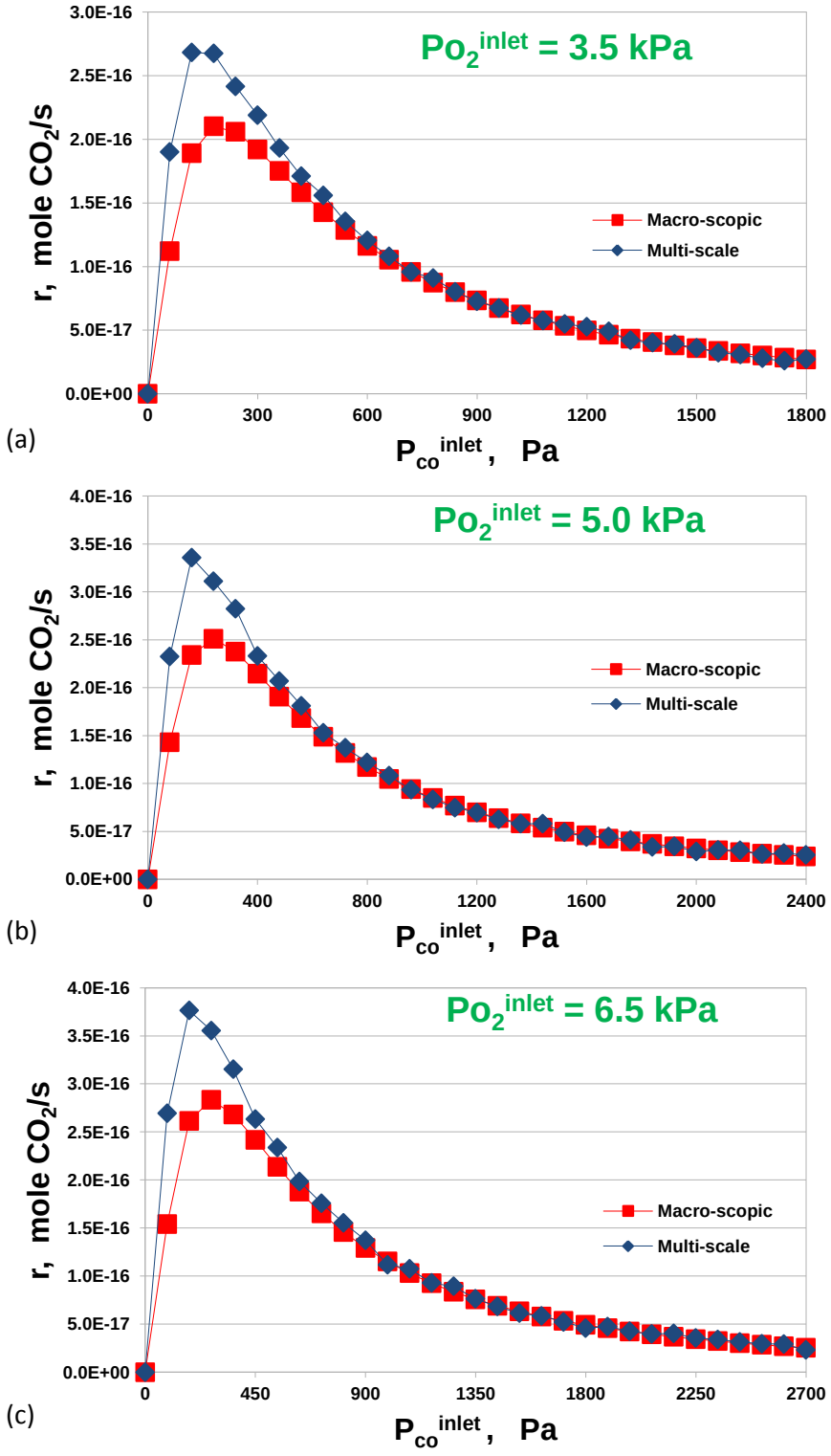


Figure 4.13: Steady state effect of inlet partial pressure of CO on CO_2 production rate, at T = 350°C and for an inlet partial pressure of O_2: (a) $P_{O_2}^{in}$ = 3.5kPa, (b) $P_{O_2}^{in}$ = 5.0kPa, (c) $P_{O_2}^{in}$ = 6.5kPa.

4.8.3 Parametric study

In this section parametric studies are performed, using the multi-scale framework, in order to investigate the effect of operating conditions on CO_2 production rate.

Effect of $P_{O_2}^{in}$. The effect of $P_{O_2}^{in}$ on CO_2 production rate at temperature T=623.15K and for increasing P_{CO}^{in} values is illustrated in Figure 4.14. A "volcano-type" behaviour is favoured in the CO_2 production rate. We can also observe that increasing the O_2 inlet partial pressure leads to greater CO_2 production rates and subsequently to better catalytic performance.

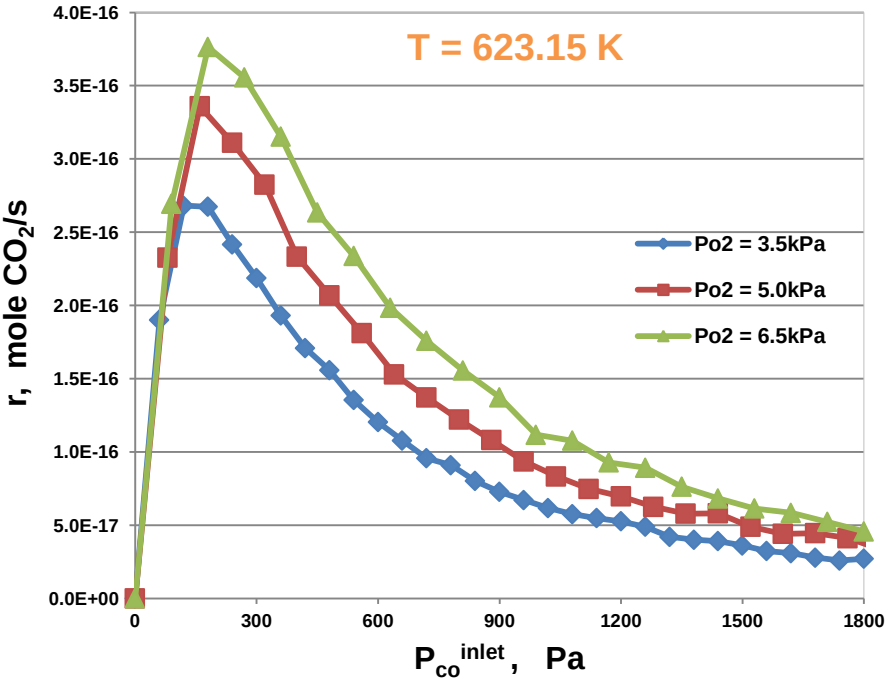


Figure 4.14: Steady state effect of inlet partial pressure of O_2 on CO_2 production rate, for an increasing inlet partial pressure of CO and at T = 350°C.

Effect of Temperature. The effect of temperature on CO_2 production rate for $P_{O_2}^{in} = 5\text{kPa}$ and for temperatures between 320°C and 380°C is depicted in Figure 4.15.

As we can see, small increases in the system operating temperature lead to noticeable increases of the electrochemically promoted CO_2 production rate as expected, since the surface reaction probability increases with increasing operating temperature. Increasing the system operating temperature causes a consequent increase in the CO desorption probability which as a result lifts the maximum production of CO_2 to the right requiring slightly greater P_{CO}^{in} values.

The effect of the system operating temperature on the electrochemically promoted CO_2 production rate is further investigated for temperature ranges between 300°C and 500°C (as the NEMCA effect in such systems is observed in that range of temperature values (Yentekakis and Vayenas, 1988; Vayenas et al., 1989; Vayenas and Bebelis, 1999)) and presented in Figure 4.16.

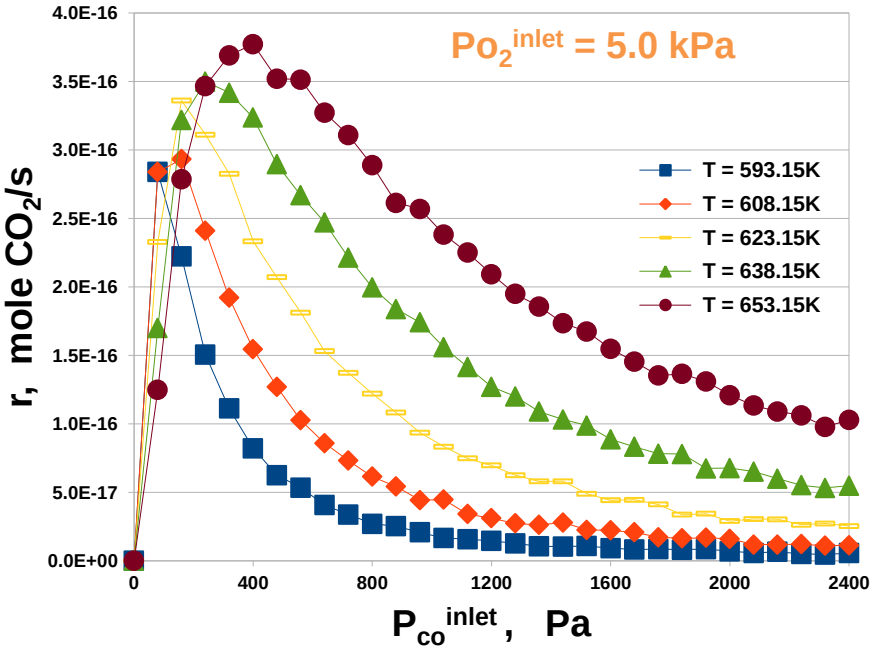


Figure 4.15: Steady state effect of temperature on CO_2 production rate, for a set inlet partial pressure of O_2 and an increasing inlet partial pressure of CO (T = 320-380°C).

As we can observe in Figure 4.16, temperature increases lead to significant increases of the catalytic performance. Also, "volcano-type" behaviour is observed for temperatures lower or equal to 400 °C, while "S-type" behaviour is favoured for temperatures greater or equal to 450°C.

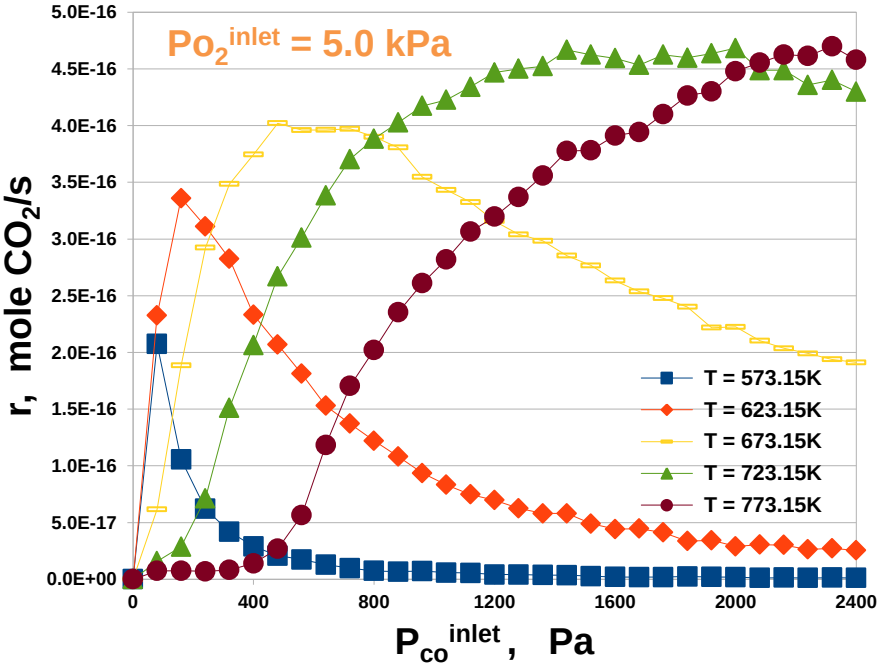


Figure 4.16: Steady state effect of temperature on CO_2 production rate, for a set inlet partial pressure of O_2 and an increasing inlet partial pressure of CO (T = 300-500°C).

4.9 Conclusions

The objective of this chapter was the formulation of a 3-dimensional, isothermal, dynamic solid oxide single pellet multi-scale framework to describe the chemical and electrochemical processes taking place in the system under potential application. The proposed framework is integrated by a 3-D macroscopic model which simulates the charge transport throughout the pellet as well as the electrochemical processes taking

place at the triple phase boundaries of the anode and the cathode, and a 2-D microscopic model which simulates the reaction-diffusion micro-processes taking place on the catalytic surface of the anode. COMSOL Multiphysics is utilised for the simultaneous simulation of the set of charge conservation PDEs employing the finite elements method, while an in-house developed lattice kMC model is employed to perform the microscopic simulations. The proposed framework allows the prediction of electronic and ionic potential curves, species coverage on the catalytic micro-lattice, transients of gas mixture concentration in the reactor, and ultimately it provides us with the CO_2 Faradaic and non-Faradaic production rates.

CO_2 production rate curves have been generated to compare the performance of the proposed multi-scale framework with a fully macroscopic one under atmospheric conditions. The models were found to exhibit similar dynamic trends for the sets of utilised conditions. However, differences have been observed between their steady state CO_2 production rates. Rising CO inlet partial pressures is leading to reducing differences between the models' steady state outputs and also to minor or almost no differences for $P_{O_2}^{in} / P_{CO}^{in} < 7$. Moreover, maximum catalytic performance is achieved at $P_{O_2}^{in} / P_{CO}^{in} \approx 30$ when utilising the multi-scale framework and at $P_{O_2}^{in} / P_{CO}^{in} \approx 20$ when using the macroscopic model. These observations suggest that the macroscopic model simulation can accurately replace the multi-scale system one for high CO partial pressures, while the use of the multi-scale system is necessary for accurate maximum CO_2 production rate estimations demonstared at low CO partial pressures.

The multi-scale framework was further exploited for temperature parametric studies. We have observed that temperature increases result in significant increases of the CO_2 production rate through increasing the surface reaction probability. Also, alterations in the system's behaviour have been observed. "Volcano-type" behaviour was favoured for temperatures lower or equal to 400 °C and "S-type" behaviour was observed for temperatures greater or equal to 450°C.

Chapter 5

Efficient lattice kMC simulations

5.1 Introduction

The aim of this chapter is the construction of an advanced and computationally efficient framework for open (microscopic) and closed circuit (multi-scale) lattice kMC simulations as an extension to the system described in Chapter 4. We believe that exploiting scaled-up multi-scale electrochemically promoted system simulations in conjuction with high-fidelity experiments will ultimately enhance their industrial potential through estimating reliable system parameters and through quantifying the effect of the circuit closure on such systems. The computational cost of such multi-scale simulations is substantial and it becomes even more significant as the size of the operating system increases. Consequently, the use of computationally efficient coarse-graining techniques becomes necessary.

Coarse-graining techniques account for the representation of a large spatial and temporal system by a reduced number of degrees of freedom (smaller spatial and temporal systems) leading to a decreased computational demand (Gorban, 2006). Several coarse-graining techniques, such as the iterative Boltzmann inversion method

(Schommers, 1983), the force-matching method (Ercolessi and Adams, 1994), the inverse Monte Carlo method (McGreevy and Pusztai, 1988), the coarse-grained Monte Carlo method (Katsoulakis et al., 2003) and 'equation-free' methods (Theodoropoulos, 2011) have been developed so far to investigate complex (bio-)molecular and multi-scale processes.

In this work, a coarse-graining methodology proposed by Kevrekidis and co-workers in the early 2010s (Gear et al., 2002; Makeev et al., 2002; Kevrekidis et al., 2003; Siettos et al., 2003), the gap-tooth method, is employed for the simulation of the reaction-diffusion phenomena taking place in the computationally demanding systems of interest. In such a coarse-graining method, detailed micro-scale simulations are used to provide information to macro-scale simulations, i.e. stochastic microscopic simulations can be used to calculate macroscopic states/properties through an averaging and/or filtering "restricting" process, which can consequently be utilised in continuum simulations (e.g. fluid dynamics).

The gap-tooth scheme (Gear et al., 2002, 2003) represents the computational domain with a subset of the total spatial domain (tooth) separated by spaces (gaps). Each tooth represents at least one 'coarse-grained' node of the macroscopic discretised grid. The main characteristic of this method is that a large 'coarse-grained' macroscopic system solution requiring long time scales can be efficiently obtained through calls to microscopic simulators (e.g. Monte Carlo, Lattice-Boltzman) using small spatial domains (teeth) and short time scales. The integrated multi-scale system communicates the "restricted" microscopic (stochastic) simulation output through continuum (macroscopic) methods through the use of coarse timesteppers, without the need to

construct the corresponding master equations. Hence, this methodology belongs to the class of 'equation-free' approaches (Theodoropoulos et al., 2000; Kevrekidis et al., 2003). The basic steps of this approach are "lifting", evolution, "restriction" and interpolation. Lifting accounts for the conversion of macroscopic state initial conditions, i.e. concentration, to consistent micro-scopic ones. Evolution accounts for the simulation of the microscopic model and subsequently for the computation of microscopic states. During restriction, the microscopic states are converted into average macroscopic information that are used to obtain the interfacial fluxes between the micro- and the macro-scopic domains. Interpolation obtains the solution of the macroscopic system at the gap regions.

This methodology has received increasing attention in complex multi-scale frameworks (Armaou et al., 2005; Majumber and Broadbelt, 2006; Hari and Theodoropoulos, 2009; Schaefer and Jansen, 2013). Armaou et al. (2005) have employed this 'equation-free' approach to reduce the computational requirement of the design calculations of a feedback controller that was used to stabilise a reaction-diffusion system based on the Lattice-Boltzmann method.

Majumber and Broadbelt (2006) have developed a multi-scale model to describe the fluid phase concentration variations in catalytic flow reactors. The proposed framework is integrated by a Finite Differences (FD) description of the fluid flow and the catalytic surface micro-kinetics are implemented through a kinetic Monte Carlo scheme. The FD and kMC solvers' communication is achieved through the use of the basic timestepping concepts, such as lifting (the macro-scopic concentration of fluid phase is translated into species coverage in the micro-scopic domain), evolving (the obtained micro-scopic

information are converted to tooth constrained interfacial molar fluxes between the kMC and FD simulations) and interpolation (calculation of the interfacial molar fluxes for the whole length of the reactor).

Schaefer and Jansen (2013) have presented an extension of the multi-scale model formulated by Majumber and Broadbelt (2006). They used sequential treatment of the gap regions (in contrast with Majumber and Broadbelt (2006) where all rates and concentrations are updated at the same time, requiring more kMC simulations during the first iterations due to the poor estimation of the rates and the concentrations close to the outlet of the reactor) and subsequently carried out reduced kMC simulations as a result to achieve better system performance.

Hari and Theodoropoulos (2009) have performed multi-scale simulations to describe the fluid phase concentration variations in a microreactor. In that study, kMC simulations were carried out at all the mesh points (nodes) of the FD discretisation not considering any gap regions and subsequently no interpolation was taken into account.

Gear and Kevrekidis (2002) and Gear et al. (2003) have proposed an alternative gap-tooth approach, which focuses on microscopic simulations of particles evolution on a lattice. The difference in this approach is that instead of interpolating the solution between macroscopic mesh points, the interpolation occurs at the microscopic level through lattice-to-lattice lateral interactions. Lattice-to-lattice lateral interactions, i.e. interactions between the neigbouring teeth (in the microscopic domain), are taken into account here through particles ingoing and outgoing exchange fluxes (Figure 5.1) which are obtained using intelligent interpolation rules.

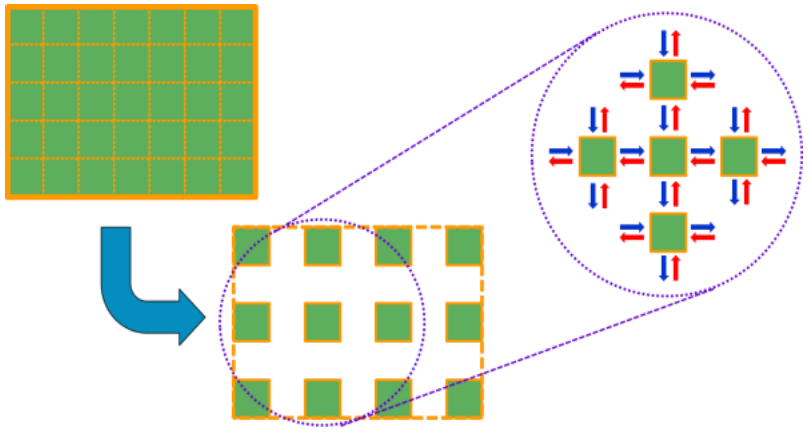


Figure 5.1: Schematic of the gap-tooth geometry.

In this chapter, we employ an in-house developed lattice gap-tooth framework for accurate and efficient microscopic and multi-scale simulations of heterogeneously catalysed systems through the implementation of the lattice kMC method. We first validate our gap-tooth/kMC framework considering no gaps between neighbouring teeth against a single lattice simulation (entire computational domain) and taking into account only the diffusion micro-process for a single species. The investigation of the effect of the gap size on the species diffusion in the gap-tooth scheme comes next. The framework is further exploited for open circuit (microscopic) and closed circuit (multi-scale) system simulations where the CO (electro-)oxidation is chosen as an illustrative example, as an extension of the work presented in the previous chapters.

5.2 The gap-tooth interpolation techniques

As mentioned earlier, the gap-tooth scheme is a well-known methodology employed to perform large macro-, micro- and multi-scale simulations with efficiency. The system in the gap-tooth framework is represented by two different spatial descriptions (a subgroup of the spatial domain known as teeth and the corresponding spaces between

the teeth referred to as gaps) that co-exist in the computational domain (Figure 5.1). Moreover, the size of the gap areas should ideally be much larger than the teeth sizes. The simulation of the system is directly executed at the teeth using boundary conditions obtained through interpolation in the gap areas (Gear and Kevrekidis, 2002). The lattice-to-lattice lateral interactions between two consecutive teeth are described through ingoing and outgoing exchange fluxes as illustrated in Figure 5.2.

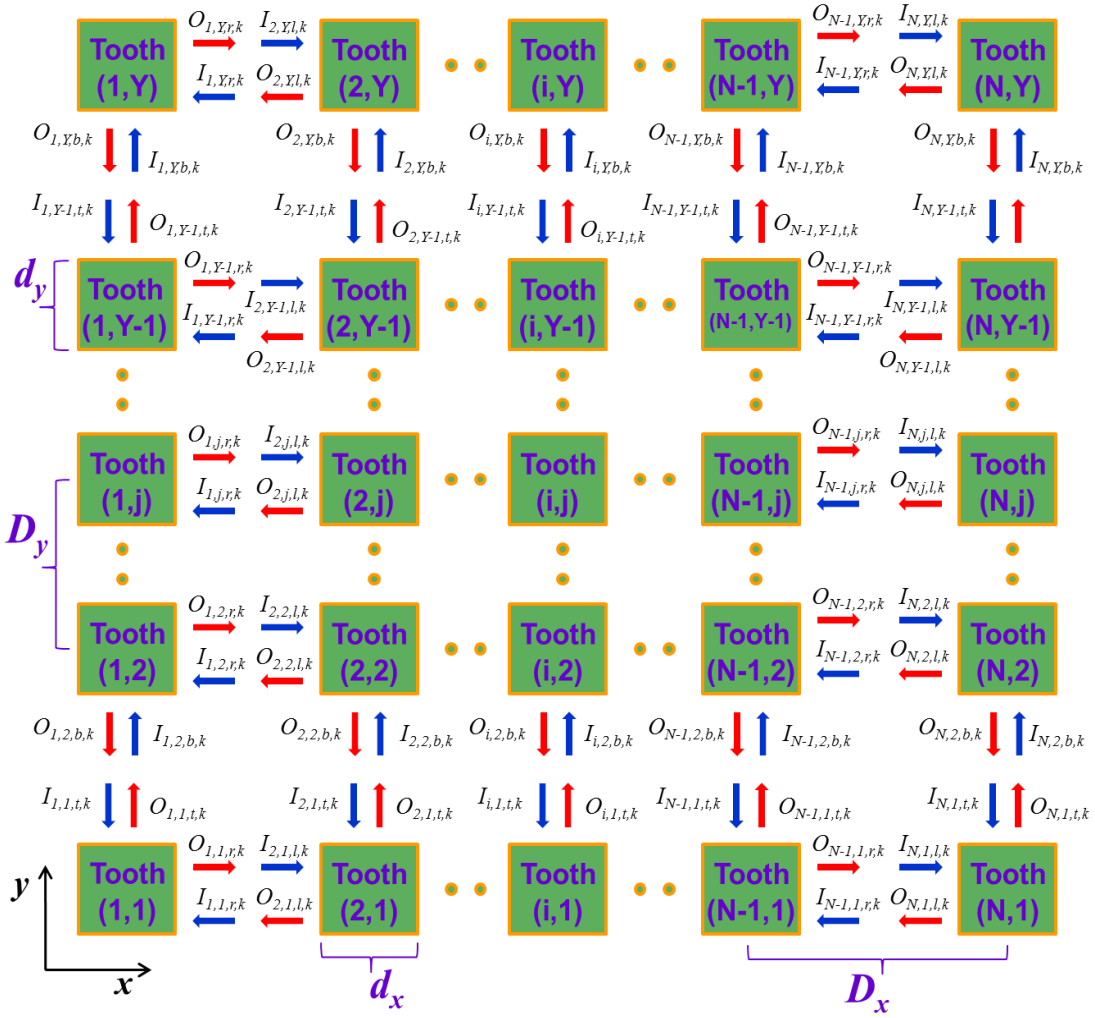


Figure 5.2: Schematic of the 2-D gap-tooth lattice-to-lattice lateral interactions.

Here, the lattice-to-lattice lateral interactions are related to the diffusion of species inwards to and outwards from each tooth. The outgoing fluxes of species are naturally

computed by the system simulation through calculating the number of particles that jump out of each micro-lattice (tooth), while the ingoing ones are generated through the use of intelligent interpolation rules. Obviously, the term "intelligent" here does not refer to the interpolation convention, which is rather straightforward, but actually to the complex underlying multi-scale concepts of lifting, restriction and evolution, which allow us to connect the microscopic lattice simulations to the macroscopic reality of the system.

Considering a 2-D gap-tooth scheme (as in Figure 5.2) and linear (1^{st} order) interpolation rules, the exchange fluxes can be expressed as (Gear and Kevrekidis, 2002; Gear et al., 2003):

$$I_{i,j,l,k} = \alpha_x \cdot O_{i-1,j,r,k} + (1-\alpha_x) \cdot O_{i,j,r,k} \tag{5.1}$$

$$I_{i,j,r,k} = \alpha_x \cdot O_{i+1,j,l,k} + (1-\alpha_x) \cdot O_{i,j,l,k} \tag{5.2}$$

$$I_{i,j,t,k} = \alpha_y \cdot O_{i,j+1,b,k} + (1-\alpha_y) \cdot O_{i,j,b,k} \tag{5.3}$$

$$I_{i,j,b,k} = \alpha_y \cdot O_{i,j-1,t,k} + (1-\alpha_y) \cdot O_{i,j,t,k} \tag{5.4}$$

where I and O are the ingoing and outgoing species respectively, i is the number of each tooth in the x-direction, j is the number of each tooth in the y-direction, k is the type of species, r and l are the right and left sides of each tooth respectively, t and b are the top and bottom sides of each tooth respectively and $a_{x/y}$ is the interpolation coefficient in the x/y-direction.

The interpolation coefficient, $a_{x/y}$, depends only on the gap-tooth geometry and is given by:

$$\alpha_{x/y} = \frac{d_{x/y}}{D_{x/y}} = \frac{d_{x/y}}{d_{x/y} + gap_{x/y}} \tag{5.5}$$

where $d_{x/y}$ is the length of each tooth in the x/y-direction, $D_{x/y}$ is the distance between the centres of two consecutive teeth in the x/y-direction and $gap_{x/y}$ is the corresponding space between the teeth in the x/y-direction (see Figure 5.2).

This 1st order interpolation is equivalent to directing $a_{x/y}$ of the particles outgoing from tooth (*i,j)* as influx to a consecutive tooth *(i±1,j) or (i,j±1)* and re-direct the remaining *(1-*$a_{x/y}$*)* particles back to tooth (*i,j).*

The ingoing fluxes of species can also be calculated using quadratic (2nd order) interpolation expressions as (Gear and Kevrekidis, 2002; Gear et al., 2003):

$$I_{i,j,l,k} = \frac{\alpha_x\left(1+\alpha_x\right)}{2}\cdot O_{i-1,j,r,k} + (1-\alpha_x^2)\cdot O_{i,j,r,k} - \frac{\alpha_x\left(1-\alpha_x\right)}{2}\cdot O_{i+1,j,r,k} \tag{5.6}$$

$$I_{i,j,r,k} = \frac{\alpha_x\left(1+\alpha_x\right)}{2}\cdot O_{i+1,j,l,k} + (1-\alpha_x^2)\cdot O_{i,j,l,k} - \frac{\alpha_x\left(1-\alpha_x\right)}{2}\cdot O_{i-1,j,l,k} \tag{5.7}$$

$$I_{i,j,t,k} = \frac{\alpha_y\left(1+\alpha_y\right)}{2}\cdot O_{i,j+1,b,k} + (1-\alpha_y^2)\cdot O_{i,j,b,k} - \frac{\alpha_y\left(1-\alpha_y\right)}{2}\cdot O_{i,j-1,b,k} \tag{5.8}$$

$$I_{i,j,b,k} = \frac{\alpha_y\left(1+\alpha_y\right)}{2}\cdot O_{i,j-1,t,k} + (1-\alpha_y^2)\cdot O_{i,j,t,k} - \frac{\alpha_y\left(1-\alpha_y\right)}{2}\cdot O_{i,j+1,t,k} \tag{5.9}$$

This 2nd order interpolation, i.e. in x-direction, is equivalent to directing the outgoing particles from the right side of tooth (*i,j),* $O_{i,j,r,k}$, to the following fractions of inputs:

- $a_x(1+a_x)/2$ to the left side of tooth (i,j+1), $I_{i,j+1,l,k}$
- $(1-a_x^2)$ to the left side of tooth (i,j), $I_{i,j,l,k}$
- $-a_x(1-a_x)/2$ to the left side of tooth (i,j+1), $I_{i,j-1,l,k}$

The quadratic interpolation is in general more adequate for boundary conditions (Gear and Kevrekidis, 2002), since diffusion depends on the derivative of the flux (and not on the flux itself). Nevertheless, we have found that it can not been used in systems with discontinuous external provision of particles or in systems with zero coverages of particles at any of the teeth.

More specifically, when the external flux of species is not continuous, the species are accumulated at the middle teeth leading to zero species coverage at the edge teeth (see Figures A.1 and A.2 in Appendix A.3.4). Consequently, the use of the 3rd fraction of the interpolation expression (negative) is improper due to the unavailability of particles that need to be subtracted from the neighbouring teeth.

A possible solution to this is to use linear interpolation expressions when no species (anti-particles) are available to be subtracted. For this reason, only 1st order interpolation rules will be taken into account in our case studies.

5.3 Case studies

5.3.1 The diffusion micro-process

In order to validate the performance of the gap-tooth framework, we have first taken into account an 1-D 'no'gap-tooth scheme, i.e. considering no gaps between neighbouring teeth, which was compared against a single lattice simulation using a spatial version of the kMC method and taking into consideration only the diffusion micro-process (as in Eq. (5.10)) of a single species X.

$$X \cdot S + \cdot S \xrightarrow{k_{diff}} \cdot S + X \cdot S \tag{5.10}$$

Such a validated 'no'gap-tooth scheme can be extensively used to perform single lattice kMC simulations with computational efficiency employing parallelisation techniques, such as the message passing interface (MPI).

Considering no gap (a_x=1) between the teeth, the fluxes of ingoing species can be simplified to:

$$I_{i,j,l,k} = O_{i-1,j,r,k} \tag{5.11}$$

$$I_{i,j,r,k} = O_{i+1,j,l,k} \tag{5.12}$$

The schematic representation of a single lattice consisting of 1100x100 sites and of an equivalent 1-D 'no'gap-tooth framework comprising of 11 teeth in the x-direction of 100x100 sites each is illustrated in Figure 5.3.

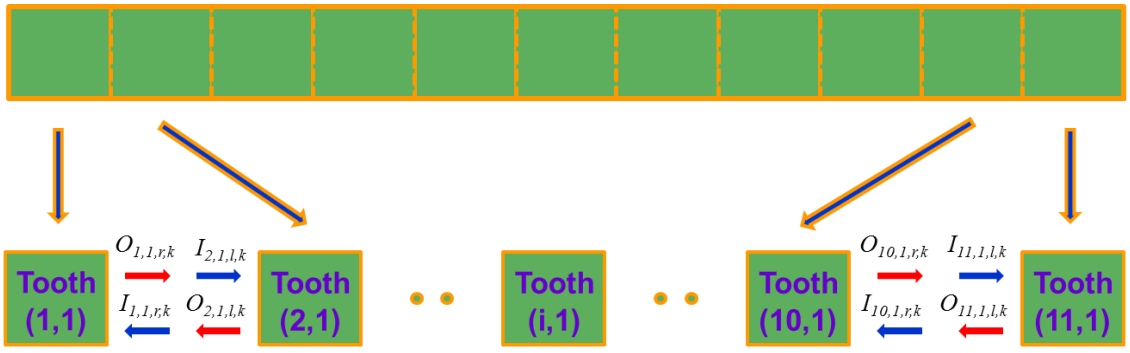


Figure 5.3: Schematic of an 1-D NoGap-Tooth representation of the single lattice.

An external flux of 50 species per 10^{-4} s is introduced at the left boundary of the single lattice and at the left boundary of tooth (1,1) in the 'no'gap-tooth framework respectively. Such a flux is sufficient to keep the effective provision of species constant, without entirely blocking the sites positioned at the respective boundaries, and consequently, leads to identical total particle conservation in both systems. The species

provision (external flux) is stopped at time t = 10^{-2} s in order to let species diffuse without the influence of external driving forces, since continuous provision of species at the left side boundaries would result in an effective blocking of the boundary sites and hence to an actual external flux lower than the specified one. Furthermore, a (high) diffusion probability, k_{diff} = 10^{6} s^{-1} (equivalent to a macroscopic diffusion coefficient of the value of 6.5 x 10^{-10} m^2/s), is selected to enable fast diffusion of species in order to illustrate the effect of lattice-to-lattice lateral interactions between the teeth. In addition, the interpolation of species between the consecutive teeth was chosen to take place at a (short) gap-tooth time reporting horizon ($t_{rep}^{gap-tooth}$) of 10^{-6} s allowing for very regular species exchanges.

An investigation of the effect of the ingoing species distribution on the species coverage on each tooth was undertaken, utilising different distribution methodologies and validating the 'no'gap-tooth simulation against the entire lattice one.

The following distribution methods were utilised:

- Random distribution of ingoing species over the whole micro-lattice (tooth) area (random distribution).
- Random distribution of ingoing species but only at the boundaries of each tooth (boundary distribution).
- Random distribution of ingoing species but within thin 'zones' around the boundaries of each tooth (zone distribution).

A comparison between the cumulative coverage of the diffusing species (a cumulative coverage of the diffusing species on a tooth (i,1) represents the total coverage of the species on all the teeth up to and including tooth (i,1)) computed by the single lattice simulation and that computed by the 'no'gap-tooth simulation (using

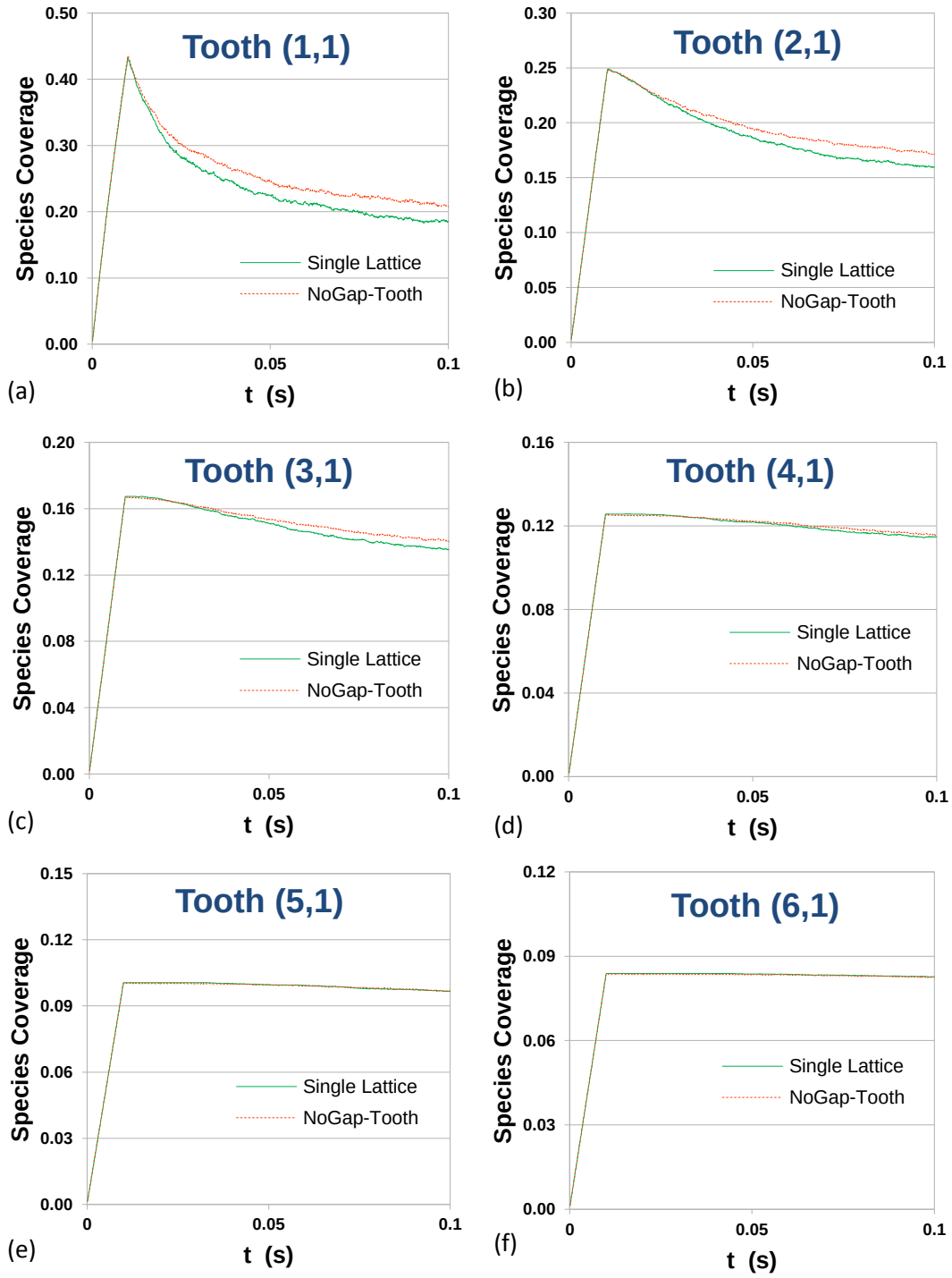
boundary, random and zone distribution of ingoing species techniques) is depicted in Figures 5.4 to 5.7.

As we can see in Figure 5.4, the ‘no’gap-tooth simulator can accurately capture the short term dynamics of the diffusing species when using boundary distribution. However, small differences between the single lattice and the ‘no’gap-tooth resulting coverage profiles are observed at teeth (1,1) to (4,1) at the long term dynamics. More specifically, the species coverage computed by the ‘no’gap-tooth simulation is slightly greater than the one computed by the single lattice simulation. This suggests that the effective diffusion of species in the ‘no’gap-tooth simulation is slower than in the single lattice simulation. Furthermore, no differences are exhibited at all the teeth greater than tooth (5,1).

In the case of random distribution (Figure 5.5), the simulations exhibit greater differences than the ones observed in the ‘boundary’ distribution. More specifically, the species coverage profiles computed by the ‘no’gap-tooth simulation are lower than the ones computed by the single lattice simulation. This suggests that the species diffusion in the ‘no’gap-tooth simulation is faster than the single lattice one. Furthermore, reduced differences are observed at the end teeth (8,1) to (10,1) as expected due to particles conservation in the system.

In the case of (1-5) zone distribution (distribution of species within 5 columns of sites adjacent to the boundaries), the differences between the single lattice and the ‘no’gap-tooth coverage profiles are smaller than the boundary distribution ones (Figure 5.6). However, the diffusion is again here slower than the one in single lattice simulation.

Finally, as we can see in Figure 5.7, when we distribute the ingoing species in a (1-10) zone around the boundaries leads to identical behaviour of the single lattice and the 'no' gap-tooth simulations in both short and long term dynamics.



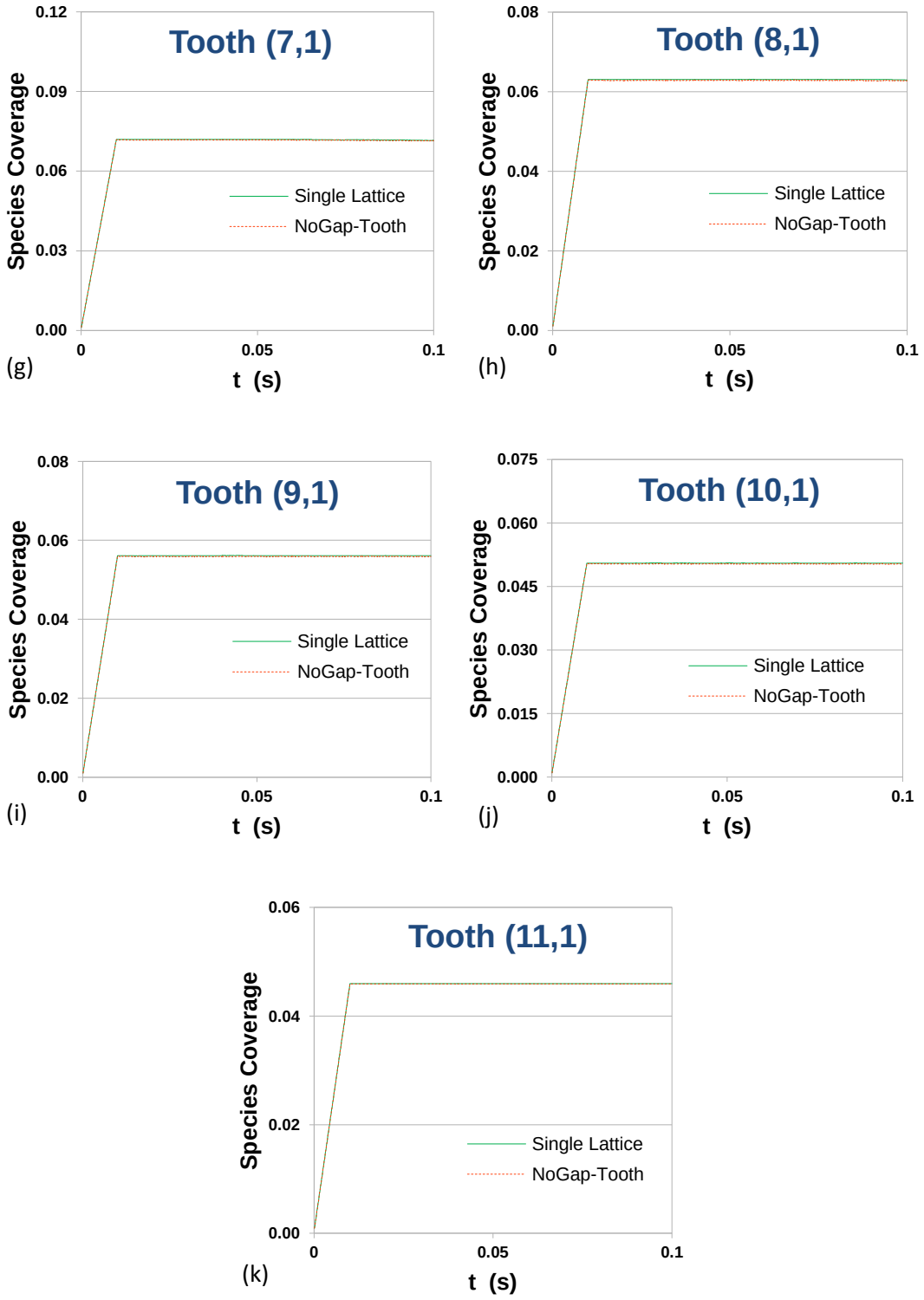
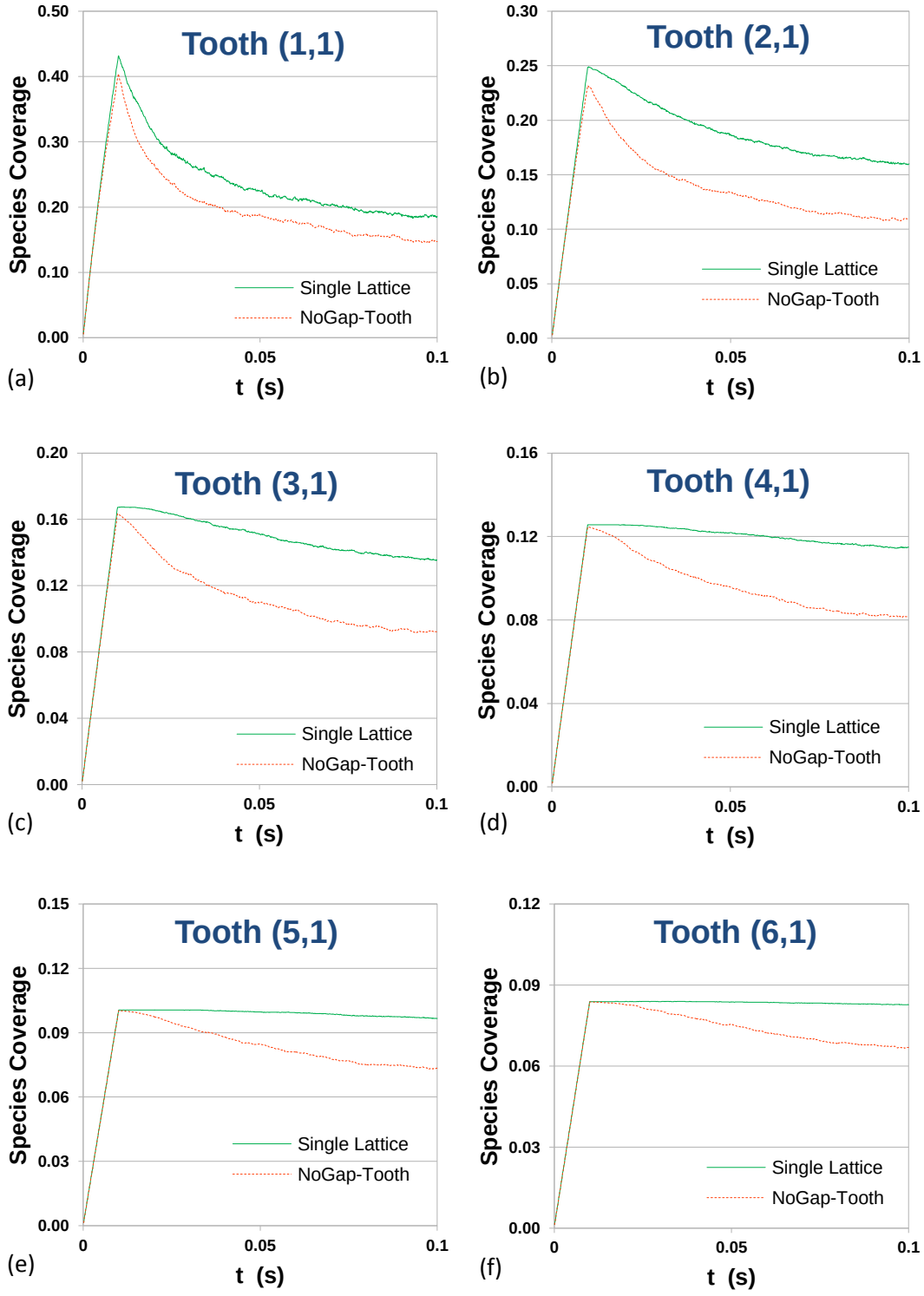


Figure 5.4: Diffusing species cumulative coverage profiles comparison between a single lattice (green solid lines) and a NoGap-Tooth (red dashed lines) framework, using 'boundary' distribution of ingoing species.


Tooth (1,1)
Species Coverage
0.50
0.40
0.30
0.20
0.10
0.00
0
0.05
0.1
t (s)
Single Lattice
NoGap-Tooth
(a)
Tooth (2,1)
Species Coverage
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
0.05
0.1
t (s)
Single Lattice
NoGap-Tooth
(b)
Tooth (3,1)
Species Coverage
0.20
0.16
0.12
0.08
0.04
0.00
0
0.05
0.1
t (s)
Single Lattice
NoGap-Tooth
(c)
Tooth (4,1)
Species Coverage
0.16
0.12
0.08
0.04
0.00
0
0.05
0.1
t (s)
Single Lattice
NoGap-Tooth
(d)
Tooth (5,1)
Species Coverage
0.15
0.12
0.09
0.06
0.03
0.00
0
0.05
0.1
t (s)
Single Lattice
NoGap-Tooth
(e)
Tooth (6,1)
Species Coverage
0.12
0.09
0.06
0.03
0.00
0
0.05
0.1
t (s)
Single Lattice
NoGap-Tooth
(f)

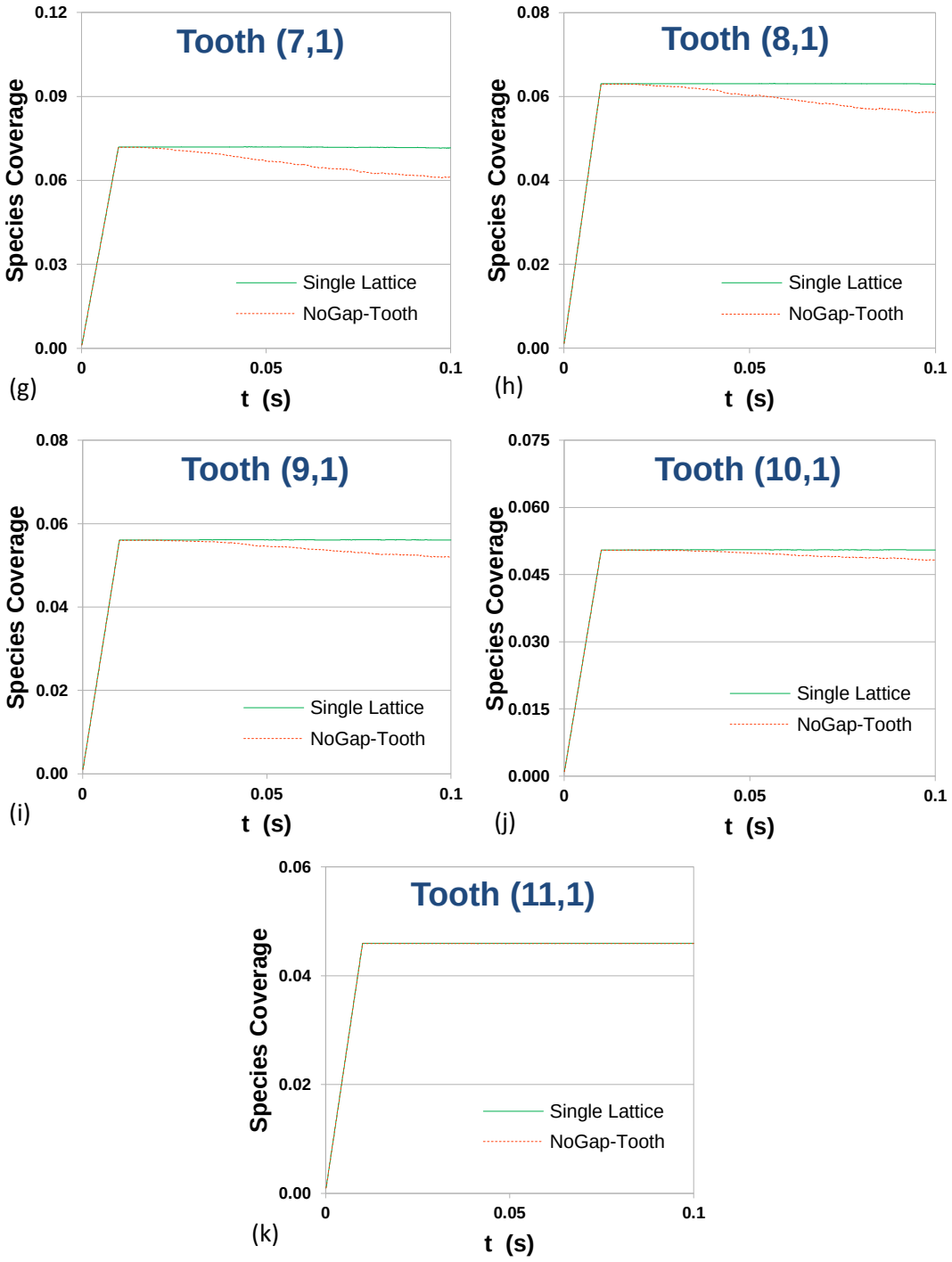
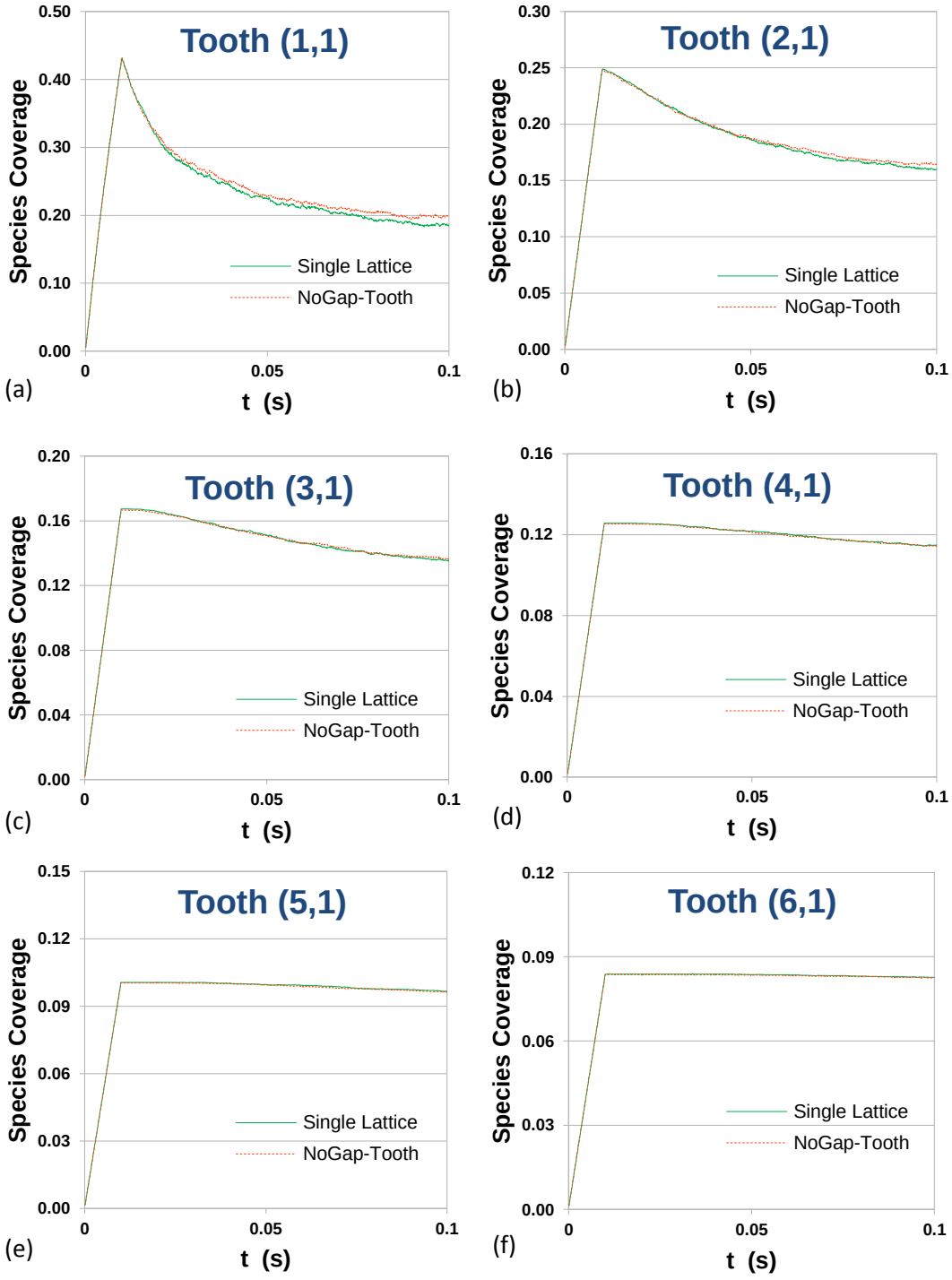


Figure 5.5: Diffusing species cumulative coverage profiles comparison between a single lattice (green solid lines) and a NoGap-Tooth (red dashed lines) framework, using 'random' distribution of ingoing species.


Tooth (1,1)
Species Coverage
0.50
0.40
0.30
0.20
0.10
0.00
0
0.05
0.1
t (s)
Single Lattice
NoGap-Tooth
(a)
Tooth (2,1)
Species Coverage
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
0.05
0.1
t (s)
Single Lattice
NoGap-Tooth
(b)
Tooth (3,1)
Species Coverage
0.20
0.16
0.12
0.08
0.04
0.00
0
0.05
0.1
t (s)
Single Lattice
NoGap-Tooth
(c)
Tooth (4,1)
Species Coverage
0.16
0.12
0.08
0.04
0.00
0
0.05
0.1
t (s)
Single Lattice
NoGap-Tooth
(d)
Tooth (5,1)
Species Coverage
0.15
0.12
0.09
0.06
0.03
0.00
0
0.05
0.1
t (s)
Single Lattice
NoGap-Tooth
(e)
Tooth (6,1)
Species Coverage
0.12
0.09
0.06
0.03
0.00
0
0.05
0.1
t (s)
Single Lattice
NoGap-Tooth
(f)

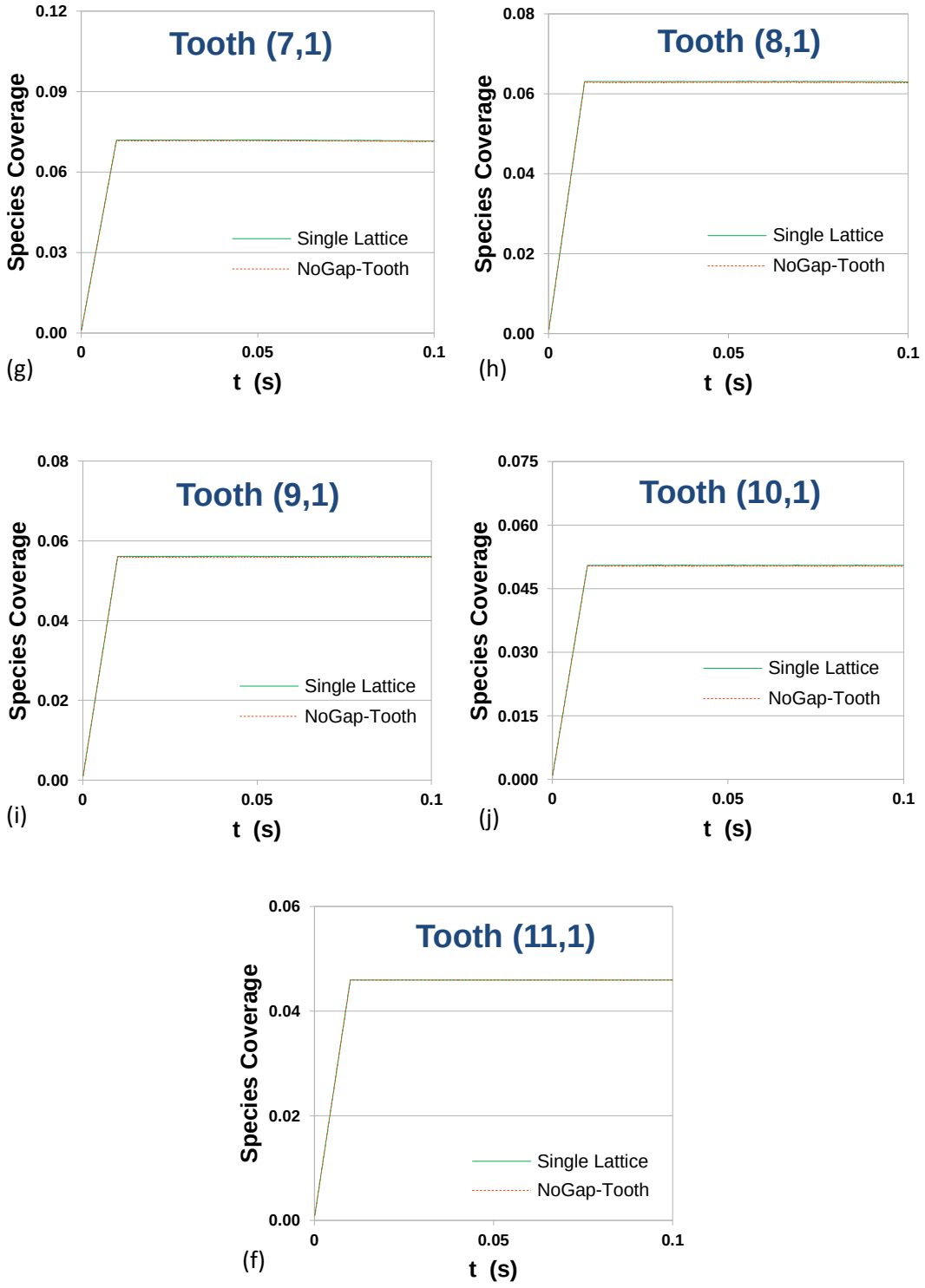
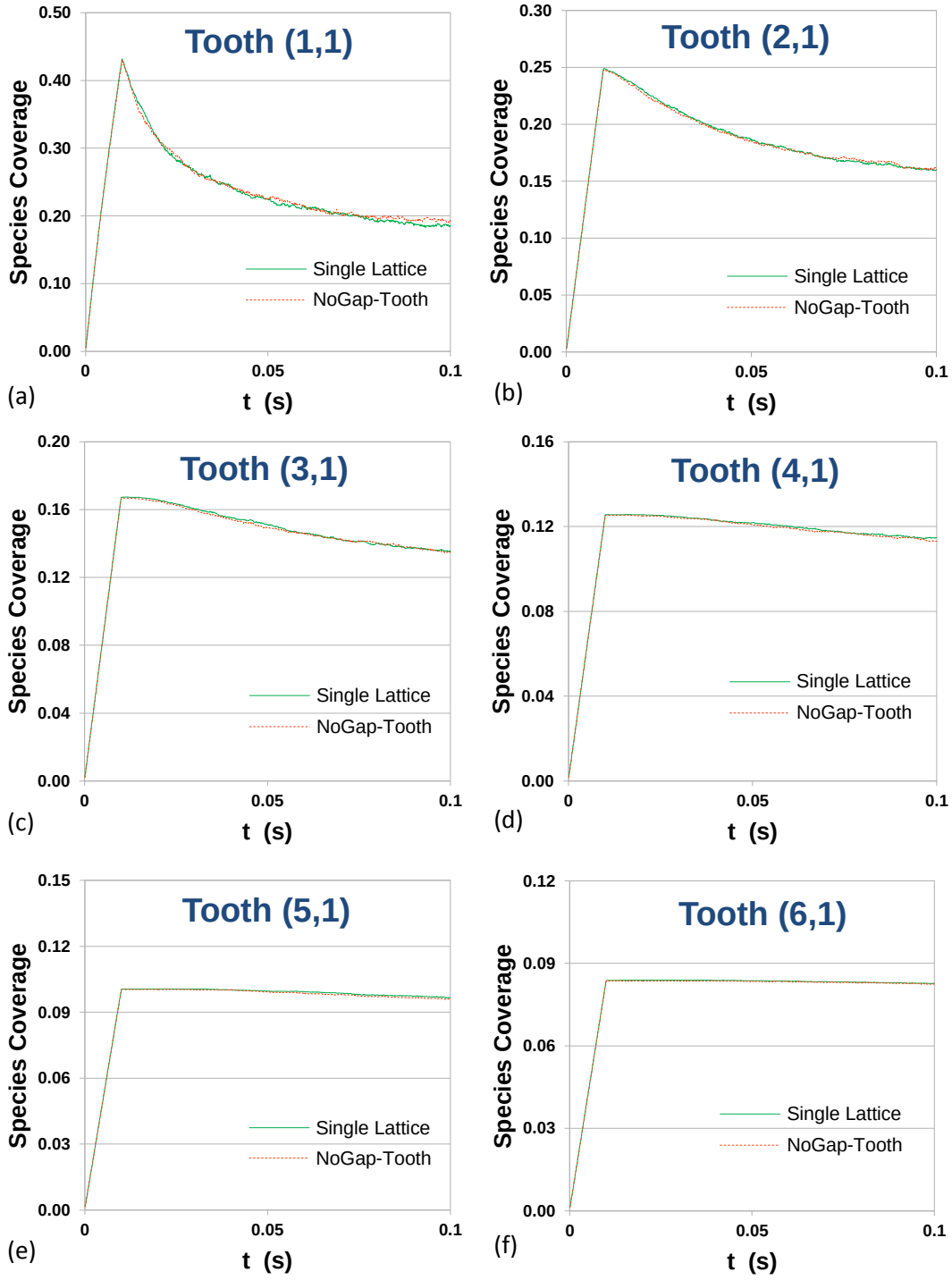


Figure 5.6: Diffusing species cumulative coverage profiles comparison between a single lattice (green solid lines) and a NoGap-Tooth (red dashed lines) framework, using (1-5) 'zone' distribution of ingoing species.


Tooth (1,1)
Species Coverage
0.50
0.40
0.30
0.20
0.10
0.00
0
0.05
0.1
t (s)
Single Lattice
NoGap-Tooth
(a)
Tooth (2,1)
Species Coverage
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
0.05
0.1
t (s)
Single Lattice
NoGap-Tooth
(b)
Tooth (3,1)
Species Coverage
0.20
0.16
0.12
0.08
0.04
0.00
0
0.05
0.1
t (s)
Single Lattice
NoGap-Tooth
(c)
Tooth (4,1)
Species Coverage
0.16
0.12
0.08
0.04
0.00
0
0.05
0.1
t (s)
Single Lattice
NoGap-Tooth
(d)
Tooth (5,1)
Species Coverage
0.15
0.12
0.09
0.06
0.03
0.00
0
0.05
0.1
t (s)
Single Lattice
NoGap-Tooth
(e)
Tooth (6,1)
Species Coverage
0.12
0.09
0.06
0.03
0.00
0
0.05
0.1
t (s)
Single Lattice
NoGap-Tooth
(f)

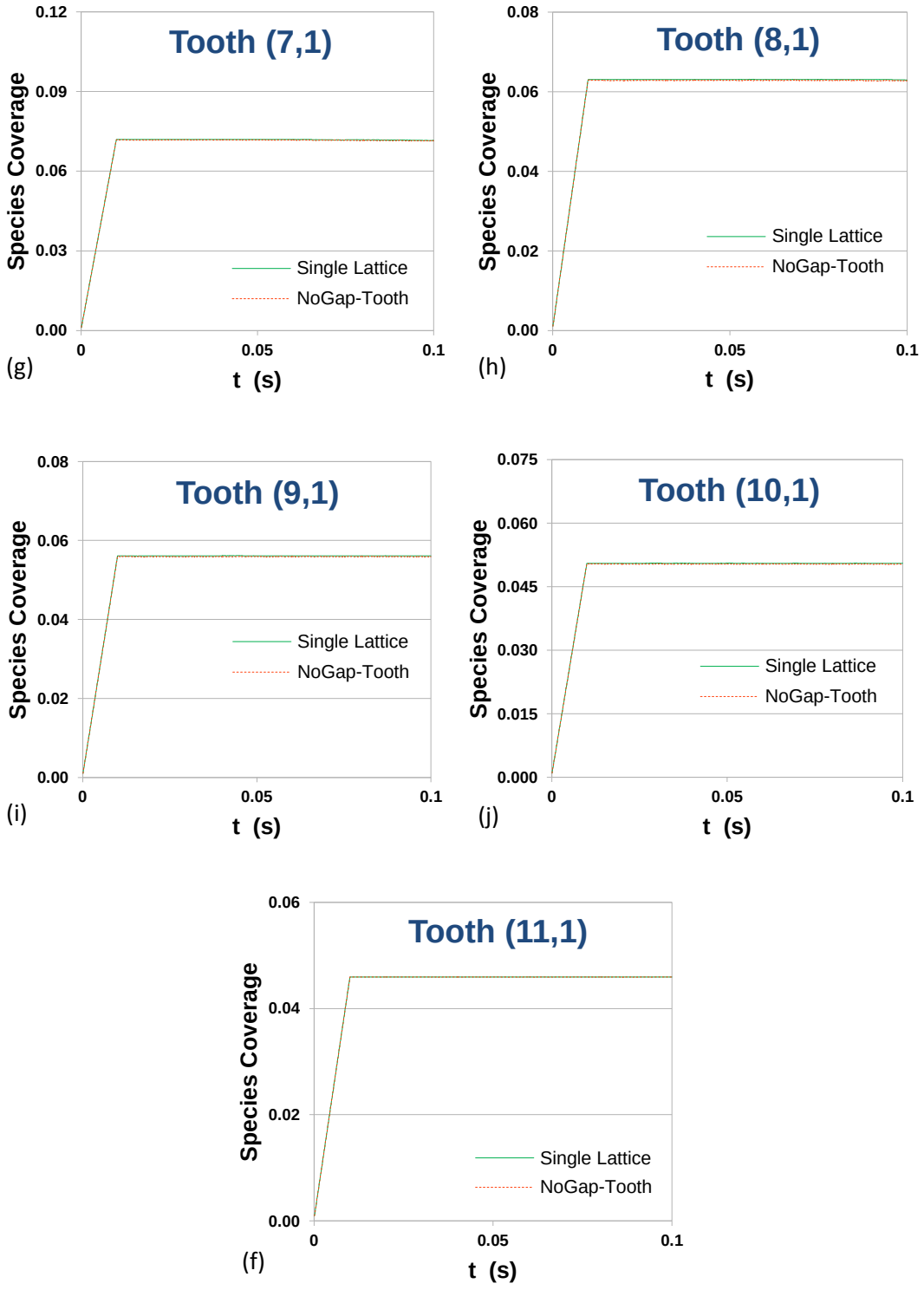


Figure 5.7: Diffusing species cumulative coverage profiles comparison between a single lattice (green solid lines) and a NoGap-Tooth (red dashed lines) framework, using (1-10) 'zone' distribution of ingoing species.

Having found that the single lattice dynamics can be sufficiently captured using the (1-10) zone distribution 'no'gap-tooth scheme, we have extended our study to analyse the effect of the gap/tooth dimensions on the species evolution in a gap-tooth framework. The gap-tooth scheme is presented in Figure 5.8. The single lattice -again here- consists of 1100x100 sites and is represented by 5 teeth consisting of 100x100 sites each (d_x = 100 sites) and a gap of 150 sites (D_x = 250 sites) between neighbouring teeth. The same operating conditions (external flux of 50 species per 10^{-4} s, $t_{rep}^{gap-tooth}$ = 10^{-6} s, k_{diff} = 10^{6} s^{-1}) as before are utilised.

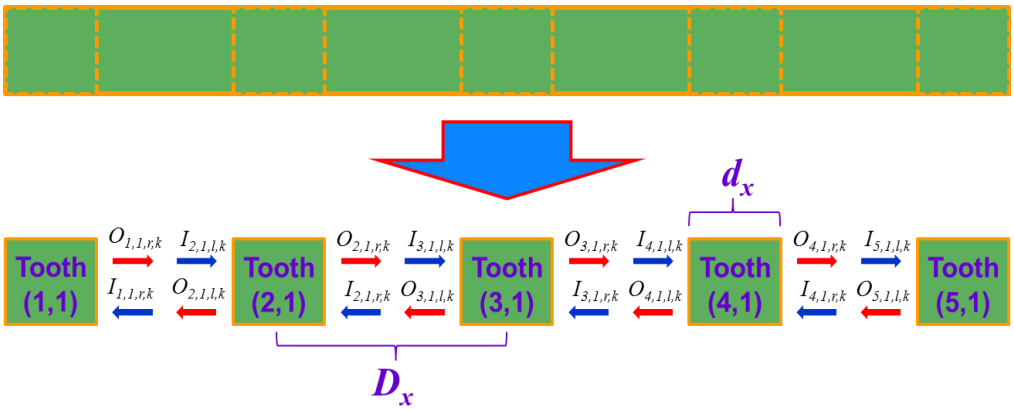


Figure 5.8: Schematic of an 1-D gap-tooth representation of the catalytic lattice.

In order to analyse the effect of the utilised gap on the particles evolution in the gap-tooth framework, we have carried out a comparison between the gap-tooth simulation -using both random and (1-10) zone distribution techniques- and the single lattice simulation for several sets of gap/tooth dimensions. In Figure 5.9, a comparison of the diffusing species cumulative coverage computed by the single lattice and that computed by the gap-tooth scheme using random distribution is depicted. As we can see the former is well represented by the gap-tooth simulation with differences in the middle teeth.

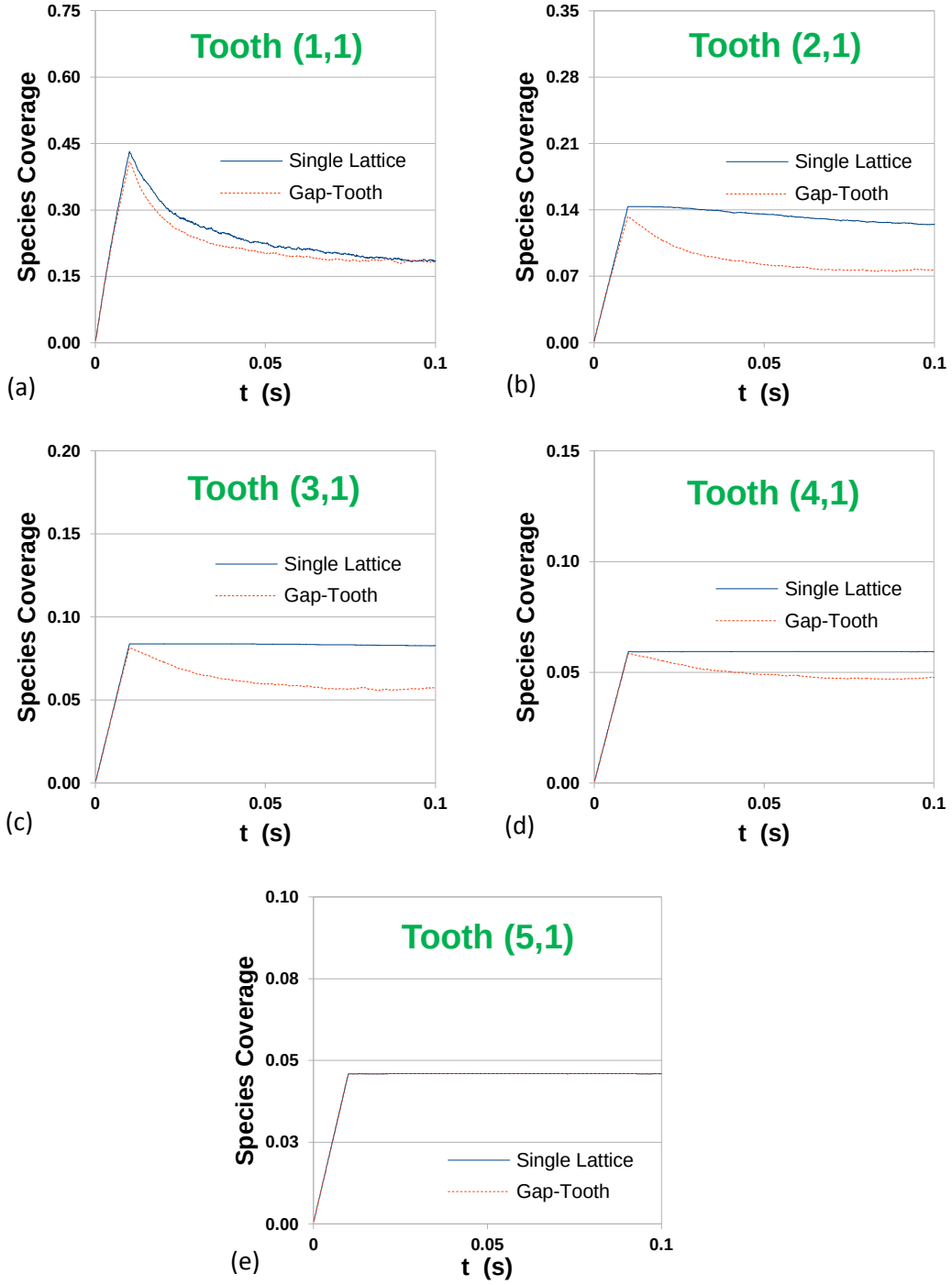


Figure 5.9: Diffusing species cumulative coverage profiles comparison between a single lattice (blue solid lines) and a Gap-Tooth (red dashed lines) framework, using 'random' distribution of ingoing species with gaps and teeth of 150 and 100 particles respectively.

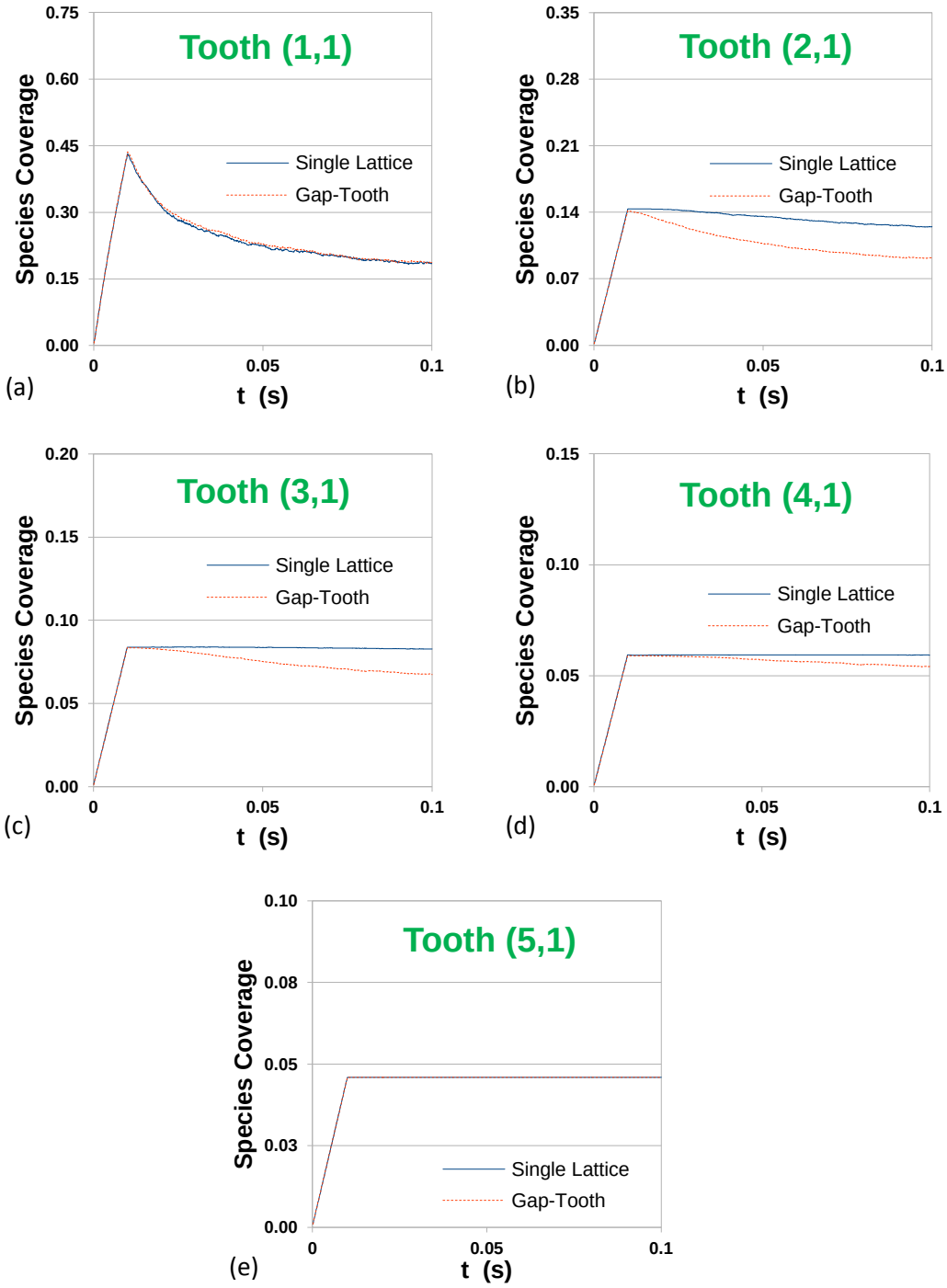


Figure 5.10: Diffusing species cumulative coverage profiles comparison between a single lattice (blue solid lines) and a Gap-Tooth (red dashed lines) framework, using (1-10) 'zone' distribution of ingoing species with gaps and teeth of 150 and 100 particles respectively.

As can been seen in Figure 5.10, using the (1-10) zone distribution we get better agreement with the single lattice simulation. The comparison between the gap-tooth simulation using random and that using zone distribution is illustrated in Figure 5.11. Noticeable differences can be observed between the two gap-tooth distribution techniques.

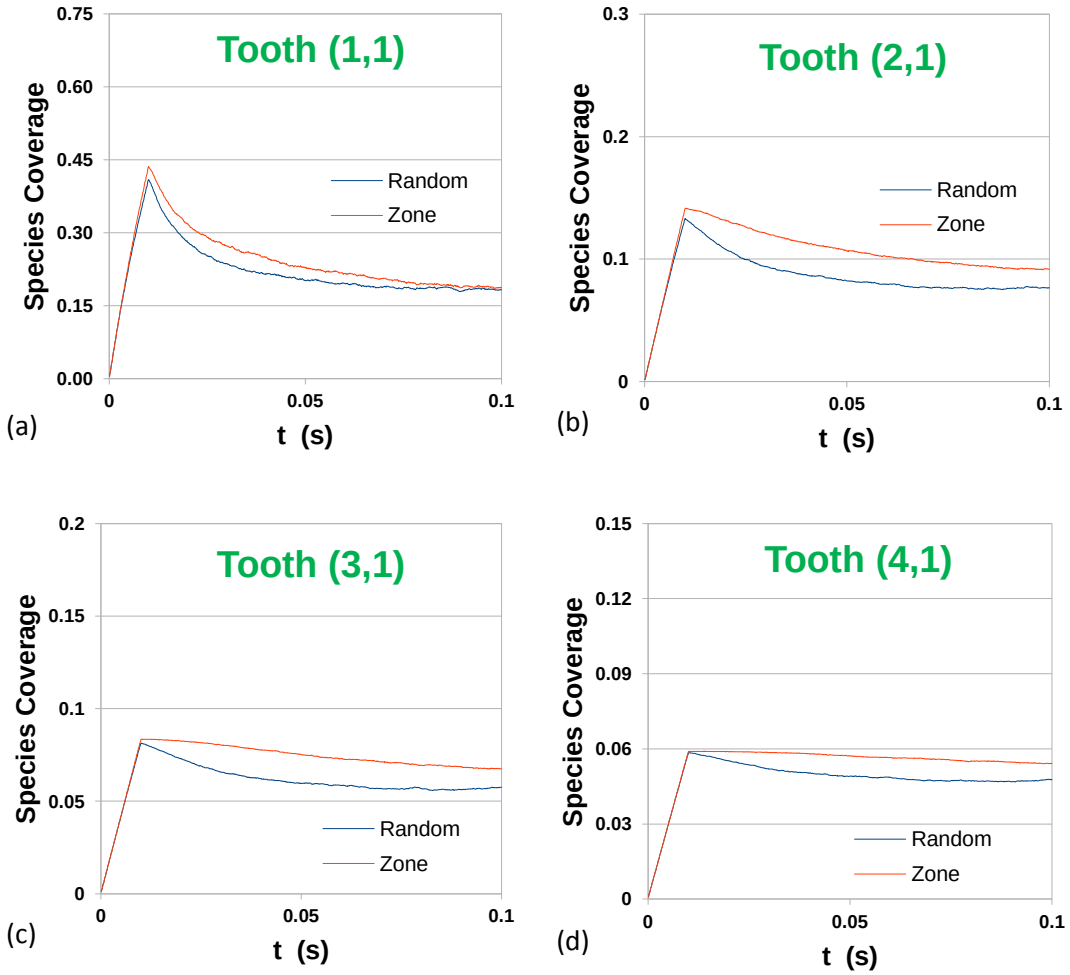


Figure 5.11: Diffusing species cumulative coverage profiles comparison between a 'zone'(red lines) and a 'random' (blue lines) distributions of ingoing species with gaps and teeth of 150 and 100 particles respectively.

Furthermore, as we can see in Figures 5.12 and 5.13, increasing the gap between the teeth to 175 and 200 sites and consequently decreasing the teeth size to 80 and 60 sites respectively, leads to reduced differences between the random and zone distribution gap-tooth simulations. On the contrary, although the differences between the two distribution methodologies are getting reduced when the teeth size is decreased, as can been seen in Figures 5.14 and 5.15, both gap-tooth simulations exhibit increasing differences with the single lattice simulation.

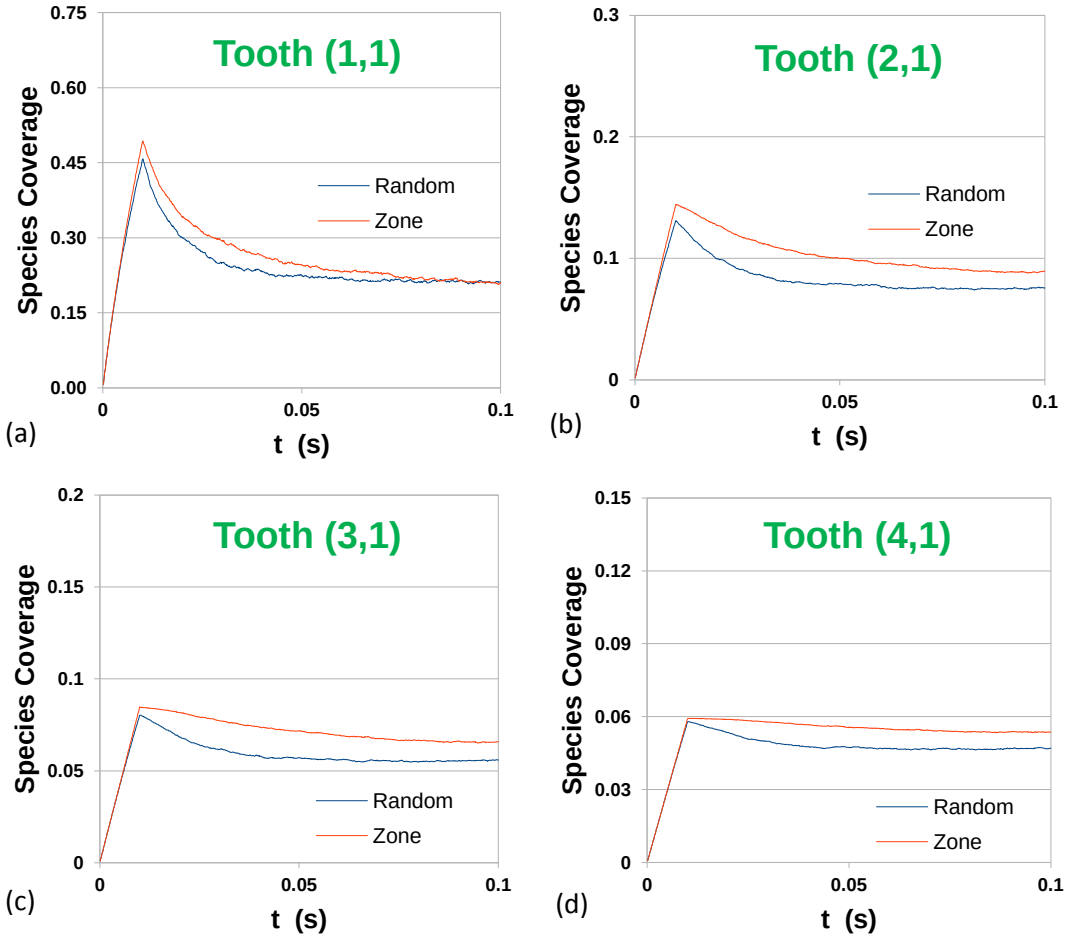


Figure 5.12: Diffusing species cumulative coverage profiles comparison between a 'zone'(red lines) and a 'random' (blue lines) distributions of ingoing species with gaps and teeth of 175 and 80 particles respectively.

This was expected since the zone distribution gets closer to the random one when the tooth size is decreased and as discussed earlier, the random distribution leads to faster diffusion of species in the (no)gap-tooth scheme. Correction factors should be taken into account in order to diminish the observed differences but this is out of the scope of the current study and it will be considered in future work studies.

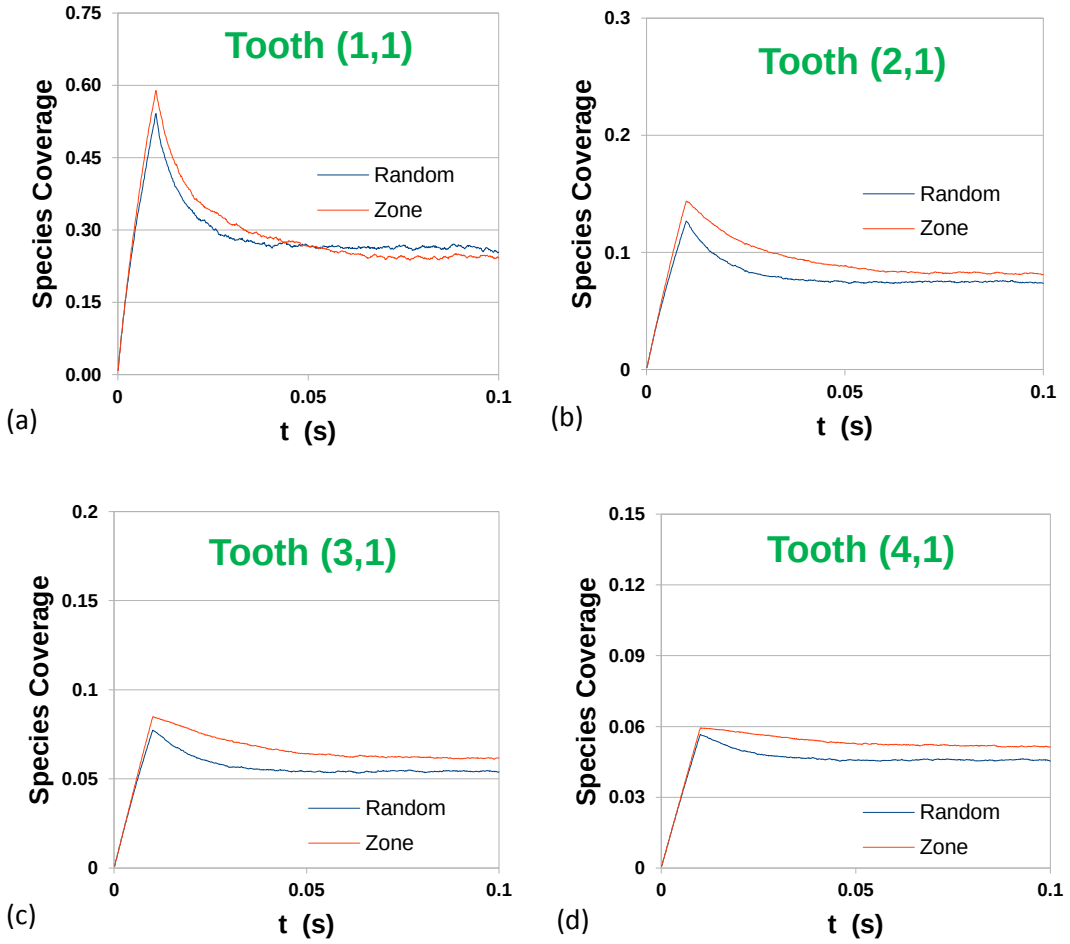


Figure 5.13: Diffusing species cumulative coverage profiles comparison between a 'zone'(red lines) and a 'random' (blue lines) distributions of ingoing species with gaps and teeth of 200 and 60 particles respectively.

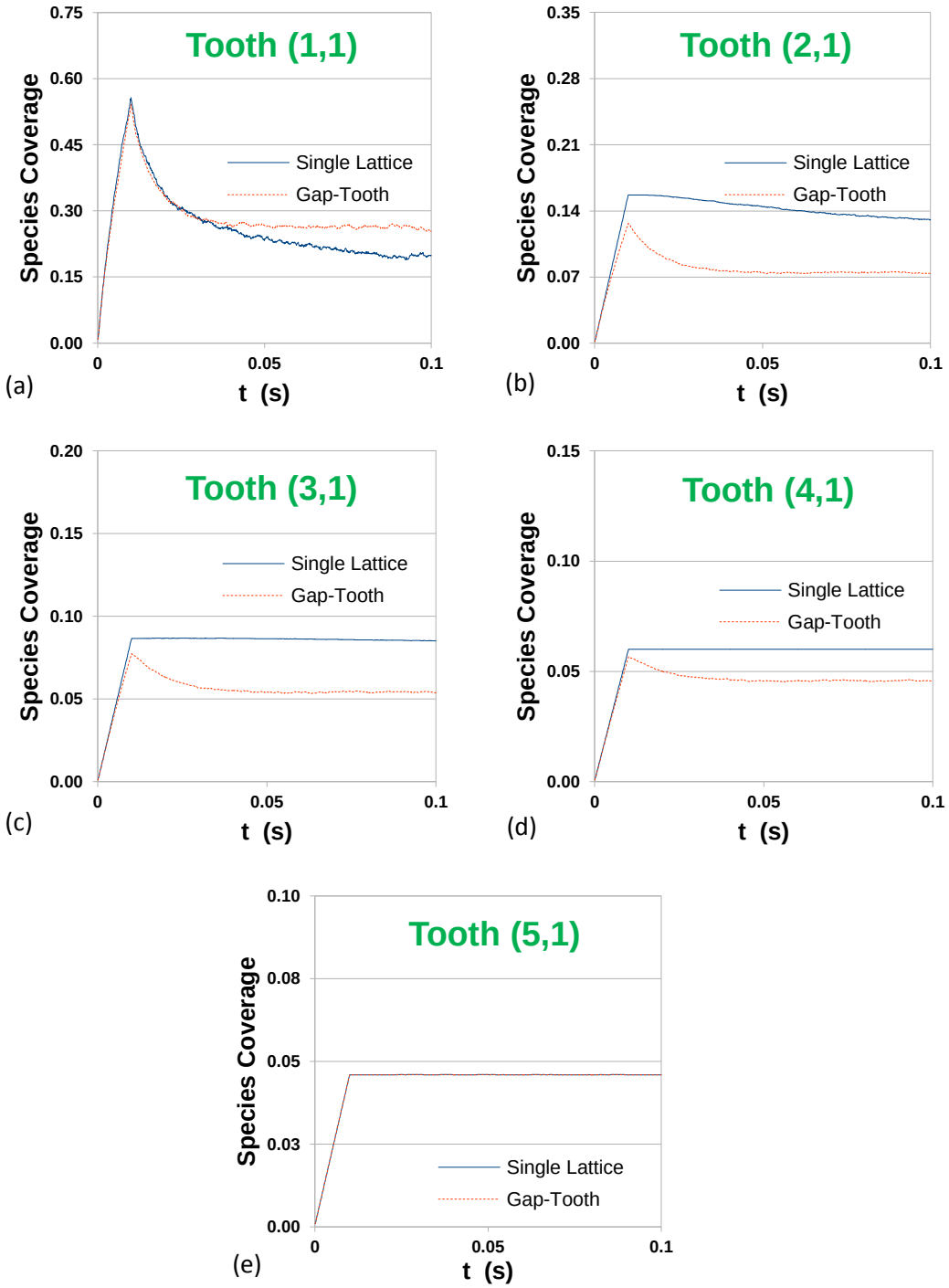


Figure 5.14: Diffusing species cumulative coverage profiles comparison between a single lattice (blue solid lines) and a Gap-Tooth (red dashed lines) framework, using 'random' distribution of ingoing species with gaps and teeth of 200 and 60 particles respectively.

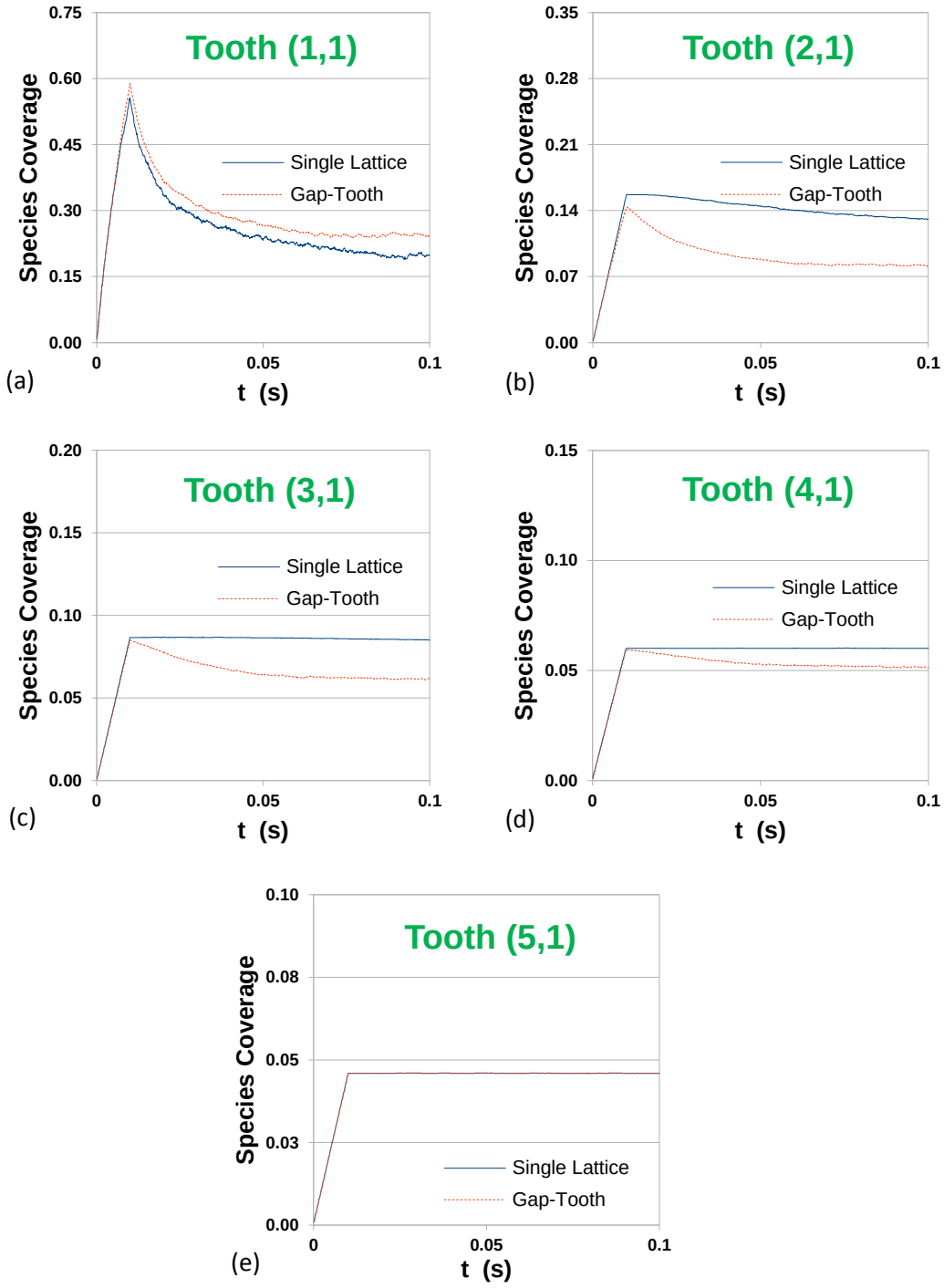


Figure 5.15: Diffusing species cumulative coverage profiles comparison between a single lattice (blue solid lines) and a Gap-Tooth (red dashed lines) framework, using (1-10) 'zone' distribution of ingoing species with gaps and teeth of 200 and 60 particles respectively.

5.3.2 Open circuit kMC simulations using the gap-tooth

Having found that the (1-10) 'zone' distribution could be satisfactorily used in gap-tooth frameworks with no or relatively small gap sizes, it was further employed to perform open circuit kMC simulations using the CO oxidation over Pt as an illustrative system.

Table 5.1: The scheme of CO oxidation micro-processes.

Open-circuit CO oxidation micro-processes	
Adsorption/desorption of O_2	$O_{2\,(g)} + 2\cdot S \underset{k^d_{-13}}{\overset{k^a_{13}}{\rightleftarrows}} 2O\cdot S$ (5.13)
Adsorption/desorption of CO	$CO_{(g)} + \cdot S \underset{k^d_{-14}}{\overset{k^a_{14}}{\rightleftarrows}} CO\cdot S$ (5.14)
Surface reaction between adsorbed O and CO	$O\cdot S + CO\cdot S \xrightarrow{k^r_{15}} CO_{2\,(g)} + 2\cdot S$ (5.15)
CO surface diffusion	$CO\cdot S + \cdot S \xrightarrow{k^{CO*}_{diff}} \cdot S + CO\cdot S$ (5.16)

As before, the utilised 1-D gap-tooth scheme consists of 5 teeth 100x100 sites each (d_x = 100 sites) and 150 sites gap (D_x = 250 sites) between the consecutive teeth (as in Figure 5.8). The CO oxidation micro-processes taken into account in this case study are presented in Table 5.1. They describe the main CO mechanism augmented by a diffusion micro-process for the CO adsorbed on the catalytic surface. The CO diffusion micro-process (5.16) is considered here in order to illustrate the lattice-to-lattice lateral interactions. The selected open circuit model parameters and operating conditions are tabulated in Table 5.2.

Table 5.2: Open circuit kMC model utilised parameters and operating conditions.

Parameter	***Symbol***	***Value***
Catalytic surface temperature, K	T_S	673.15
O_2 partial pressure, Pa	P_{O_2}	500
CO partial pressure, Pa	P_{CO}	500
Reporting horizon, s	$t_{rep}^{gap-tooth}$	10^{-6}
Pt specific site density, mol m^{-2}	N_S	2.5407×10^{-5} (Yentekakis et al., 1994)
Sticking coefficient of O_2 on Pt	S_{O_2}	1.0×10^{-2} (Kaul et al., 1987)
Sticking coefficient of CO on Pt	S_{CO}	3.5×10^{-1} (Kaul et al., 1987)
O_2 desorption pre-exponential factor, s^{-1}	$k^{d}_{o,-13}$	2.4×10^{13} (Kaul et al., 1987)
O_2 desorption activation energy, J mol^{-1}	$E^{d}_{A,-13}$	217568 (Kaul et al., 1987)
CO desorption pre- exponential factor, s^{-1}	$k^{d}_{o,-14}$	6.5×10^{13} (Kaul et al., 1987)
CO desorption activation energy, J mol^{-1}	$E^{d}_{A,-14}$	104600 (Kaul et al., 1987)
CO and O_2 surface reaction pre-exp factor, s^{-1}	$k^{r}_{o,15}$	2.7×10^{6} (Kaul et al., 1987)

CO oxidation activation energy, J mol^{-1}	$E^{r}_{A,15}$	50208 (Kaul et al., 1987)
CO diffusion micro-processes coefficient, s^{-1}	k^{CO}_{diff}	1 (Hari, 2009)

Figure 5.16 illustrates a comparison between the CO coverage profiles resulted by the single lattice and gap-tooth simulations CO coverage profiles. It can be observed that both single lattice and gap-tooth simulations exhibit identical dynamic trends for the whole time range. A comparison between the CO_2 production rate computed by the kMC gap-tooth simulation and that computed by the single lattice kMC simulation is presented in Figure 5.17. As we can see, the former can very accurately capture the single lattice simulation at both short and long term dynamics for just a fraction (78%) of the required CPU time.

To quantify the effect of the lattice-to-lattice lateral interactions on such an open circuit system and operating conditions, we have extended our study to carry out a kMC gap-tooth simulation not taking into account any lateral interactions (CO was not allowed to diffuse outwards each lattice).

Figures 5.18 and 5.19 show a comparison between the CO_2 production rate profiles and the CO coverage respectively, computed by the single lattice kMC simulation and the gap-tooth kMC simulation without lattice-to-lattice lateral interactions. Almost no differences between the two simulations are observed. We believe that this is due to the comparatively small diffusion probability employed in these simulations compared with the adsorption/desorption/surface reaction ones, which means that the effect of these reactions outweighs that of diffusion.

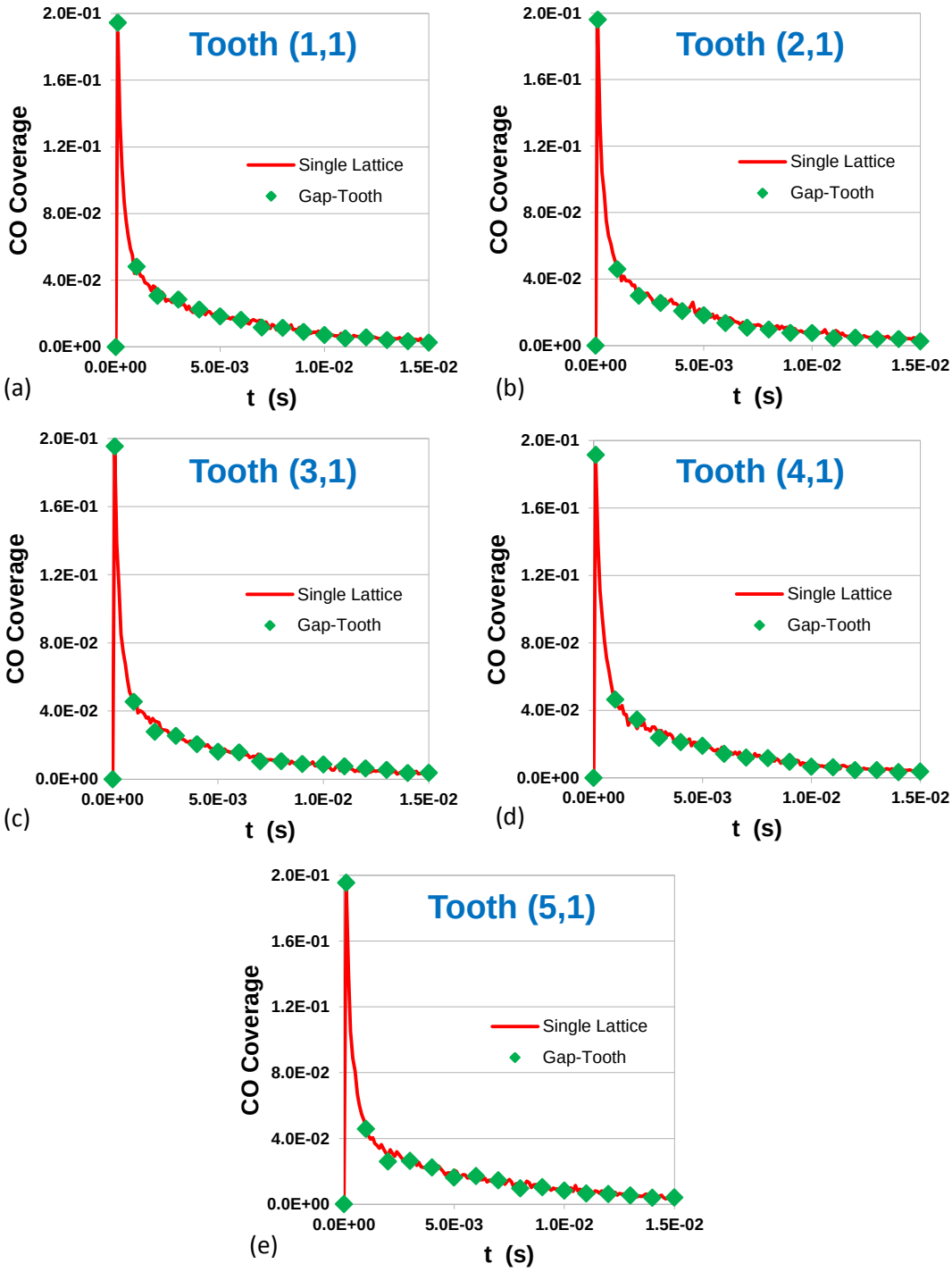


Figure 5.16: CO coverage profiles comparison between a single lattice and (a) Tooth (1,1), (b) Tooth (2,1), (c) Tooth (3,1), (d) Tooth (4,1), (e) Tooth (5,1), with lattice-to-lattice lateral interactions and under open circuit (atmospheric) conditions. The solid (red) lines represent the single lattice simulation and the (green) diamonds represent the gap-tooth simulation.

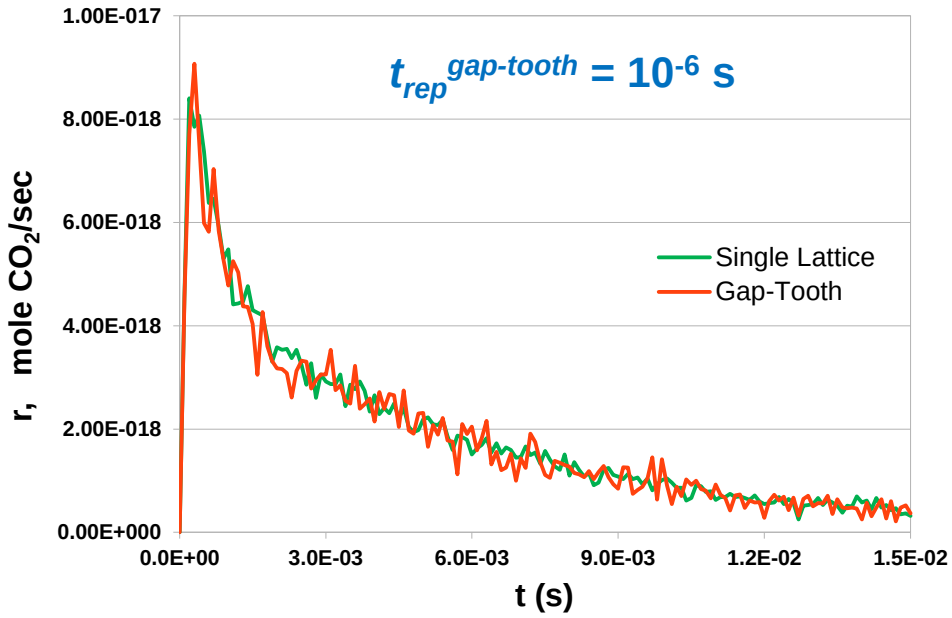


Figure 5.17: CO_2 production rate profiles comparison between a single lattice (green lines) and a gap-tooth (red lines) frameworks with lattice-to-lattice lateral interactions and under open circuit (atmospheric) conditions.

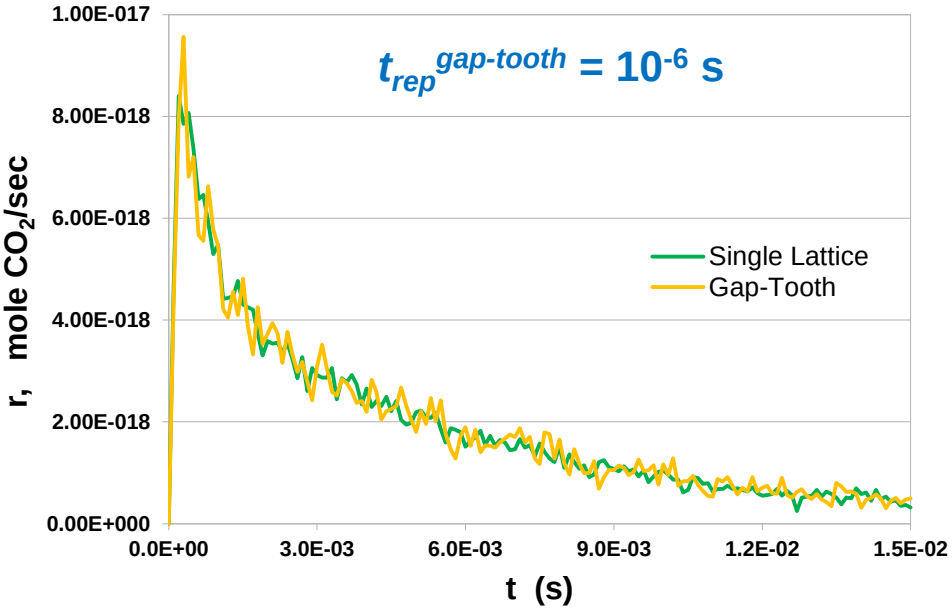


Figure 5.18: CO_2 production rate profiles comparison between a single lattice (green lines) and a gap-tooth (orange lines) frameworks without lattice-to-lattice lateral interactions and under open circuit (atmospheric) conditions.

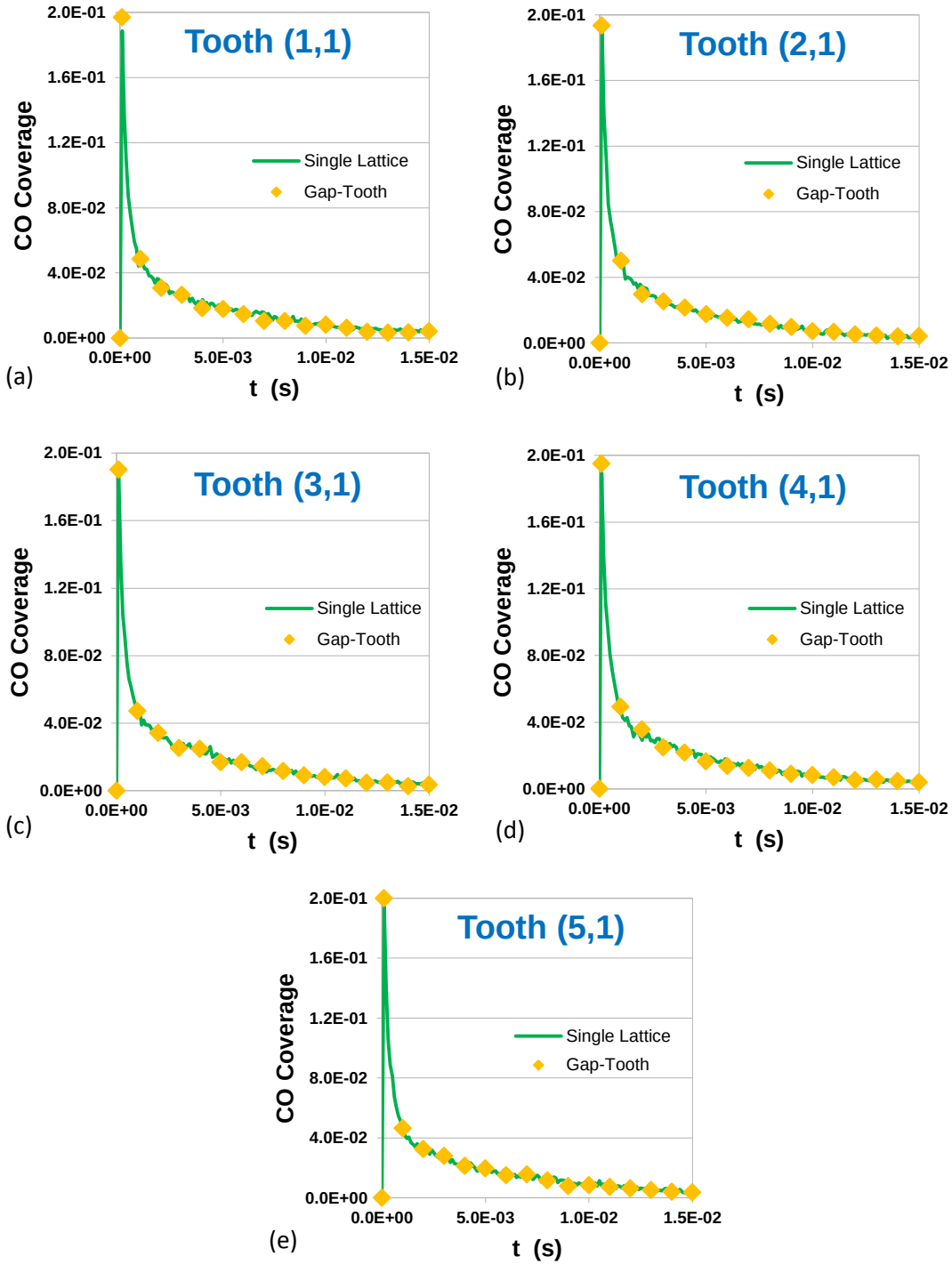


Figure 5.19: CO coverage profiles comparison between a single lattice and (a) Tooth (1,1), (b) Tooth (2,1), (c) Tooth (3,1), (d) Tooth (4,1), (e) Tooth (5,1), with lattice-to-lattice lateral interactions under open circuit (atmospheric) conditions. The solid (green) lines represent the single lattice simulation and the (orange) diamonds represent the gap-tooth simulation.

Furthermore, we have shown in Fragkopoulos and Theodoropoulos (2013) that even when diffusion events represent the 60% of the total micro-processes on the lattice (using a high diffusion probability, 10^6 s^{-1}), the differences between the single lattice and the gap-tooth scheme considering no lattice-to-lattice lateral interactions are very small due the dominant presence of the catalytic micro-processes. Despite this, it is important to take into account diffusional effects in the closed circuit system due to the necessity to control the backspillover species diffusion on the catalytic surface which will subsequently lead to regulating the catalytic performance enhancement.

5.3.3 Multi-scale EPOC simulations using the gap-tooth

In this section we have employed the gap-tooth kMC model to perform 3-D multi-scale electrochemically promoted CO oxidation simulations. To carry out the gap-tooth multi-scale simulations, we have employed the multi-scale system presented in chapter 4 also taking into account the diffusion micro-process for CO (Rxn (5.16)). Here, the multi-scale catalytic lattice of 1800x400 sites is represented by a 2-D gap-tooth framework consisting of 5 teeth 200x100 sites (d_x = 200 sites) each in the x-direction and 2 teeth 200x100 sites (d_y = 200 sites) each in the y-direction, with gaps between the neighbouring teeth of 200 sites (D_x = 400 sites, D_y = 300 sites) in both the x- and y-directions (Figure 5.20). The chosen gap-tooth multi-scale system parameters and operating conditions are tabulated in Tables 4.4 and 5.3 respectively.

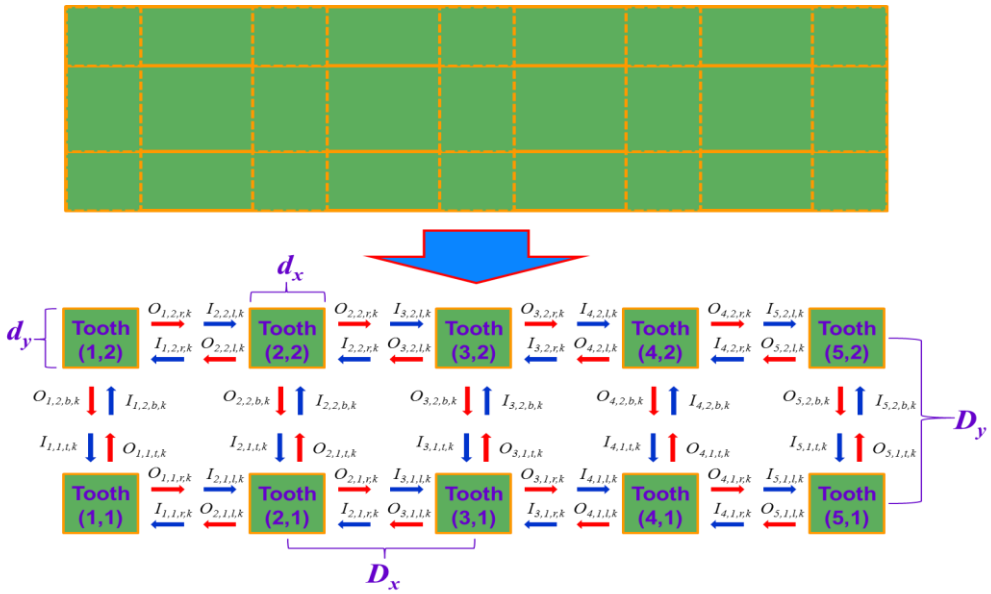


Figure 5.20: Schematic of a 2-D Gap-Tooth representation of the multi-scale system catalytic lattice.

Table 5.3: The gap-tooth multi-scale operating conditions.

Parameter	***Symbol***	***Value***
Operating temperature, K	T_{op}	623.15
CO inlet partial pressure, Pa	P_{O_2}	$5x10^{2}$
O_2 inlet partial pressure, Pa	P_{CO}	$5x10^{3}$
Volumetric flowrate of gas mixture, $m^3\ s^{-1}$	F_d	$2.5x10^{-6}$
Operating potential, V	Φ_{op}	0.7

A comparison between the CO_2 closed-circuit production rates computed by the multi-scale single lattice (1800x400 sites) system and that computed by the CFD/ gap-tooth/kMC scheme, using increasing gap-tooth time reporting horizons ($t_{rep}^{gap-tooth}$), is illustrated in Figures 5.21 to 5.25.

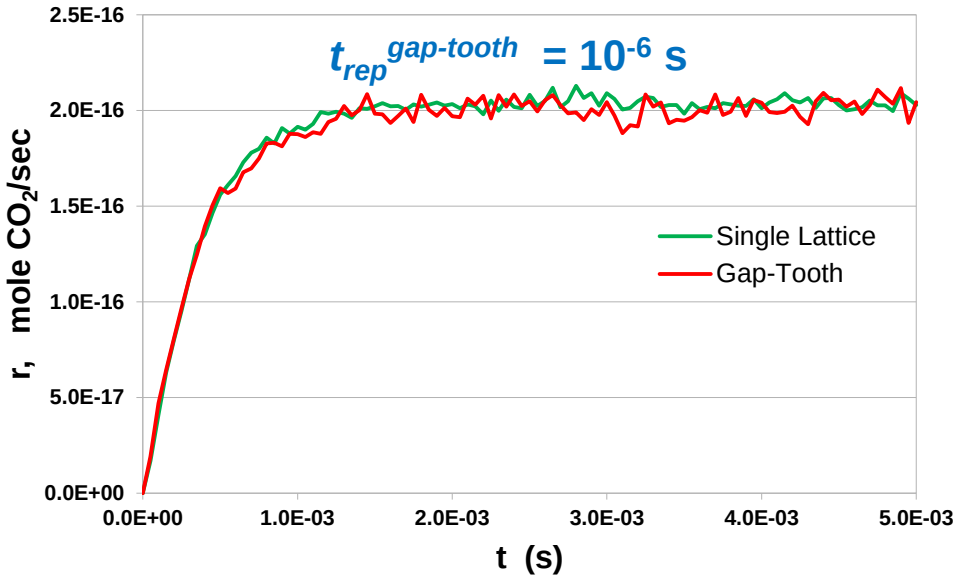


Figure 5.21: CO_2 production rate profiles comparison between the multi-scale single lattice (green lines) and gap-tooth (red lines) frameworks under closed circuit conditions, using a gap-tooth time reporting horizon of 10^{-6} s.

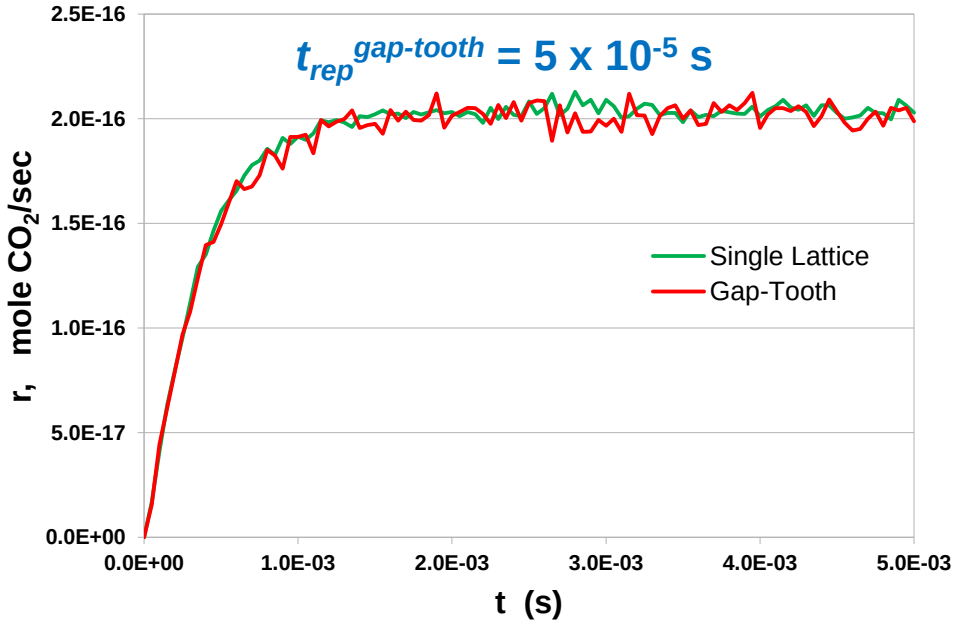


Figure 5.22: CO_2 production rate profiles comparison between the multi-scale single lattice (green lines) and gap-tooth (red lines) frameworks under closed circuit conditions, using a gap-tooth time reporting horizon of 5 x 10^{-5} s.

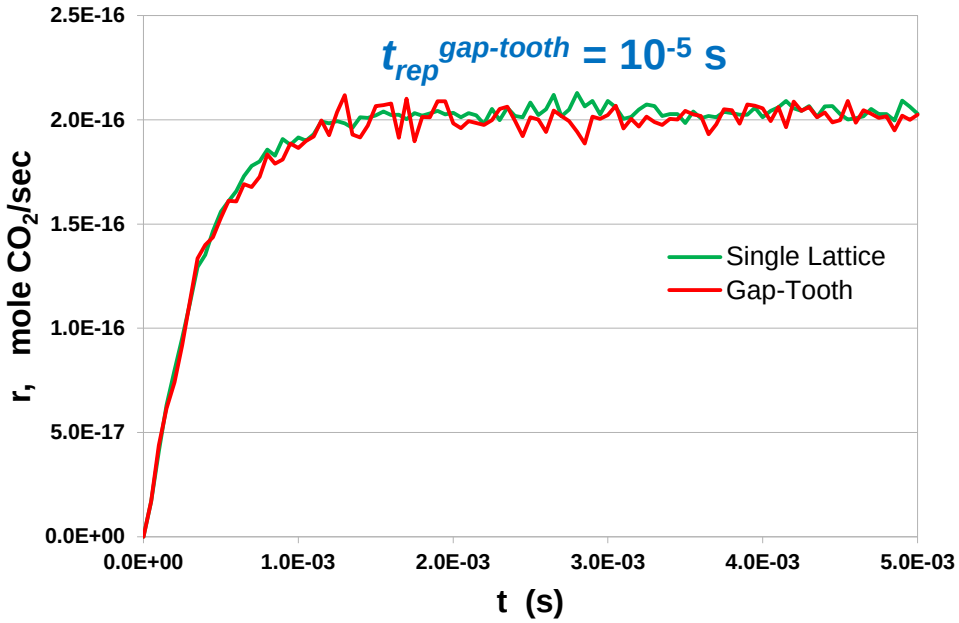


Figure 5.23: CO_2 production rate profiles comparison between the multi-scale single lattice (green lines) and gap-tooth (red lines) frameworks under closed circuit conditions, using a gap-tooth time reporting horizon of 10^{-5} s.

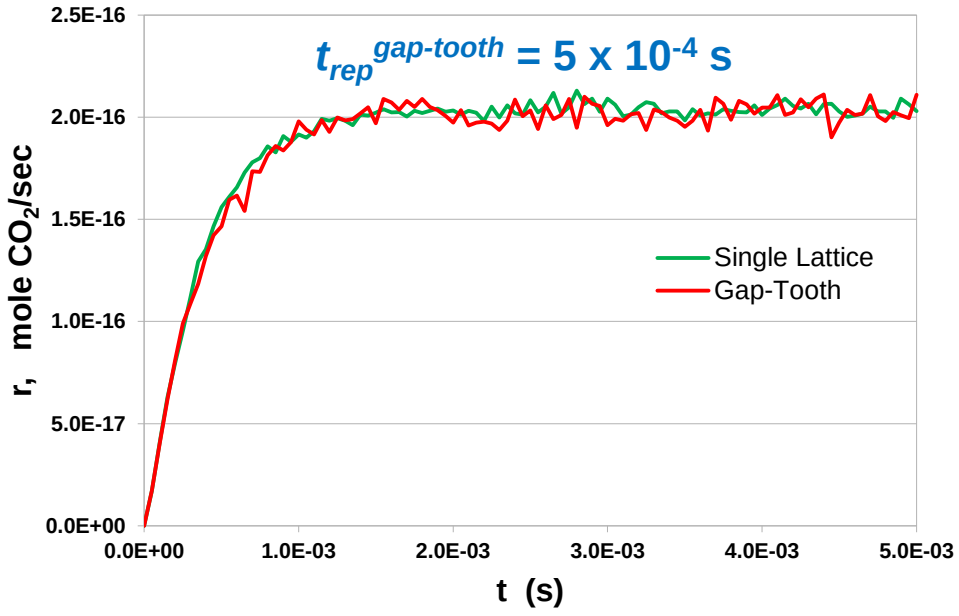


Figure 5.24: CO_2 production rate profiles comparison between the multi-scale single lattice (green lines) and gap-tooth (red lines) frameworks under closed circuit conditions, using a gap-tooth time reporting horizon of 5 x 10^{-4} s.

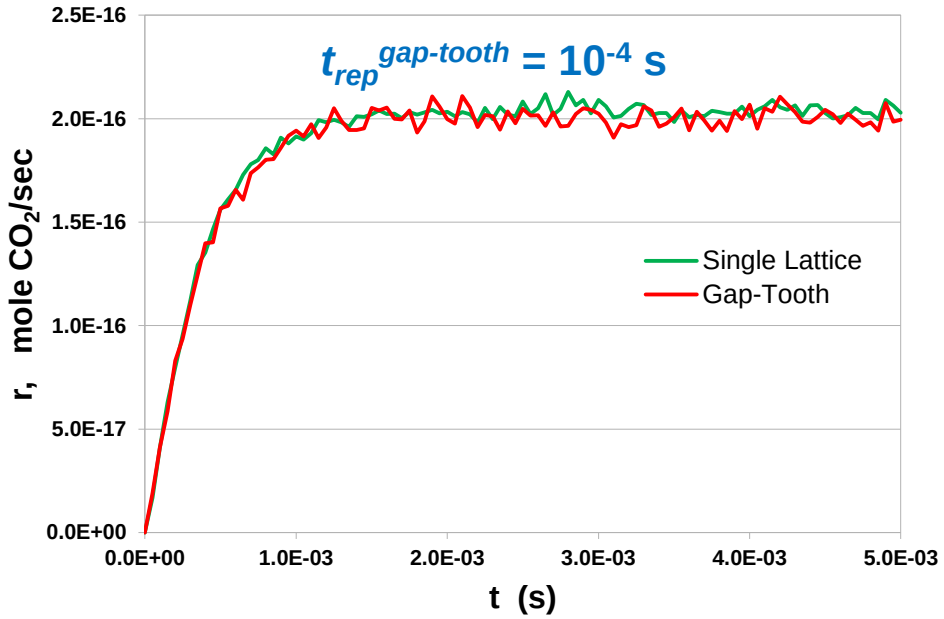


Figure 5.25: CO_2 production rate profiles comparison between the multi-scale single lattice (green lines) and gap-tooth (red lines) frameworks under closed circuit conditions, using a gap-tooth time reporting horizon of 10^{-4} s.

As can been seen in Figures 5.21 to 5.25, both single lattice and gap-tooth multi-scale systems result in identical responses for the whole time range and for all utilised gap-tooth time reporting horizons as expected, since the catalytic micro-processes dominate the system's behaviour (Fragkopoulos and Theodoropoulos, 2013). Also, increasing the gap-tooth time reporting horizon and consequently reducing the system updates due to species interpolation, leads to an increase in the CPU time saving. More specifically, using a gap-tooth time reporting horizon equal to 10^{-6} s leads to a CPU time saving of 15% (Intel® Xeon® Processor 5160 @ 3.00GHz). Raising the utilised gap-tooth time reporting horizon to 10^{-4} s results in a subsequent increase in the CPU time saving from 15% to 71%. The utilised gap-tooth time reporting horizons and the consequent computational savings are summarised in Table 5.4.

Table 5.4: The 2-D closed-circuit gap-tooth computational gain.

Time Reporting Horizon	*CPU Time Saving*
$t_{rep}^{gap-tooth} = 10^{-6}$ s	15 %
$t_{rep}^{gap-tooth} = 5 \times 10^{-5}$ s	33 %
$t_{rep}^{gap-tooth} = 10^{-5}$ s	49 %
$t_{rep}^{gap-tooth} = 5 \times 10^{-4}$ s	62 %
$t_{rep}^{gap-tooth} = 10^{-4}$ s	71 %

We should note here the difference between the time reporting horizons for the kMC (t_{rep}^{kMC}) and the gap-tooth ($t_{rep}^{gap-tooth}$) simulations. The former is related to the frequency at which the kMC and macroscopic simulations are communicating their results in the multi-scale framework, while the latter refers to the frequency of ingoing species interpolation updates in the gap-tooth framework.

5.4 Conclusions

The objective of the work presented in this chapter was the development of a gap-tooth lattice kMC framework to perform large open and closed circuit catalytic simulations with computational efficiency. The micro-catalytic domain has been divided into a number of representative micro-lattices, the area of which was just a fraction of the total catalytic surface. Efficient coarse-graining techniques implemented through the gap-tooth method have been employed for the simulation of the lattice-to-lattice lateral interactions between neighbouring micro-surfaces.

The framework was first validated against a single lattice (whole catalytic surface) simulation considering no gap between neighbouring teeth for a single diffusion micro-process, using linear interpolation rules and several lattice-to-lattice species exchange fluxes distribution techniques.

It was found that distributing the ingoing species within thin zones around the boundaries (rather than randomly in the whole micro-lattice domain or at the lattice boundary areas) leads to an ideal representation of the single lattice simulation.

To analyse the effect of the selected gap on the species evolution in the gap-tooth area, the single diffusion gap-tooth framework was further exploited taking into account a couple of different gap/tooth dimensions. It was observed that although increasing the gap areas between the teeth and consequently decreasing the teeth size resulted in reduced differences between the random and zone distribution methodologies, the error between the gap-tooth and single lattice simulations was increased.

Having validated the in-house developed gap-tooth kMC framework using only the diffusion micro-process for a single species, this methodology was employed in an open circuit CO oxidation system taking into account all the corresponding micro-processes augmented by the CO diffusion one so as to illustrate the effect of the lattice-to-lattice lateral interactions on such a system. It was found that the single lattice responses were very accurately captured by the full gap-tooth simulation ones for the whole time range. Nevertheless, it was shown that the lattice-to-lattice lateral interactions effect on the system simulation was rather minor due to the dominant presence of the adsorption/desorption/reaction micro-processes taking place in such systems.

The gap-tooth scheme was further employed in the electrochemically promoted CO oxidation multi-scale framework presented in the previous chapter giving accurate dynamic profiles of the resulting closed-circuit CO_2 production rates -and for a fraction of the total computational cost.

It was observed that increasing the gap-tooth time reporting horizon (meaning that the number of the lattice-to-lattice interpolation events was reduced) up to 10^{-4} s leads to increases in the computational gain up to 71%, also maintaining identical CO_2 production rate responses.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The main objective of this thesis was the formulation of a systematic and accurate EPOC framework to describe the mass and charge transport phenomena taking place in electrochemically promoted systems as well as to enhance our understanding about the polarisation effect on heterogeneously catalysed systems. Such a detailed framework could be a valuable tool for robust design and control of scaled-up EPOC systems which will consequently increase their industial potential.

The first goal of this study was to develop a systematic macroscopic, multi-dimensional, isothermal, dynamic solid oxide single pellet model in order to describe the catalytic and electrocatalytic processes taking place in an electrochemically promoted CO oxidation system and to demonstrate the NEMCA effect on the selected reaction system under the application of external voltages. The constructed model takes into account mass and charge transport as well as the electrochemical processes occuring at the TPBs of the cathode and anode electrodes. COMSOL Multiphysics, a commercial CFD package, was used to perform the simulation of the model employing the FEM for the simultaneous solution of the set of PDEs. Furthermore, the facility of

"integrated coupling variables" incorporated in this FEM-based software was utilised so as to implement each gas species partial pressure expressions. The model has the ability to predict the species coverage on the catalytic surface, electronic and ionic potential profiles throughout the pellet, gas mixture concentration within the reactor and CO_2 production rates.

The model was exploited for a parameter estimation study since kinetic parameters, such as sticking coefficients and activation energies for the electrocatalysed reactions, were expected to be different from the open circuit ones and electrochemical kinetic parameters, such as anode exchange current density pre-exponential coefficients, were not available in the literature. To accomplish this, a combination of stochastic (SA) and deterministic (SQP) optimization techniques were carried out using a tailored 2-D version of our model (through the use of the scripting facility of COMSOL Multiphysics) to fit model predictions with experimental data available in the literature (Yentekakis et al., 1994). Potential identifiability issues due to the limited number of available experimental data were addressed through multiple restarts for the stochastic optimizer. The estimated parameters were within literature reported lower and upper limits (Kaul et al., 1987). It was shown that both 2- and 3-D models could predict physically realistic trends.

Sensitivity analysis for the estimated parameters was carried out to obtain insights on the reliability of the parameter estimation problem and to quantify the effect of each parameter on the observed CO_2 production rate. It was found that variations in the catalytic kinetic constants could result in significant alterations in the CO_2 production rate, while the ones in the rate constants of reactions where the BSS was participating in

as well as in the exchange current density pre-exponential coefficients, led to minor or almost no changes in the CO_2 production rate. This observation suggests that the electrochemically induced alteration in the CO_2 production rate was essentially due to the non-Faradaic (NEMCA) effect.

CO_2 production rate transients have been generated to study the parametric effect of various operating parameters on the catalytic performance. It was shown that raising the P_{CO}^{in} favoured "volcano-type" behaviour in the CO_2 NEMCA production rate, while "S-type" behaviour was demonstrated in the CO_2 Faradaic production rate. On the contrary, raising the $P_{O_2}^{in}$ favoured "S-type" behaviour in the CO_2 production rate in both non-Faradaic (NEMCA) and Faradaic rate profiles. Furthermore, an enhancement in the catalytic performance was demonstrated when the operating temperature was raised and for a ratio of inlet partial pressures of $P_{O_2}^{in} / P_{CO}^{in} < 6$.

Having found that the non-Faradaic effect plays a dominant role in the enhancement of the catalytic performance, we extended our model to simulate the CO combustion reaction-diffusion phenomena taking place on the catalyst surface at the molecular level. A comprehensive 3-dimensional, isothermal, dynamic solid oxide single pellet multi-scale framework was developed for the investigation of the chemical and electrochemical phenomena at their appropriate length-scales.

The proposed framework couples a 3-D macroscopic model for charge transport simulations throughout the solid oxide single pellet using a FEM-based package (COMSOL Multiphysics), with a 2-D microscopic model for the execution of stochastic reaction-diffusion catalytic simulations employing an in-house developed lattice kinetic

Monte Carlo algorithm written in FORTRAN programming language. The FEM/FORTRAN communication was achieved through the use of a MATLAB interface.

CO_2 production rate curves have been generated to compare the performance of the proposed multi-scale framework with a fully macroscopic one (Fragkopoulos et al., 2013) under atmospheric conditions. Although the models seemed to exhibit similar dynamic trends for the sets of utilised conditions, differences have been found between their steady state CO_2 production rates. Furthermore, increasing the CO inlet partial pressures was found to lead to reducing differences between the models' steady state outputs, while minor or almost no differences were demonstrated for a ratio of inlet partial pressures of $P_{O_2}^{in} / P_{CO}^{in} < 7$. Finally, maximum catalytic performance in the macro- and multi-scale frameworks was achieved at different inlet gas phase partial pressure ratios. This evidence demonstrates the importance and the added value of using a multi-scale approach for the simulation of such electrochemically promoted systems.

The multi-scale framework was further exploited for a temperature parametric investigation. This study has shown that increasing the operating temperature leads to significant increases in the CO_2 production rate. Moreover, modifications in the system's behaviour have been observed. "Volcano-type" behaviour was favoured for temperatures lower than or equal to 400 °C and "S-type" behaviour was observed for temperatures greater or equal to 450°C.

Finally, in the last section of this thesis the construction of an advanced and computationally efficient framework for open (micro-scopic) and closed circuit (multi-

scale) large lattice kMC simulations was proposed. The computational requirement of such simulations is significant and it becomes even more vigorous as the size of the system increases. As a result, the use of computationally efficient coarse-graining techniques becomes necessary (Gorban et al., 2006). Such a coarse-graining methodology, the gap-tooth method (Gear et al., 2002), has been employed in this study for the simulation of the reaction-diffusion phenomena occuring in the computationally demanding systems of interest. In the proposed gap-tooth scheme, the catalytic domain has been divided into a number of representative micro-lattices, the area of which was just a fraction of the total catalytic surface. Furthermore, intelligent linear (1^{st} order) interpolation rules (Gear et al., 2003) were employed for the simulation of the lattice-to-lattice lateral interactions between consecutive micro-surfaces.

The gap-tooth/kMC framework was validated against an entire (single) lattice kMC simulation taking into consideration the diffusion micro-process of a single species as well as no gap between neighbouring teeth. Such a 'no'gap-tooth/kMC system can be a valuable tool for efficient simulations of catalytic systems. For the validation study, several lattice-to-lattice species exchange fluxes distribution techniques were employed. Distributing the exchanged species within thin zones around the boundaries of each micro-lattice was found to exhibit an ideal representation of the single lattice simulation.

The gap-tooth/kMC framework was further exploited to investigate the effect of the gap areas size on the species evolution in the gap-tooth domain. Here, the simulation of the single diffusion example - using random and zone distribution techniques - was carried out taking into consideration various gap/tooth sizes. Although increasing the gap areas between the teeth and consequently decreasing the teeth size was found to

lead to reduced differences between the random and zone distribution techniques, the error between the gap-tooth and single lattice simulations was increased. This observation was expected since the zone distribution gets closer to the random one when decreasing the lattice size.

Having validated the in-house developed gap-tooth/kMC framework using only the diffusion micro-process for a single species, this methodology was further used for open circuit CO oxidation (microscopic) simulations. This case study has shown that the single lattice kMC responses were very accurately captured by the full gap-tooth/kMC simulation ones for the whole time range. However, it was observed that the lattice-to-lattice lateral interactions (through the diffusion micro-process) effect on the system simulation was minor due to the dominant presence of the reaction micro-processes occuring in such systems (Fragkopoulos and Theodoropoulos, 2013).

Finally, a FEM/gap-tooth/kMC multi-scale scheme was constructed and employed for electrochemically promoted CO combustion simulations. The proposed framework has demonstrated a very accurate representation of the single lattice NEMCA CO_2 production rate dynamics for just a fraction of the total computational requirement. Furthermore, increasing the utilised time reporting horizon up to 10^{-4} sec resulted in CPU time saving increases to up to 71 %.

To this end, we believe that exploiting such electrochemically promoted systems for scaled-up simulations in combination with high-fidelity experimental data will quantify the polarisation effect on the catalytic performance and subsequently, it will enhance their industrial potential.

6.2 Future Work

To the best of the author's knowledge, the models presented in this thesis have been the first attempt to describe the NEMCA effect and cover most of its aspects in a systematic and accurate level. Nevertheless, there are still limitations that need to be further investigated in order to increase the predictive capability of the formulated frameworks. Recommendations for potential extension of this work which could improve the proposed models and increase the EPOC system understanding, enabling for the design and control of scaled-up commercial systems, are listed in the following paragraphs.

The developed solid oxide single pellet macroscopic model is capable of predicting the electrochemically enhanced CO_2 production rates sufficiently. However, the model was validated against only one set of temperature and applied potential data, due to the lack of experimental data available in literature. Moreover, the effect of the BSS diffusion on the reaction rates has not been explicitly quantified for the same reason. To overcome these limitations, high-fidelity experiments that should clearly describe the effect of the applied potential on both the catalyst work function and the reaction rate need to be carried out for a wide range of operating conditions and to be used for model validation purposes taking into account promotion rules such as the ones proposed by Vayenas and co-workers (Vayenas et al., 2001b; Brosda and Vayenas, 2002).

The developed gap-tooth/kMC model was validated against a single lattice kMC model considering no gaps between consecutive teeth, i.e. fully representing the entire computational domain, and a very good agreement for the whole time range was

demonstrated. Such a 'no'gap-tooth/kMC model can be efficiently used for single lattice kMC simulations employing the widely used message passing interface (MPI) parallelisation technique (Pacheco, 1998). Furthermore, using gaps between neighbouring teeth led to small differences between the gap-tooth/kMC and the single kMC dynamics in cases where diffusion was the dominant or the only micro-process taken into account. More specifically, the species evolution in the gap-tooth/kMC framework seemed to be slightly faster than the one in the single kMC model. To eliminate these differences, correction factors should be taken into consideration in the gap-tooth/kMC framework. Nevertheless, since the scope of this study was the efficient simulation of an EPOC system, and since the FEM-gap-tooth-kMC EPOC multi-scale framework developed in this study was capable of very accurately capturing the CO_2 production rate transients for both short and long term dynamics with computational efficiency, we have not considered such correction factors in the present work.

Finally, the coupling of electrochemically promoted catalytic processes (NEMCA) and electrocatalysis (fuel cell technology) is very promising nowadays. Caravaca et al. (2011) were the first to report such a possibility suggesting that the EPOC phenomenon coupled with the steam electrolysis process could lead to increased H_2 production rates. Nevertheless, this combination will not be possible if the electrochemically induced promotion effect will not first be sufficiently quantified. We strongly believe that the models developed and described in this thesis aided by a good range of experimental data will ultimately improve our insights for complex phenomena occuring in EPOC systems. Consequently, the incorporation of such effects in widely used fuel cell systems will be tackled bringing new challenges in modern electrocatalysis.

Bibliography

Achenbach E., *Three-dimensional and time-dependent simulation of a planar solid oxide fuel cell stack*. Journal of Power Sources, **49**, 333-348, 1994.

Aguiar P., C.S. Adjiman and N.P. Brandon, *Anode-supported intermediate temperature direct internal reforming solid oxide fuel cell. I: Model-based steady-state performance*. Journal of Power Sources, **138**, 120-136, 2004.

Alqahtany H., P.-H. Chiang, D. Eng, M. Stoukides and A. Robbat, *Electrocatalytic decomposition of hydrogen sulfide*. Catalysis Letters, **13**, 289-296, 1992.

Andersson M., J. Yuan and B. Sundén, *Review on modeling development for multiscale chemical reactions coupled transport phenomena in solid oxide fuel cells*. Applied Energy, **87**, 1461-1476, 2010.

Andreassi L., C. Toro and S. Ubertini, *Modeling carbon monoxide direct oxidation in solid oxide fuel cells*. Journal of Fuel Cell Science and Technology, **6**, 21307-1-15, 2009.

Andreassi L., G. Rubeo, S. Ubertini, P. Lunghi and R. Bove, *Experimental and numerical analysis of radial flow solid oxide fuel cell*. International Journal of Hydrogen Energy, **32**, 4559-4574, 2007.

Armaou A., I.G. Kevrekidis and C. Theodoropoulos, *Equation-free gaptooth-based controller design for distributed complex/multiscale processes*. Computers and Chemical Engineering, **29**, 731-740, 2005.

Bakker E., *Electrochemical sensors*. Analytical Chemistry, **76**, 3285-3298, 2004.

Balbuena P.B. and V.R. Subramanian, *Theory and experiment in electrocatalysis*, 2010. New York: Springer.

Balomenou S., D. Tsiplakides, A. Katsaounis, S. Thiemann-Handler, B. Cramer, G. Foti, C. Comninellis and C.G. Vayenas, *Novel monolithic electrochemically promoted*

catalytic reactor for environmentally important reactions. Applied Catalysis B: Environmental, **52**, 181-196, 2004.

Balomenou S.P., D. Tsiplakides, A. Katsaounis, S. Brosda, A. Hammad, G. Foti, C. Comninellis, S. Thiemann-Handler, B. Cramer and C.G. Vayenas, *Monolithic electrochemically promoted reactors: A step for the practical utilisation of electrochemical promotion*. Solid State Ionics, **177**, 2201-2204, 2006.

Baranova E.A., A. Thursfield, S. Brosda, G. Foti, C. Comninellis and C.G. Vayenas, *Electrochemical promotion of ethylene oxidation over Rh catalyst thin films sputtered on YSZ and TiO_2/YSZ supports*. Journal of The Electrochemical Society, **152**, E40-E49, 2005.

Baranova E.A., G. Foti and C. Comninellis, *Promotion of Rh catalyst interfaced with TiO_2*. Electrochemistry Communications, **6**, 170-175, 2004.

Bard A.J. and L.R. Faulkner, *Electrochemical methods: Fundamentals and applications*, 2nd ed. 2001. New York: John Wiley & Sons, Inc.

Bartholomew C.H. and R.J. Farrauto, *Fundamentals of industrial catalytic processes*, 2005. New York: John Wiley & Sons, Inc.

Bebelis S. and C.G.Vayenas, *Non-Faradaic electrochemical modification of catalytic activity: 1. The case of ethylene oxidation on Pt*. Journal of Catalysis, **118**, 125-146, 1989.

Bessler W.G., J. Warnatz and D.G. Goodwin, *The influence of equilibrium potential on the hydrogen oxidation kinetics of SOFC anodes*. Solid State Ionics, **177**, 3371-3383, 2007a.

Bessler W.G., S. Gewies and M. Vogler, *A new framework for physically based modeling of solid oxide fuel cells*. Electrochimica Acta, **53**, 1782-1800, 2007b.

Boggs P.T. and J.W. Tolle, *Sequential quadratic programming*. Acta Numerica, **4**, 1-51, 1995.

Bond G.C., *Principles of catalysis*, 1972. London: The Chemical Society.

Bonis I., S. Valiño-Pazos, I.S. Fragkopoulos and C. Theodoropoulos, *Modelling of micro- and nano-patterned electrodes for the study and control of spillover processes in catalysis*. Computer Aided Process Engineering, **29**, 151-155, 2011.

Bortz A.B., M.H. Kalos and J.L. Lebowitz, *A new algorithm for Monte Carlo simulation of Ising spin systems*. Journal of Computational Physics, **17**, 10-18, 1975.

Boudart M. and G. Djéga-Mariadassou, *Kinetics of heterogeneous catalytic reactions*, 1984. Princeton: Princeton University Press.

Bove R. and S. Ubertini, *Modeling solid oxide fuel cell operation: Approaches, techniques and results*. Journal of Power Sources, **159**, 543-559, 2006.

Brosda S. and C.G. Vayenas, *Rules and mathematical modeling of electrochemical and classical promotion. 2: Modeling*. Journal of Catalysis, **208**, 28-53, 2002.

Campbell C.T., G. Ertl, H. Kuppers and J. Segner, *A molecular beam study of the catalytic oxidation of CO on a Pt(111) surface*. Journal of Chemical Physics, **73**, 5862-5873, 1980.

Campbell C.T., G. Ertl, H. Kuppers and J. Segner, *A molecular beam investigation of the interactions of CO with a Pt(111) surface*. Surface Science, **107**, 207-219, 1981a.

Campbell C.T., G. Ertl , H. Kuppers and J. Segner, *A molecular beam investigation of the interactions of CO with a Pt(111) surface*. Surface Science, **107**, 220-236, 1981b.

Caravaca A., A. de Lucas-Consuegra, C. Molina-Mora, J.L. Valverde and F. Dorado, *Enhanced H_2 formation by electrochemical promotion in a single chamber steam electrolysis cell*. Applied Catalysis B: Environmental, **106**, 54-62, 2011.

Chaisantikulwat A., C. Diaz-Goano and E.S. Meadows, *Dynamic modelling and control of planar anode-supported solid oxide fuel cell*. Computers and Chemical Engineering, **32**, 2365-2381, 2008.

Cheddie D. and N. Munroe, *Parametric model of an intermediate temperature PEMFC.* Journal of Power Sources, **156**, 414-423, 2006.

Chen D., W. Bi, W. Kong and Z. Lin, *Combined micro-scale and macro-scale modeling of the composite electrode of a solid oxide fuel cell*. Journal of Power Sources, **195**, 6598-6610, 2010.

Chen G., *Electrochemical technologies in wastewater treatment*. Separation and Purification Technology, **38**, 11-41, 2004.

COMSOL, *COMSOL Multiphysics 3.5a - Reference Guide*, 2008. COMSOL AB.

COMSOL, *COMSOL Multiphysics 4.2a - Material Library*, 2011. COMSOL AB.

Constantinou I., D. Archonta, S. Brosda, M. Lepage, Y. Sakamoto and C.G. Vayenas, *Electrochemical promotion of NO reduction by C_3H_6 on Rh catalyst-electrode films supported on YSZ and on dispersed Rh/YSZ catalysts*. Journal of Catalysis, **251**, 400–409, 2007.

Costamagna P., A. Selimovic, M.D. Borghi and G. Agnew, *Electrochemical model of the integrated planar solid oxide fuel cell (IP-SOFC)*. Chemical Engineering Journal, **102**, 61-69, 2004.

De Lucas-Consuegra A., F. Dorado, C. Jimenez-Borja and J.S. Varlerde, *Influence of the reaction conditions on the electrochemical promotion by potassium for the selective catalytic reduction of N_2O by C_3H_6 on platinum*. Applied Catalysis B: Environmental, **78**, 222-231, 2008.

Dooling D.J. and L.J. Broadbelt, *Generic monte carlo tool for kinetic modeling*. Industrial and Engineering Chemistry Research, **40**, 522-529, 2001.

Dornte R.W., *Oxidation of white oils*. Industrial & Engineering Chemistry, **28**, 26-30, 1936.

Ducros R. and R.P. Merrill, *The interaction of oxygen with Pt(100)*. Surface Science, **55**, 227-245, 1976.

Dumont M., M. Poriaux and R. Dagonnier, *On surface reaction kinetics in the presense of islands*. Surface Science, **169**, L307-L310, 1986.

Eastern Manufacturing Inc., http://www.easterncatalytic.com.

Eley D.D. and E.K. Rideal, *The catalysis of the parahydrogen conversion by tungsten*. Proceedings of the Royal Society A, **178**, 429-451, 1941.

Ellis T.M.R, I.R. Philips and T.M. Lahey, *Fortran 90 programming*, 1994. Edinburgh Gate: Addison-Wesley Publishers Ltd.

Ercolessi F. and J.B. Adams, *Interatomic potentials from first-principles calculations: The force-matching method*. Europhysics Letters, **26**, 583-588, 1994.

Ertl G., *Reactions at well-defined surfaces*. Surface Science, **299-300**, 742-754, 1994.

Ertl G., H. Knötzinger and J. Weitcamp, *Handbook of heterogeneous catalysis*, 1997. Weinheim: Wiley-VCH Publishers.

Ferguson J.R., J.M. Fiard and R. Herbin, *Three-dimensional numerical simulation for various geometries of solid oxide fuel cells*. Journal of Power Sources, **58**, 109-122, 1996.

Fichthorn K.A. and W.H. Weinberg, *Theoretical foundations of dynamical Monte Carlo simulations*. Journal of Chemical Physics, **95**, 1090-1096, 1991.

Fichthorn K., E. Gulari and R. Ziff, *Noise-induced bistability in a Monte Carlo surface-reaction model*. Physical Review Letters, **63**, 1527-1530, 1989.

Foti G., V. Stankovic, I. Bolzonella and C. Comninellis, *Transient behavior of electrochemical promotion of gas-phase catalytic reactions*. Journal of Electroanalytical Chemistry, **532**, 191-199, 2002.

Fragkopoulos I.S. and C. Theodoropoulos, *Multi-scale modelling of electrochemically promoted systems*. Electrochimica Acta, 2014. *To be submitted.*

Fragkopoulos I.S. and C. Theodoropoulos, *Multiscale models of electrochemically-promoted large catalytic surfaces*. Proceedings of the 3rd International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH), 155-162, 2013.

Fragkopoulos I.S., I. Bonis and C. Theodoropoulos, *Macroscopic multi-dimensional modelling of electrochemically promoted systems*. Chemical Engineering Science, **104**, 647-661, 2013.

Fragkopoulos I.S., I. Bonis and C. Theodoropoulos, *Multiscale modelling of spillover processes in heterogeneous catalytic systems*. Computer Aided Process Engineering, **30**, 1013-1017, 2012.

Frenzel A., C.G. Vayenas, A. Giannikos, P. Petrolekas and C. Pliangos, *Hydrogenation of organic compounds with the use of the NEMCA effect*, 2001. US Patent 6,194,623.

Fuel Cell Today, http://www.fuelcelltoday.com.

Gates B.C., *Catalytic chemistry*, 1992. Singapore: John Wiley & Sons, Inc.

Gear C.W. and I.G. Kevrekidis, *Boundary processing for Monte Carlo simulations in the gap-tooth scheme*. Computational Physics, arXiv:physics/0211043v1, 2002.

Gear C.W., I.G. Kevrekidis and C. Theodoropoulos, *'Coarse' integration/bifurcation analysis via microscopic simulators: Micro-Galerkin methods*. Computers and Chemical Engineering, **26**, 941-963, 2002.

Gear C.W., J. Li and I.G. Kevrekidis, *The gap-tooth method in particle simulations*. Physics Letters A, **316**, 190-195, 2003.

Gill P.E., W. Murray and M.H. Wright, *Practical optimization*, 1981. London: Academic Press.

Gillespie D.T., *A general method for numerically simulating the stochastic time evolution of coupled chemical*. Journal of Computational Physics, **22**, 403-434, 1976.

Gillespie D.T., *Exact stochastic simulation of coupled chemical reactions*. The Journal of Physical Chemistry, **81**, 1340-1361, 1977.

Gogoi T.K. and R. Das, *Inverse analysis of an internal reforming solid oxide fuel cell system using simplex search method*. Applied Mathematical Modelling, **37**, 6994-7015, 2013.

Gorban A.N., N. Kazantzis, I.G. Kevrekidis, H.C. Öttinger and C. Thedoropoulos, *Model reduction and coarse-graining approaches for multiscale phenomena*, 2006. Berlin-Heidelberg: Spinger.

Gorte R. and L.D. Schmidt, *Desorption kinetics with precursor intermediates*. Surface Science, **76**, 559-573, 1978.

Guerrero S. and E.E. Wolf, *Kinetic Monte Carlo simulation of the preferential oxidation of CO using normally distributed rate probabilities*. Chemical Engineering Science, **66**, 4477-4487, 2011.

Hammad A., S. Souentie, E.I. Papaioannou, S. Balomenou, D. Tsiplakides. J.C. Figueroa, C. Cavalca and C.J. Pereira, *Electrochemical promotion of the SO_2 oxidation over thin Pt films interfaced with YSZ in a monolithic electropromoted reactor*. Applied Catalysis B: Environmental, **103**, 336-342, 2011.

Hari B. and C. Theodoropoulos, *Integrated multi-scale models for simulation and design of microreactor systems*. Chemical Engineering Transactions, **17**, 1269-1274, 2009.

Hari B., *Multi-scale modelling, simulation, design and analysis of microreactors*. PhD Thesis, Chemical Engineering and Analytical Sciences, 2009, The University of Manchester.

Harkness I.R. and R.M. Lambert, *Electrochemical promotion of the NO+ethylene reaction over platinum*. Journal of Catalysis, **152**, 211-214, 1995.

Hegedus L.L., R. Aris, A.T. Bell, M. Boudart, N.Y. Chen, B.C. Gates, W.O. Haag, G.A. Somorjai and J. Wei, *Catalyst design: Progress and perspectives*, 1987. New York: John Wiley & Sons, Inc.

Heyne H. and F.C. Tompkins, *Application of infrared spectroscopy and surface potential measurements in a study of the oxidation of carbon monoxide on platinum*. Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences, **292**, 460-478, 1966.

Hinshelwood C.N., *Kinetics of chemical change in gaseous systems*, 1926. London: Oxford University Press.

Ho T.X., P. Kosinski, A.C. Hoffmann and A. Vik, *Modeling of transport, chemical and electrochemical phenomena in a cathode-supported SOFC*. Chemical Engineering Science, **64**, 3000-3009, 2009.

Hong J.K., I.-H. Oh, S.-A. Hong and W.Y. Lee, *Electrochemical oxidation of methanol over a silver electrode deposited on Yttria-Stabilized Zirconia electrolyte*. Journal of Catalysis, **163**, 95-105, 1996.

Hougen O.A. and K.M. Watson, *Chemical Process Principles*, Part III, 1947. New York: John Wiley & Sons, Inc.

Hougen O.A. and K.M. Watson, *Solid catalysts and reaction rates*. Industrial & Engineering Chemistry, **35**, 529-541, 1943.

Huang H., Z. Wang, H. Liu, H. Sun, Y. Wei and X. Zhou, *A kinetic model for analyzing partial oxidation reforming af heavy hydrocarbon over a novel self-sustained electrochemical promotion catalyst*. International Journal of Hydrogen Energy, **37**, 15125-15134, 2012.

Huang K. and J.B. Goodenough, *Solid oxide fuel cell technology: Principles, performance and operations*, 2009. Cambridge: Woodhead Publishing.

Ibrahim N., D. Poulidi and I.S. Metcalfe, *The role of sodium surface species on electrochemical promotion of catalysis in a Pt/YSZ system: The case of ethylene oxidation*. Journal of Catalysis, **303**, 100-109, 2013.

Ingram G.D., I.T. Cameron and K.M. Hangos, *Classification and analysis of integrating frameworks in multiscale modelling*. Chemical Engineering Science, **59**, 2171-2187, 2004.

Jaccoud A., *Electrochemical promotion of Pt catalysts for gas phase reactions*. PhD Thesis, Chemistry and Chemical Engineering, 2007, École Polytechnique Fédérale de Lausanne.

Jaksic J.M., D. Labou, C.M. Lachjevac, A. Siokou and M.M. Jaksic, *Potentiodynamic estimation of key parametric criterions and interrelating reversible spillover effects for electrochemical promotion*. Applied Catalysis A: General, **380**, 1-14, 2010.

Jensen I., H.C. Fogedby and R. Dickman, *Critical exponents for an irreversible surface reaction model*. Physical Review A, **41**, 3411-3414, 1990.

Jiménez-Borja C., B. Delgado, L.F. Díaz-Díaz, J.L. Valverde and F. Dorado, *Enhancing the combustion of natural gas by electrochemical promotion of catalysis*. Electrochemistry Communications, **23**, 9-12, 2012.

Kaloyannis A. and C.G. Vayenas, *Non-Faradaic electrochemical modification of catalytic activity: 12. Propylene oxidation on Pt*. Journal of Catalysis, **182**, 37-47, 1999.

Karasali H., *Electrochemical promotion of CO oxidation and CO, CO_2 hydrogenation on transition metals*. PhD Thesis, Department of Chemical Engineering, 1994, University of Patras.

Karavasilis C., S. Bebelis and C.G. Vayenas, *In Situ controlled promotion of catalyst surfaces via NEMCA: The effect of Na on the Ag-catalyzed ethylene epoxidation in the presence of chlorine moderators*. Journal of Catalysis, **160**, 205-213, 1996.

Katsaounis A., *Electrochemical promotion of catalysis (EPOC) perspectives for application to gas emissions treatment*. Global NEST Journal, **10**, 226-236, 2008.

Katsoulakis M.A., A.J. Majda and D.G. Vlachos, *Coarsed-grained stochastic processes and Monte Carlo simulations in lattice systems*. Journal of Computational Physics, **186**, 250-278, 2003.

Kaul D.J., R. Sant and E.E. Wicke, *Integrated kinetic modeling and transient FTIR studies of CO oxidation on Pt/SiO2*. Chemical Engineering Science, **42**, 1399-1411, 1987.

Kaul D.J. and E.E. Wolf, *Fourier transform infrared spectroscopy studies of surface reaction dynamics: II. Surface coverage and inhomogeneous temperature patterns of self-sustained oscillations during CO oxidation on* Pt/SiO_2. Journal of Catalysis, **91**, 216-230, 1985.

Kevrekidis I.G., C.W. Gear, J.M. Hyman, P.G. Kevrekidis, O. Runborg and C. Theodoropoulos, *Equation-free, coarse-grained multiscale computation: Enabling microscopic simulators to perform system-level analysis*. Communications in Mathematical Sciences, **1**, 715-762, 2003.

Khaleel M.A., D.R. Rector, Z. Lin, K. Johnson and K. Rechnagle, *Multiscale electrochemistry modeling of solid oxide fuel cells*. International Journal for Multiscale Computational Engineering, **3**, 33-48, 2005.

Kim J.H., W.K. Liu and C. Lee, *Multi-scale solid oxide fuel cell materials modeling*. Computational Mechanics, **44**, 683-703, 2009.

Kirkpatrick S., C.D. Gelatt and M.P. Vecchi, *Optimization by Simulated Annealing*. Science, **220**, 671-680, 1983.

Kiskinova M.P., *Poisoning and promotion in catalysis based on surface science concepts an experiments*, in: Studies in surface science and catalysis, 1982. Amsterdam: Elsevier Science Publishers B.V.

Koutsodontis C., A. Katsaounis, J.C. Figueroa, C. Cavalca, C. Pereira and C.G. Vayenas, *The effect of catalyst film thickness on the electrochemical promotion of ethylene oxidation on Pt.* Topics in Catalysis, **39**, 97-100, 2006.

Kwan T., *Power rate law in heterogeneous catalysts*. The Journal of Physical Chemistry, **60**, 1033-1037, 1956.

Ladas S., S. Bebelis and C.G. Vayenas, *Work function measurements on catalyst films subject to in situ electrochemical promotion*. Surface Science, **251-252**, 1062-1068, 1991.

Lambert R.M., A. Palermo, F.J. Williams and M.S. Tikhon, *Electrochemical promotion of catalytic reactions using alkali ion conductors*. Solid State Ionics, **136-137**, 677-685, 2000.

Langmuir I., *The adsorption of gases on plane surfaces of glass, mica and platinum*. Journal of the American Chemical Society, **40**, 1361-1403, 1918.

Langmuir I., *The mechanism of the catalytic action of platinum in the reactions* $2CO+O_2=2CO_2$ *and* $2H_2+O_2=2H_2O$. Transactions of the Faraday Society, **17**, 621-654, 1922.

Leah R.T, N.P. Brandon and P. Aguiar, *Modelling of cells, stacks and systems based around metal-supported planar IT-SOFC cells with CGO electrolytes operating at 500-600 °C.* Journal of Power Sources, **145**, 336-352, 2005.

Lee S.F. and C.W. Hong, *Multi-scale design simulation of a novel intermediate-temperature micro solid oxide fuel cell stack system*. International Journal of Hydrogen Energy, **35**, 1330-1338, 2010.

Leiva E.P.M., C. Vazquez, M.I. Rojas and M.M. Mariscal, *Computer simulation of the effective double layer occurring on a catalyst surface under electro-chemical promotion conditions*. Journal of Applied Electrochemistry, **38**, 1065-1073, 2008.

Majumber D. and L.J. Broadbelt, *A multiscale scheme for modeling catalytic flow reactors*. AIChE Journal, **52**, 4214-4228, 2006.

Makeev A.G., D. Maroudas, A.Z. Panagiotopoulos and I.G. Kevrekidis, *Coarse bifurcation analysis of kinetic Monte Carlo simulations: A lattice-gas model with lateral interactions*. Journal of Chemical Physics, **117**, 8229-8240, 2002.

Makri M., A. Buekenhoudt, J. Luyten and C.G. Vayenas, *Non-Faradaic electrochemical modification of the catalytic activity of Pt using a* $CaZr_{0.9}In_{0.1}O_{3.\alpha}$ *proton conductor*. Ionics, **2**, 282–288, 1996.

Maroudas D., *Multiscale modeling of hard materials: Challenges and opportunities for chemical engineering*. AIChE Journal, **46**, 878-882, 2000.

Marwood M. and C.G. Vayenas, *Electrochemical promotion of a dispersed platinum catalyst*. Journal of Catalysis, **178**, 429-440, 1998.

Marwood M., A. Kaloyannis and C.G. Vayenas, *Electrochemical promotion of the NO reduction by C_2H_4 on Pt/YSZ and by CO on Pd/YSZ*. Ionics, **2**, 302-311, 1996.

Matei F., C. Jiménez-Borja, J. Canales-Vázquez, S. Brosda, F. Dorado, J.L. Valverde and D. Ciuparu, *Enhanced electropromotion of methane combustion on palladium catalysts deposited on highly porous supports*. Applied Catalysis B: Environmental, **132-133**, 80-89, 2013.

MATLAB, *MATLAB® & SIMULINK® Release notes for R2007a*, 2007. The MathWorks, Inc.

Matsushima T., *Tracer studies on the reaction paths of the CO oxidation over platinum*. Journal of Catalysis, **55**, 337-347, 1978.

McGreevy R.L. and L. Pusztai, *Reverse Monte Carlo simulation: A new technique for the determination of disordered structures*. Molecular Simulation, **1**, 359-367, 1988.

Metcalfe I.S., *Electrochemical promotion of catalysis: I: Thermodynamic considerations*. Journal of Catalysis, **199**, 247-258, 2001a.

Metcalfe I.S., *Electrochemical promotion of catalysis: II: The role of a stable spillover species and prediction of reaction rate modification*. Journal of Catalysis, **199**, 259-272, 2001b.

Metropolis N. and S. Ulam, *The Monte Carlo method*. Journal of the American Statistical Association, **44**, 335-341, 1949.

Michaels J.N. and C.G. Vayenas, *Kinetics of vapor-phase oxidative dehydrogenation of ethylbenzene.* Journal of Catalysis, **85**, 477-487, 1984.

Mutoro E., C. Koutsodontis, B. Luerssen, S. Brosda, C.G. Vayenas and J. Janek, *Electrochemical promotion of Pt(111)/YSZ(111) and Pt-FeO_x/YSZ(111) thin catalyst films: Electrocatalytic, catalytic and morphological studies*. Applied Catalysis B: Environmental, **100**, 328-337, 2010.

Myshlyavtsev A.V. and V.P. Zhdanov, *The effect of nearest-neighbour and next-nearest-neigbour lateral interactions on thermal-desorption spectra*. Chemical Physics Letters, **162**, 43-46, 1989.

Neophytides S., D. Tsiplakides, P. Stonehart, M. Jaksic and C.G. Vayenas, *Electrochemical enhancement of a catalytic reaction in aqueous solution*. Nature, **370**, 45-47, 1994.

Nicole J. and C. Comninellis, *Electrochemical promotion of IrO_2 catalyst activity for the gas-phase combustion of ethylene*. Journal of Applied Electrochemistry, **28**, 223-226, 1998.

Nicole J., D. Tsiplakides, S. Wodiunig and C. Comninellis, *Activation of catalyst for gas-phase combustion by electrochemical pretreatment*. The Journal of the Electrochemical Society, **145**, L312-314, 1997.

Otten R.H.J.M. and L.P.P.P. van Ginneken, *The annealing algorithm*, 1989. Boston, MA: Kluwer.

Pacheco P.S., *A user's guide to MPI*, 1998. San Francisco: Morgan Kaufmann Publishers, Inc.

Palermo A., R.M. Lambert, I.R. Harkness, I.V. Yentekakis, O. Mar'ina and C.G. Vayenas, *Electrochemical promotion by Na of the platinum-catalyzed reaction between CO and NO*. Journal of Catalysis, **161**, 471-479, 1996.

Palmer R.L. and J.N. Smith, *Molecular beam study of CO oxidation on a (111) platinum surface*. Journal of Chemical Physics, **60**, 1453-1463, 1974.

Pantelides C., *New challenges and opportunities for process modelling*. Computer Aided Process Engineering, **11**, 15-26, 2001.

Perry J.H., R.H. Perry, C.H. Chilton and S.D. Kirkpatrick, *Chemical Engineers' Handbook*. 4th ed., 1963. New York: John Wiley & Sons, Inc.

Petrolekas P.D., S. Balomenou and C.G. Vayenas, *Electrochemical promotion of ethylene oxidation on Pt catalyst films deposited on CeO_2*. The Journal of the Electrochemical Society, **145**, 1202-1206, 1998.

Phillips R., *Crystals, defects and microstructures: Modeling across the scales*, 1998. Cambridge University Press.

Pitselis G., P. Petrolekas and C.G. Vayenas, *Electrochemical promotion of ammonia decomposition over Fe catalyst film interfaced with K^+- & H^+- conductors*. Ionics, **3**, 110-116, 1997.

Pliangos C., C. Raptis, T. Badas, D. Tsiplakides and C.G. Vayenas, *Electrochemical promotion of a classically promoted Rh catalyst for the reduction of NO*. Electrochimica Acta, **46**, 331-339, 2000.

Poling B.E., J.M. Prausnitz, R.C. Reid and J.P. O'Connell, *The properties of gases and liquids*, 5th ed. 2001. New York: McGraw-Hill, Inc.

Politova T.I., V.V. Gal'vita, V.D. Belyaev and V.A. Sobyanin, *Non-Faradaic catalysis: The case of CO oxidation over Ag-Pd alloy electrode in a solid oxide electrolyte cell*. Catalysis Letters, **44**, 75-81, 1997.

Poppe J., S. Völkening, A. Schaak, E. Schürtz, J. Janek and R. Imbihl, *Electrochemical promotion of catalytic CO oxidation on Pt/YSZ catalysts under low pressure conditions*. Physical Chemistry Chemical Physics, **1**, 5241-5249 , 1999.

Poulidi D., G.C. Mather and I.S. Metcalfe, *Wireless electrochemical modification of catalytic activity on a mixed protonic-electronic conductor*. Solid State Ionics, **178**, 675-680, 2007.

Poulidi D., M.E. Rivas and I.S. Metcalfe, *Controlled spillover in a single catalyst pellet: Rate modification, mechanism and relationship with electrochemical promotion*. Journal of Catalysis, **281**, 188-197, 2011.

Presvytes D. and C.G. Vayenas, *Mathematical modeling of the operation of SOFC Nickel-cermet anodes*. Ionics, **13**, 9-18, 2007.

Pritchard J., *Electrochemical Promotion*. Nature, **343**, 592-593, 1990.

Raimondeau S. and D.G. Vlachos, *Recent developments on multiscale, hierarchical modeling of chemical reactors*. Chemical Engineering Journal, **90**, 3-23, 2002a.

Raimondeau S. and D.G. Vlachos, *The role of adsordate-layer nonuniformities in catalytic reactor design: Multiscale simulations for CO oxidation on Pt*. Computers and Chemical Engineering, **26**, 965-980, 2002b.

Reese J.S., S. Raimondeau and D.G. Vlachos, *Monte Carlo algorithms for complex surface reaction mechanisms: Efficiency and accuracy*. Journal of Computational Physics, **173**, 302-321, 2001.

Rideal E.K., *A note on a simple molecular mechanism for heterogeneous catalytic reactions*. Mathematical Proceedings of the Cambridge Philosophical Society, **35**, 130-132, 1939.

Ruiz E., D. Cillero, P.J. Martínez, Á. Morales, G.S. Vicente, G. de Diego and J.M. Sánchez, *Bench scale study of electrochemically promoted catalytic CO_2 hydrogenation to renewable fuels*. Catalysis Today, **210**, 55-66, 2013.

Sahibzada M., W. Morton, A. Hartley, D. Mantzavinos and I.S. Metcalfe, *A simple method for the determination of surface exchange and ionic transport kinetics in oxides*. Solid State Ionics, **136-137**, 991-996, 2000.

Salciccioli M., M. Stamatakis, S. Caratzoulas and D.G. Vlachos, *A review of multiscale modeling of metal-catalysed reactions: Mechanism development for complexity and emergent behavior*. Chemical Engineering Science, **66**, 4319-4355, 2011.

Sapountzi F.M., M.N. Tsampas and C.G. Vayenas, *Electrochemical promotion of CO conversion to CO_2 in PEM fuel cell PROX reactor*. Catalysis Today, **146**, 319-325, 2009.

Schaefer C. and A.P.J. Jansen, *Coupling of kinetic Monte Carlo simulations of surface reactions to transport in a fluid for heterogeneous catalytic reactor modeling*. Journal of Chemical Physics, **138**, 054102-1-9, 2013.

Schommers W., *Pair potentials in disordered many-particle systems: A study for liquid gallium*. Physical Review A, **28**, 3599-3605, 1983.

Siettos C.I., A. Armaou, A.G. Makeev and I.G. Kevrekidis, *Microscopic/stochastic timesteppers and coarse control: A kMC example*. AIChE Journal, **49**, 1922-1926, 2003.

Silverberg M., A. Ben-Shaul and F. Rebentrost, *On the effects of adsorbate aggregation on the kinetics of surface reactions*. The Journal of Chemical Physics, **83**, 6501-6513, 1985.

Stoukides M. and C.G. Vayenas, *The effect of electrochemical oxygen pumping on the rate and selectivity of ethylene oxidation on polycrystalline silver*. Journal of Catalysis, **70**, 137-146, 1981.

Suzuki M., N. Shikazono, K. Fukagata and N. Kasagi, *Numerical analysis of coupled transport and reaction phenomena in an anode-supported flat-tube solid oxide fuel cell*. Journal of Power Sources, **180**, 29-40, 2008.

Szymczak P. and A.J.C. Ladd, *Stochastic boundary conditions to the convection-diffusion equation including chemical reactions at solid surfaces*. Physical Review E, **69**, 036704-1-9, 2004.

Tello J.S. and W.A. Curtin, *A coupled discrete/continuum model for multiscale diffusion*. International Journal for Multiscale Computational Engineering, **3**, 257-265, 2005.

Theodoropoulos C., *Optimisation and linear control of large scale nonlinear systems: A review and a suite of model reduction-based techniques,* In: Gorban A.N. and D. Roose, ed. Coping with complexity: Model reduction and data analysis. Lecture Notes in Computational Science and Engineering, **75**, 37-61, 2011.

Theodoropoulos C., Y.-H. Qian and I.G. Kevrekidis, *"Coarse" stability and bifurcation analysis using time-steppers: A reaction-diffusion example*. Proceedings of the National Academy of Sciences of the United States of America, **97**, 9840-9843, 2000.

Tseronis K., I. Bonis, I.K. Kookos and C. Theodoropoulos, *Parametric and transient analysis of non-isothermal, planar solid oxide fuel cells*. International Journal of Hydrogen Energy, **37**, 530-547, 2012.

Tseronis K., I.K. Kookos and C. Theodoropoulos, *Modelling mass transport in solid oxide fuel cell anodes: A case for a multidimensional dusty gas-based model*. Chemical Engineering Science, **63**, 5626-5638, 2008.

Tseronis K., *Modelling and design of solid oxide fuel cell systems*. PhD Thesis, Chemical Engineering and Analytical Sciences, 2009, The University of Manchester.

Tsiplakides D., S. Balomenou, A. Katsaounis, D. Archonta, C. Koutsodontis and C.G. Vayenas, *Electrochemical promotion of catalysis: Mechanistic investigations and monolithic electropromoted reactors*. Catalysis Today, **100**, 133-144, 2005.

Varkaraki E., J. Nicole, E. Plattner, C. Comninellis and C.G. Vayenas, *Electrochemical promotion of IrO_2 catalyst for the gas-phase combustion of ethylene*. Journal of Applied Electrochemistry, **25**, 978-981, 1995.

Vayenas C.G. and G.E. Pitselis, *Mathematical modeling of electrochemical promotion and of metal-support interactions*. Industrial and Engineering Chemistry Research, **40**, 4209-4215, 2001.

Vayenas C.G. and S. Bebelis, *Electrochemical promotion of heterogeneous catalysis*. Catalysis Today, **51**, 581-594, 1999.

Vayenas C.G. and S.I. Bebelis, *Electrochemical Promotion*. Solid State Ionics, **94**, 267-277, 1997.

Vayenas C.G., D. Archonta and D. Tsiplakides, *Scanning tunneling microscopy observation of the origin of electrochemical promotion and metal-support interactions*. Journal of Electroanalytical Chemistry, **554-555**, 301-306, 2003.

Vayenas C.G., M.M. Jaksic, S. Bebelis and S.G. Neophytides, *The electrochemical activation of catalysis*. Modern Aspects of Electrochemistry, **29**, 57-202, 1995.

Vayenas C.G., *Promotion, electrochemical promotion and metal-support interactions: Their common features*. Catalysis Letters, **143**, 1085-1097, 2013.

Vayenas C.G., S. Bebelis and M. Despotopoulou, *Non-Faradaic electrochemical modification of catalytic activity: 4. The use of β''-Al_2O_3 as the solid electrolyte*. Journal of Catalysis, **128**, 415-435, 1991.

Vayenas C.G., S. Bebelis and S. Ladas, *Dependence of catalytic rates on catalyst work function*. Nature, **343**, 625-627, 1990a.

Vayenas C.G., S. Bebelis and S. Neophytides, *Non-Faradaic electrochemical modification of catalytic activity*. The Journal of Physical Chemistry, **92**, 5083-5085, 1988.

Vayenas C.G., S. Bebelis, C. Pliangos, S. Brosda and D. Tsiplakides, *Electrochemical activation of catalysis: Promotion, electrochemical promotion, and metal-support interactions*, 2001a. New York: Kluwer Academic/Plenum Publishers.

Vayenas C.G., S. Bebelis, I.V. Yentekakis and H.-G. Lintz, *Non-Faradaic electrochemical modification of catalytic activity: A status report*. Catalysis Today, **11**, 303-442, 1992.

Vayenas C.G., S. Bebelis, I.V. Yentekakis, P. Tsiakaras and H. Karasali, *Non-Faradaic electrochemical modification of catalytic activity on Pt metals*. Platinum Metals Review, **34**, 122-130, 1990b.

Vayenas C.G., S. Bebelis, I.V. Yentekakis, S. Neophytides and J. Yi, *Ion spillover as the origin of the NEMCA effect.* Studies in Surface Science and Catalysis, **77**, 111-116, 1993.

Vayenas C.G., S. Bebelis, S. Neophytides and I.V. Yentekakis, *Non-Faradaic electrochemical modification of catalytic activity in solid electrolyte cells*. Applied Physics A: Materials Science and Processing, **49**, 95-103, 1989.

Vayenas C.G., S. Brosda and C. Pliangos, *Rules and mathematical modeling of electrochemical and chemical reaction: 1. Reaction classification and promotional rules*, Journal of Catalysis, **203**, 329-350, 2001b.

Vayenas C.G., S. Ladas, S. Bebelis, I.V. Yentekakis, S. Neophytides, J. Yi, C. Karavasilis and C. Pliangos, *Electrochemical promotion in catalysis: Non-*

Faradaic electrochemical modification of catalytic activity. Electrochimica Acta, **39**, 1849-1855, 1994.

Vernoux P., F. Gaillard, R. Karoum and A. Billard, *Reduction of nitrogen oxides over Ir/YSZ electrochemical catalysts*. Applied Catalysis B: Environmental, **73**, 73-83, 2007.

Vogler M., A. Bieberle-Hutter, L. Gauckler, J. Warnatz and W.G. Bessler, *Modeling study of surface reactions, diffusion, and spillover at a Ni/YSZ patterned anode*. Journal of The Electrochemical Society, **156**, B663-B672, 2009.

Wang G., Y. Yang, H. Zhang and W. Xia, *3-D model of thermo-fluid and electrochemical for planar SOFC*. Journal of Power Sources, **167**, 398-405, 2007.

Weinan E., *Principles of multiscale modeling*, 2011. Cambridge University Press.

Winter M. and R.J. Brodd, *What are batteries, fuel cells, and supercapacitors?* Chemical Reviews, **104**, 4245-4269, 2004.

Wodiunig S. and C. Comninellis, *Electrochemical promotion of RuO_2 catalysts for the gas phase combustion of C_2H_4*. Journal of European Ceramic Society, **19**, 931-934, 1999.

Xia C., C. Falgairette, Y. Li, G. Foti, C. Comninellis and W. Harbich, *Electrochemical promotion of CO combustion over Pt/YSZ under high vacuum conditions*. Applied Catalysis B: Environmental, **113-114**, 250-254, 2012.

Xia C., M. Hugentobler, Y. Li, G. Foti, C. Comninellis and W. Harbich, *Electrochemical promotion of CO combustion over non-percolated Pt particles supported on YSZ using a novel bipolar configuration*. Electrochemistry Communications, **13**, 99-101, 2010.

Yentekakis I.V. and C.G. Vayenas, *In situ controlled promotion of Pt for CO oxidation via NEMCA using CaF_2, as the solid electrolyte*. Journal of Catalysis, **149**, 238–242, 1994.

Yentekakis I.V. and C.G. Vayenas, *The effect of electrochemical oxygen pumping on the steady-state and oscillatory behavior of CO oxidation on polycrystalline Pt*. Journal of Catalysis, **111**, 170-188, 1988.

Yentekakis I.V. and S. Bebelis, *Study of the NEMCA effect in a single-pellet catalytic reactor*. Journal of Catalysis, **137**, 278-283, 1992.

Yentekakis I.V., G. Moggridge, C.G. Vayenas and R.M. Lambert, *In situ controlled promotion of catalyst surfaces via NEMCA: The effect of Na on Pt-catalyzed CO oxidation*. Journal of Catalysis, **146**, 292-305, 1994.

Yentekakis I.V., S. Neophytides and C.G. Vayenas, *Solid electrolyte aided study of the mechanism of CO oxidation on polycrystalline platinum*. Journal of Catalysis, **111**, 152-169, 1988.

Yiokari C.G., G.E. Pitselis, D.G. Polydoros, A.D. Katsaounis and C.G. Vayenas, *High-pressure electrochemical promotion of ammonia synthesis over an industrial iron catalyst*. The Journal of Physical Chemistry A, **104**, 10600-10602, 2000.

Yoshida J., K. Kataoka, R. Horcajada and A. Nagaki, *Modern strategies in electroorganic synthesis*. Chemical Reviews, **108**, 2265-2299, 2008.

Zhdanov V.P. and B. Kasemo, *Kinetic oscillations on nm-sized catalyst particles: Oxide model*. Surface Science, **511**, 23-33, 2002.

Zhdanov V.P. and B. Kasemo, *Kinetic phase transitions in simple reactions on solid surfaces*. Surface Science Reports, **20**, 113-189, 1994.

Zhdanov V.P. and B. Kasemo, *Simulation of kinetic oscillations in the $CO+O_2$/Pt reaction on the nm scale*. Journal of Catalysis, **214**, 121-129, 2003.

Ziff R.M., E. Gulari and Y. Barshad, *Kinetic phase transitions in an irreversible surface-reaction model*. Physical Review Letters, **56**, 2553-2556, 1986.

APPENDIX

Source Codes

A.1 COMSOL scripts (Chapter 3)

```
function [fem0 r_NEMCA r_Faradaic r_Total]=Macroscopic_2D(Tin,Po2_inlet,Pco_inlet,Fd)

% COMSOL Multiphysics Model M-file
% Generated by COMSOL 3.5a (COMSOL 3.5.0.603, $Date: 2008/12/03 17:02:19 $)

flclear fem

% COMSOL version
clear vrsn
vrsn.name = 'COMSOL 3.5';
vrsn.ext = 'a';
vrsn.major = 0;
vrsn.build = 603;
vrsn.rcs = '$Name:  $';
vrsn.date = '$Date: 2008/12/03 17:02:19 $';
fem.version = vrsn;

% Geometry
g1=rect2('1e-2','400e-6','base','corner','pos',{'0','0'},'rot','0');
parr={point2(0.0060,0)};
g2=geomcoerce('point',parr);
parr={point2(0.0095,0)};
g3=geomcoerce('point',parr);
parr={point2(0.00975,0)};
g4=geomcoerce('point',parr);
parr={point2(0.0050,4.0E-4)};
g5=geomcoerce('point',parr);
parr={point2(0.0030,0)};
g6=geomcoerce('point',parr);

% Analyzed geometry
clear p s
p.objs={g3,g4,g5,g2,g6};
p.name={'PT1','PT2','PT3','PT4','PT5'};
p.tags={'g3','g4','g5','g2','g6'};

s.objs={g1};
s.name={'R1'};
s.tags={'g1'};

fem.draw=struct('p',p,'s',s);
fem.geom=geomcsg(fem);

% Initialize mesh
fem.mesh=meshinit(fem, ...
                  'hauto',5, ...
```

```
                    'hmaxvtx',[3,1e-5,4,1e-5], ...
                    'hmaxedg',[2,3e-5,3,3e-5,4,3e-5,5,3e-5], ...
                    'hmaxsub',[1,7e-5]);

% Refine mesh
fem.mesh=meshrefine(fem, ...
                     'mcase',0, ...
                     'rmethod','regular');

% Constants
fem.const = {'T','645.15 [K]', ...
  'F','96485 [s*A/mol]', ...
  'R','8.3145 [J/mol/K]', ...
  'gamma_c','6.91e8 [A/m^2]', ...
  'gamma_a1','1e+010[A/m^2]', ...
  'A_a2','4e+04 [A/m^2]', ...
  'gamma_a3','6e+08 [A/m^2]', ...
  'Pref','101325 [Pa]', ...
  'Eact_c','110e3 [J/mol]', ...
  'Eact_a','120e3 [J/mol]', ...
  'a_c','0.5 [1]', ...
  'a_a','0.5 [1]', ...
  'D_O','0 [m^2/s]', ...
  'D_CO','0 [m^2/s]', ...
  'D_BSS','4e-15 [m^2/s]', ...
  'Ns','2.5407e-5 [mol/m^2]', ...
  'M_o2','32e-3 [kg/mol]', ...
  'M_CO','28e-3 [kg/mol]', ...
  'S_o2','7.69e-05 [1]', ...
  'S_CO','5.38e-01 [1]', ...
  'ko_1','2.4e13 [1/s]', ...
  'E_1','2.43139e+05 [J/mol]', ...
  'ko_2','6.5e13 [1/s]', ...
  'E_2','9.9618e+04 [J/mol]', ...
  'ko3','2.7e6 [1/s]', ...
  'E3','3.5186e+04 [J/mol]', ...
  'mu_co_0','-137.3e3 [J/mol]', ...
  'mu_co2_0','-394.4e3 [J/mol]', ...
  'mu_BSS','-236.4e3 [J/mol]', ...
  'width','2e-2 [m]', ...
  'Po2_in','5.8e3 [Pa]', ...
  'Pco_in','3.511e3 [Pa]', ...
  'Feed', Fd, ...
  'k4','5.73e-03 [1/s]'};

% Application mode 1
clear appl
appl.mode.class = 'FlPDEC';
appl.dim = {'Vio','Vio_t'};
appl.name = 'Velectrolyte';
appl.assignsuffix = '_Velectrolyte';
clear bnd
bnd.g = {0,'Janode','-Jcathode'};
bnd.type = 'neu';
bnd.ind = [1,3,2,3,2,1,1,1,1];
appl.bnd = bnd;
clear equ
equ.f = 0;
equ.da = 0;
equ.c = 'sigma_ionic';
equ.ind = [1];
appl.equ = equ;
fem.appl{1} = appl;

% Application mode 2
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'Vca','Vca_t'};
appl.name = 'Cathode_TPB_flux';
appl.assignsuffix = '_Cathode_TPB_flux';
clear prop
```

```
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm3','lm4'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear bnd
bnd.usage = {0,1};
bnd.weak = {0,xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'};
bnd.ind = [1,2,1,2,1,1,1,1,1];
appl.bnd = bnd;
fem.appl{2} = appl;

% Application mode 3
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'Vca','Vca_t2'};
appl.name = 'Vcathode';
appl.assignsuffix = '_Vcathode';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm5','lm6'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear pnt
pnt.constrf = {0,'test(Vca_0-Vca)'};
pnt.constr = {0,'Vca_0-Vca'};
pnt.ind = [1,1,2,1,1,1,2,1,1];
appl.pnt = pnt;
clear bnd
bnd.usage = {0,1};
bnd.weak = {0,'sigma_el_ca*(-VcaTx_test*VcaTx-VcaTy_test*VcaTy)'};
bnd.ind = [1,2,1,2,1,1,1,1,1];
appl.bnd = bnd;
fem.appl{3} = appl;

% Application mode 4
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'Van','Van_t'};
appl.name = 'Anode_TPB_flux';
appl.assignsuffix = '_Anode_TPB_flux';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm7','lm8'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear bnd
bnd.usage = {0,1};
bnd.weak = {0,'sigma_el_an*(-VanTx_test*Van-VanTy_test*Van)-Van_test*Janode'};
bnd.ind = [1,1,2,1,2,1,1,1,1];
appl.bnd = bnd;
fem.appl{4} = appl;

% Application mode 5
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'Van','Van_t2'};
appl.name = 'Vanode';
appl.assignsuffix = '_Vanode';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm9','lm10'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear pnt
pnt.constrf = {0,'test(Van_0-Van)'};
pnt.constr = {0,'Van_0-Van'};
pnt.ind = [1,1,1,2,1,1,1,1,1];
```

```
appl.pnt = pnt;
clear bnd
bnd.usage = {0,1};
bnd.weak = {0,'sigma_el_an*(-VanTx_test*VanTx-VanTy_test*VanTy)'};
bnd.ind = [1,1,2,1,2,1,1,1,1];
appl.bnd = bnd;
fem.appl{5} = appl;


% Application mode 6
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'theta_O','theta_O_t'};
appl.name = 'Oxygen_Coverage';
appl.assignsuffix = '_Oxygen_Coverage';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm11','lm12'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear bnd
bnd.usage = {0,1};
bnd.weak = {0,'D_O*(-theta_OTx_test*theta_OTx-
theta_OTy_test*theta_OTy)+theta_O_test*r_O'};
bnd.ind = [1,1,2,1,2,1,1,1,1];
appl.bnd = bnd;
fem.appl{6} = appl;

% Application mode 7
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'theta_CO','theta_CO_t'};
appl.name = 'CO_Coverage';
appl.assignsuffix = '_CO_Coverage';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm13','lm14'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear bnd
bnd.usage = {0,1};
bnd.weak = {0,'D_CO*(-theta_COTx_test*theta_COTx-
theta_COTy_test*theta_COTy)+theta_CO_test*r_CO'};
bnd.ind = [1,1,2,1,2,1,1,1,1];
appl.bnd = bnd;
fem.appl{7} = appl;

% Application mode 8
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'theta_BSS','theta_BSS_t2'};
appl.name = 'BSS_Coverage';
appl.assignsuffix = '_BSS_Coverage';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm17','lm18'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear bnd
bnd.usage = {0,1};
bnd.weak = {0,'D_BSS*(-theta_BSSTx_test*theta_BSSTx-
theta_BSSTy_test*theta_BSSTy)+theta_BSS_test*(r_BSS+abs(Jan_BSS)/2/F/Ns*(1-theta_O-
theta_CO-theta_BSS))'};
bnd.ind = [1,1,2,1,2,1,1,1,1];
appl.bnd = bnd;
fem.appl{8} = appl;
fem.frame = {'ref'};
fem.border = 1;
```

```
clear units;
units.basesystem = 'SI';
fem.units = units;

% Subdomain settings
clear equ
equ.ind = [1];
equ.dim = {'Vio'};

% Subdomain expressions
equ.expr = {'sigma_ionic','33400[S/m]*exp((-10300)[K]/T)'};
fem.equ = equ;

% Boundary settings
clear bnd
bnd.ind = [1,2,3,2,3,1,1,1,1];
bnd.dim = {'Vio','Vca','Vca','Van','Van','theta_O','theta_CO', ...
  'theta_BSS'};

% Boundary expressions
bnd.expr = {'sigma_el_ca',{'','1/(-1.145028e-09+1.275961e-17*T^3-4.720065e-
16*T^2+7.877040999999999e-11*T)', ...
  ''}, ...
  'sigma_el_an',{'','','1/(-4.843579e-08+2.814022e-17*T^3-1.600249e-13*T^2+5.552497e-
10*T)'}};
fem.bnd = bnd;

% Point settings
clear pnt
pnt.ind = [1,1,2,3,1,1,1,1,1];

% Point expressions
pnt.expr = {'Van_0',{'','','0[V]'}, ...
  'Vca_0',{'','0.7[V]',''}};
fem.pnt = pnt;

% Coupling variable elements
clear elemcpl

% Integration coupling variables
clear elem
elem.elem = 'elcplscalar';
elem.g = {'1'};
src = cell(1,1);
clear bnd
bnd.expr = {{{},xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx};
bnd.ipoints = {{{},'4',{},'4'},{{},{},'4',{}},{{},{},'4',{}},{{},{},'4',{}},{{}, ...
  {},'4',{}},{{},{},'4',{}},{{},{},'4',{}}};
bnd.frame = {{{},'ref',{},'ref'},{{},{},'ref',{}},{{},{},'ref',{}},{{},{}, ...
  'ref',{}},{{},{},'ref',{}},{{},{},'ref',{}},{{},{},'ref',{}}};
bnd.ind = {{'1','6','9'},{'2','4'},{'3','5'},{'7','8'}};
src{1} = {{},bnd,{}};
elem.src = src;
geomdim = cell(1,1);
geomdim{1} = {};
elem.geomdim = geomdim;
elem.var = {'Int1','Int2','Int3','Int4','Int5','Int6','Int7'};
elem.global = {'1','2','3','4','5','6','7'};
elem.maxvars = {};
elemcpl{1} = elem;
fem.elemcpl = elemcpl;

% Global expressions
fem.globalexpr = {'Voc','(0.5*(mu_co_0-mu_co2_0-
mu_BSS)+0.5*R*T*log(Po2^(3/2)/Po2^(1/2))+0.5*R*T*log(Pco/Pco2))/(1*F)', ...
  'n_c','Voc-(Vca-Vio)', ...
  'Jo_c','gamma_c*(Po2/Pref)^(0.25)*exp(-Eact_c/(R*T))', ...
  'Jcathode','3*Jo_c*(exp(a_c*2*F*n_c/(R*T))-exp(-(1-a_c)*2*F*n_c/(R*T)))', ...
  'n_a','Van-Vio', ...
  'Jo_a1','gamma_a1*Pco/Pref*Pco2/Pref*exp(-Eact_a/(R*T))', ...
  'Jo_a2','A_a2*exp(-Eact_a/(R*T))', ...
```

```
  'Jo_a3','gamma_a3*(Po2/Pref)^(-0.25)*exp(-Eact_a/(R*T))', ...
  'Jan_co2','Jo_a1*(exp(a_a*2*F*n_a/(R*T))-exp(-(1-a_a)*2*F*n_a/(R*T)))', ...
  'Jan_BSS','Jo_a2*(exp(a_a*2*F*n_a/(R*T))-exp(-(1-a_a)*2*F*n_a/(R*T)))', ...
  'Jan_o2','Jo_a3*(exp(a_a*2*F*n_a/(R*T))-exp(-(1-a_a)*2*F*n_a/(R*T)))', ...
  'Janode','Jan_co2+Jan_BSS+Jan_o2', ...
  'c_o2','Po2/(R*T)', ...
  'c_CO','Pco/(R*T)', ...
  'k1','S_o2*sqrt(0.5*R*T/(pi*M_o2))/Ns', ...
  'k_1','ko_1*exp(-E_1/(R*T))', ...
  'k2','S_CO*sqrt(0.5*R*T/(pi*M_CO))/Ns', ...
  'k_2','ko_2*exp(-E_2/(R*T))', ...
  'k3','ko3*exp(-E3/(R*T))', ...
  'k5','1e-2-k4', ...
  'r1','2*k1*c_o2*(1-theta_O-theta_CO-theta_BSS)^2', ...
  'r_1','k_1*theta_O^2/(1-theta_O)^2', ...
  'r2','k2*c_CO*(1-theta_O-theta_CO-theta_BSS)', ...
  'r_2','k_2*theta_CO', ...
  'r3','k3*theta_O*theta_CO', ...
  'r4','k4*theta_CO*theta_BSS', ...
  'r5','k5*theta_BSS^2/(1-theta_BSS)^2', ...
  'r_O','r1-r_1-r3', ...
  'r_CO','r2-r_2-r3-r4', ...
  'r_BSS','-r4-r5'};

% ODE Settings
clear ode
ode.dim={'Pco2','Pco','Po2'};
ode.f={'50-Pco2','Pco_in-Pco2-Pco','Po2_in-Pco2/2-Po2'};
ode.init={'1e-1','Pco_in','Po2_in'};
ode.dinit={'0','0','0'};
clear units;
units.basesystem = 'SI';
ode.units = units;
fem.ode=ode;

% Multiphysics
fem=multiphysics(fem);

% Extend mesh
fem.xmesh=meshextend(fem);

% Solve problem
fem.sol=femstatic(fem, ...
                  'u',0, ...
'solcomp',{'theta_O','Pco','Van','Vio','Pco2','theta_CO','Vca','Po2','theta_BSS'}, ...
'outcomp',{'theta_O','Pco','Van','Vio','Pco2','theta_CO','Vca','Po2','theta_BSS'}, ...
                  'blocksize','auto', ...
                  'maxiter',100);

% Save current fem structure for restart purposes
fem0=fem;

% Restart using previous solution

% Application mode 1
clear appl
appl.mode.class = 'FlPDEC';
appl.dim = {'Vio','Vio_t'};
appl.name = 'Velectrolyte';
appl.assignsuffix = '_Velectrolyte';
clear bnd
bnd.g = {0,'Janode','-Jcathode'};
bnd.type = 'neu';
bnd.ind = [1,3,2,3,2,1,1,1,1];
appl.bnd = bnd;
clear equ
equ.f = 0;
equ.da = 0;
equ.c = 'sigma_ionic';
equ.ind = [1];
appl.equ = equ;
```

```
fem.appl{1} = appl;

% Application mode 2
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'Vca','Vca_t'};
appl.name = 'Cathode_TPB_flux';
appl.assignsuffix = '_Cathode_TPB_flux';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm3','lm4'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear bnd
bnd.usage = {0,1};
bnd.weak = {0,'sigma_el_ca*(-VcaTx_test*Vca-VcaTy_test*Vca)+Vca_test*Jcathode'};
bnd.ind = [1,2,1,2,1,1,1,1,1];
appl.bnd = bnd;
fem.appl{2} = appl;

% Application mode 3
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'Vca','Vca_t2'};
appl.name = 'Vcathode';
appl.assignsuffix = '_Vcathode';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm5','lm6'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear pnt
pnt.constrf = {0,'test(Vca_0-Vca)'};
pnt.constr = {0,'Vca_0-Vca'};
pnt.ind = [1,1,2,1,1,1,2,1,1];
appl.pnt = pnt;
clear bnd
bnd.usage = {0,1};
bnd.weak = {0,'sigma_el_ca*(-VcaTx_test*VcaTx-VcaTy_test*VcaTy)'};
bnd.ind = [1,2,1,2,1,1,1,1,1];
appl.bnd = bnd;
fem.appl{3} = appl;

% Application mode 4
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'Van','Van_t'};
appl.name = 'Anode_TPB_flux';
appl.assignsuffix = '_Anode_TPB_flux';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm7','lm8'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear bnd
bnd.usage = {0,1};
bnd.weak = {0,'sigma_el_an*(-VanTx_test*Van-VanTy_test*Van)-Van_test*Janode'};
bnd.ind = [1,1,2,1,2,1,1,1,1];
appl.bnd = bnd;
fem.appl{4} = appl;

% Application mode 5
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'Van','Van_t2'};
appl.name = 'Vanode';
appl.assignsuffix = '_Vanode';
clear prop
```

```
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm9','lm10'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear pnt
pnt.constrf = {0,'test(Van_0-Van)'};
pnt.constr = {0,'Van_0-Van'};
pnt.ind = [1,1,1,2,1,1,1,1,1];
appl.pnt = pnt;
clear bnd
bnd.usage = {0,1};
bnd.weak = {0,'sigma_el_an*(-VanTx_test*VanTx-VanTy_test*VanTy)'};
bnd.ind = [1,1,2,1,2,1,1,1,1];
appl.bnd = bnd;
fem.appl{5} = appl;

% Application mode 6
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'theta_O','theta_O_t'};
appl.name = 'Oxygen_Coverage';
appl.assignsuffix = '_Oxygen_Coverage';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm11','lm12'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear bnd
bnd.usage = {0,1};
bnd.weak = {0,'D_O*(-theta_OTx_test*theta_OTx-
theta_OTy_test*theta_OTy)+theta_O_test*r_O'};
bnd.ind = [1,1,2,1,2,1,1,1,1];
appl.bnd = bnd;
fem.appl{6} = appl;

% Application mode 7
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'theta_CO','theta_CO_t'};
appl.name = 'CO_Coverage';
appl.assignsuffix = '_CO_Coverage';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm13','lm14'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear bnd
bnd.usage = {0,1};
bnd.weak = {0,'D_CO*(-theta_COTx_test*theta_COTx-
theta_COTy_test*theta_COTy)+theta_CO_test*r_CO'};
bnd.ind = [1,1,2,1,2,1,1,1,1];
appl.bnd = bnd;
fem.appl{7} = appl;

% Application mode 8
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'theta_BSS','theta_BSS_t2'};
appl.name = 'BSS_Coverage';
appl.assignsuffix = '_BSS_Coverage';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm17','lm18'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear bnd
bnd.usage = {0,1};
```

```
bnd.weak = {0,'D_BSS*(-theta_BSSTx_test*theta_BSSTx-
theta_BSSTy_test*theta_BSSTy)+theta_BSS_test*(r_BSS+abs(Jan_BSS)/2/F/Ns*(1-theta_O-
theta_CO-theta_BSS))'};
bnd.ind = [1,1,2,1,2,1,1,1,1];
appl.bnd = bnd;
fem.appl{8} = appl;
fem.frame = {'ref'};
fem.border = 1;
clear units;
units.basesystem = 'SI';
fem.units = units;

% Subdomain settings
clear equ
equ.ind = [1];
equ.dim = {'Vio'};

% Subdomain expressions
equ.expr = {'sigma_ionic','33400[S/m]*exp((-10300)[K]/T)'};
fem.equ = equ;

% Boundary settings
clear bnd
bnd.ind = [1,2,3,2,3,1,1,1,1];
bnd.dim = {'Vio','Vca','Vca','Van','Van','theta_O','theta_CO', ...
  'theta_BSS'};

% Boundary expressions
bnd.expr = {'sigma_el_ca',{'','1/(-1.145028e-09+1.275961e-17*T^3-4.720065e-
16*T^2+7.877040999999999e-11*T)', ...
  ''}, ...
  'sigma_el_an',{'','','1/(-4.843579e-08+2.814022e-17*T^3-1.600249e-13*T^2+5.552497e-
10*T)'}};
fem.bnd = bnd;

% Point settings
clear pnt
pnt.ind = [1,1,2,3,1,1,1,1,1];

% Point expressions
pnt.expr = {'Van_0',{'','','0[V]'}, ...
  'Vca_0',{'','0.7[V]',''}};
fem.pnt = pnt;

% Coupling variable elements
clear elemcpl

% Integration coupling variables
clear elem
elem.elem = 'elcplscalar';
elem.g = {'1'};
src = cell(1,1);
clear bnd
bnd.expr = {{{},'Jcathode/4/F',{},{}},{{},{},'r3+r4',{}},{{},{}, ...
  'Jan_co2/2/F',{}},{{},{},'r2-r_2',{}},{{},{},'r1-r_1',{}},{{},{}, ...
  'Jan_o2/4/F',{}},{{},{},'r5',{}}};
bnd.ipoints = {{{},'4',{},'4'},{{},{},'4',{}},{{},{},'4',{}},{{},{},'4',{}},{{}, ...
  {},'4',{}},{{},{},'4',{}},{{},{},'4',{}}};
bnd.frame = {{{},'ref',{},'ref'},{{},{},'ref',{}},{{},{},'ref',{}},{{},{}, ...
  'ref',{}},{{},{},'ref',{}},{{},{},'ref',{}},{{},{},'ref',{}}};
bnd.ind = {{'1','6','9'},{'2','4'},{'3','5'},{'7','8'}};
src{1} = {{},bnd,{}};
elem.src = src;
geomdim = cell(1,1);
geomdim{1} = {};
elem.geomdim = geomdim;
elem.var = {xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx};
elem.global = {'1','2','3','4','5','6','7'};
elem.maxvars = {};
elemcpl{1} = elem;
fem.elemcpl = elemcpl;
```

```
% Global expressions
fem.globalexpr = {'Voc','(0.5*(mu_co_0-mu_co2_0-
mu_BSS)+0.5*R*T*log(Po2^(3/2)/Po2^(1/2))+0.5*R*T*log(Pco/Pco2))/(1*F)', ...
  'n_c','Voc-(Vca-Vio)', ...
  'Jo_c','gamma_c*(Po2/Pref)^(0.25)*exp(-Eact_c/(R*T))', ...
  'Jcathode','3*Jo_c*(exp(a_c*2*F*n_c/(R*T))-exp(-(1-a_c)*2*F*n_c/(R*T)))', ...
  'n_a','Van-Vio', ...
  'Jo_a1','gamma_a1*Pco/Pref*Pco2/Pref*exp(-Eact_a/(R*T))', ...
  'Jo_a2','A_a2*exp(-Eact_a/(R*T))', ...
  'Jo_a3','gamma_a3*(Po2/Pref)^(-0.25)*exp(-Eact_a/(R*T))', ...
  'Jan_co2','Jo_a1*(exp(a_a*2*F*n_a/(R*T))-exp(-(1-a_a)*2*F*n_a/(R*T)))', ...
  'Jan_BSS','Jo_a2*(exp(a_a*2*F*n_a/(R*T))-exp(-(1-a_a)*2*F*n_a/(R*T)))', ...
  'Jan_o2','Jo_a3*(exp(a_a*2*F*n_a/(R*T))-exp(-(1-a_a)*2*F*n_a/(R*T)))', ...
  'Janode','Jan_co2+Jan_BSS+Jan_o2', ...
  'c_o2','Po2/(R*T)', ...
  'c_CO','Pco/(R*T)', ...
  'k1','S_o2*sqrt(0.5*R*T/(pi*M_o2))/Ns', ...
  'k_1','ko_1*exp(-E_1/(R*T))', ...
  'k2','S_CO*sqrt(0.5*R*T/(pi*M_CO))/Ns', ...
  'k_2','ko_2*exp(-E_2/(R*T))', ...
  'k3','ko3*exp(-E3/(R*T))', ...
  'k5','1e-2-k4', ...
  'r1','2*k1*c_o2*(1-theta_O-theta_CO-theta_BSS)^2', ...
  'r_1','k_1*theta_O^2/(1-theta_O)^2', ...
  'r2','k2*c_CO*(1-theta_O-theta_CO-theta_BSS)', ...
  'r_2','k_2*theta_CO', ...
  'r3','k3*theta_O*theta_CO', ...
  'r4','k4*theta_CO*theta_BSS', ...
  'r5','k5*theta_BSS^2/(1-theta_BSS)^2', ...
  'r_O','r1-r_1-r3', ...
  'r_CO','r2-r_2-r3-r4', ...
  'r_BSS','-r4-r5'};

% ODE Settings
clear ode
ode.dim={'Pco2','Pco','Po2'};
ode.f={'50-Pco2','Pco_in-Pco2-Pco','Po2_in-Pco2/2-Po2'};
ode.init={'1e-1','Pco_in','Po2_in'};
ode.dinit={'0','0','0'};
clear units;
units.basesystem = 'SI';
ode.units = units;
fem.ode=ode;

% Multiphysics
fem=multiphysics(fem);

% Extend mesh
fem.xmesh=meshextend(fem);

% Solve problem
fem.sol=femstatic(fem, ...
                  xxxxxxxxxxxxxxxxxx ...
'solcomp',{'theta_O','Pco','Van','Vio','Pco2','theta_CO','Vca','Po2','theta_BSS'}, ...
'outcomp',{'theta_O','Pco','Van','Vio','Pco2','theta_CO','Vca','Po2','theta_BSS'}, ...
                  'blocksize','auto', ...
                  'maxiter',100);

% Save current fem structure for restart purposes
fem0=fem;

% Constants
fem.const = {'T',Tin, ...
  'F','96485 [s*A/mol]', ...
  'R','8.3145 [J/mol/K]', ...
  'gamma_c','6.91e8 [A/m^2]', ...
  'gamma_a1','2.92e+011[A/m^2]', ...
  'A_a2','3.42e+04 [A/m^2]', ...
  'gamma_a3','5.01e+08 [A/m^2]', ...
  'Pref','101325 [Pa]', ...
```

```
  'Eact_c','110e3 [J/mol]', ...
  'Eact_a','120e3 [J/mol]', ...
  'a_c','0.5 [1]', ...
  'a_a','0.5 [1]', ...
  'D_O','0 [m^2/s]', ...
  'D_CO','0 [m^2/s]', ...
  'D_BSS','4e-15 [m^2/s]', ...
  'Ns','2.5407e-5 [mol/m^2]', ...
  'M_o2','32e-3 [kg/mol]', ...
  'M_CO','28e-3 [kg/mol]', ...
  'S_o2','7.69e-05 [1]', ...
  'S_CO','5.38e-01 [1]', ...
  'ko_1','2.4e13 [1/s]', ...
  'E_1','2.43139e+05 [J/mol]', ...
  'ko_2','6.5e13 [1/s]', ...
  'E_2','9.9618e+04 [J/mol]', ...
  'ko3','2.7e6 [1/s]', ...
  'E3','3.5186e+04 [J/mol]', ...
  'mu_co_0','-137.3e3 [J/mol]', ...
  'mu_co2_0','-394.4e3 [J/mol]', ...
  'mu_BSS','-236.4e3 [J/mol]', ...
  'width','2e-2 [m]', ...
  'Po2_in',Po2_inlet, ...
  'Pco_in',Pco_inlet, ...
  'Feed', Fd, ...
  'k4','5.73e-03 [1/s]'};

% Restart using previous solution

% Application mode 1
clear appl
appl.mode.class = 'FlPDEC';
appl.dim = {'Vio','Vio_t'};
appl.name = 'Velectrolyte';
appl.assignsuffix = '_Velectrolyte';
clear bnd
bnd.g = {0,'Janode','-Jcathode'};
bnd.type = 'neu';
bnd.ind = [1,3,2,3,2,1,1,1,1];
appl.bnd = bnd;
clear equ
equ.f = 0;
equ.da = 0;
equ.c = 'sigma_ionic';
equ.ind = [1];
appl.equ = equ;
fem.appl{1} = appl;

% Application mode 2
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'Vca','Vca_t'};
appl.name = 'Cathode_TPB_flux';
appl.assignsuffix = '_Cathode_TPB_flux';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm3','lm4'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear bnd
bnd.usage = {0,1};
bnd.weak = {0,'sigma_el_ca*(-VcaTx_test*Vca-VcaTy_test*Vca)+Vca_test*Jcathode'};
bnd.ind = [1,2,1,2,1,1,1,1,1];
appl.bnd = bnd;
fem.appl{2} = appl;

% Application mode 3
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'Vca','Vca_t2'};
```

```
appl.name = 'Vcathode';
appl.assignsuffix = '_Vcathode';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm5','lm6'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear pnt
pnt.constrf = {0,'test(Vca_0-Vca)'};
pnt.constr = {0,'Vca_0-Vca'};
pnt.ind = [1,1,2,1,1,1,2,1,1];
appl.pnt = pnt;
clear bnd
bnd.usage = {0,1};
bnd.weak = {0,'sigma_el_ca*(-VcaTx_test*VcaTx-VcaTy_test*VcaTy)'};
bnd.ind = [1,2,1,2,1,1,1,1,1];
appl.bnd = bnd;
fem.appl{3} = appl;

% Application mode 4
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'Van','Van_t'};
appl.name = 'Anode_TPB_flux';
appl.assignsuffix = '_Anode_TPB_flux';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm7','lm8'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear bnd
bnd.usage = {0,1};
bnd.weak = {0,'sigma_el_an*(-VanTx_test*Van-VanTy_test*Van)-Van_test*Janode'};
bnd.ind = [1,1,2,1,2,1,1,1,1];
appl.bnd = bnd;
fem.appl{4} = appl;

% Application mode 5
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'Van','Van_t2'};
appl.name = 'Vanode';
appl.assignsuffix = '_Vanode';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm9','lm10'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear pnt
pnt.constrf = {0,'test(Van_0-Van)'};
pnt.constr = {0,'Van_0-Van'};
pnt.ind = [1,1,1,2,1,1,1,1,1];
appl.pnt = pnt;
clear bnd
bnd.usage = {0,1};
bnd.weak = {0,'sigma_el_an*(-VanTx_test*VanTx-VanTy_test*VanTy)'};
bnd.ind = [1,1,2,1,2,1,1,1,1];
appl.bnd = bnd;
fem.appl{5} = appl;

% Application mode 6
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'theta_O','theta_O_t'};
appl.name = 'Oxygen_Coverage';
appl.assignsuffix = '_Oxygen_Coverage';
clear prop
clear weakconstr
```

```
weakconstr.value = 'off';
weakconstr.dim = {'lm11','lm12'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear bnd
bnd.usage = {0,1};
bnd.weak = {0,'D_O*(-theta_OTx_test*theta_OTx-
theta_OTy_test*theta_OTy)+theta_O_test*r_O'};
bnd.ind = [1,1,2,1,2,1,1,1,1];
appl.bnd = bnd;
fem.appl{6} = appl;

% Application mode 7
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'theta_CO','theta_CO_t'};
appl.name = 'CO_Coverage';
appl.assignsuffix = '_CO_Coverage';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm13','lm14'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear bnd
bnd.usage = {0,1};
bnd.weak = {0,'D_CO*(-theta_COTx_test*theta_COTx-
theta_COTy_test*theta_COTy)+theta_CO_test*r_CO'};
bnd.ind = [1,1,2,1,2,1,1,1,1];
appl.bnd = bnd;
fem.appl{7} = appl;

% Application mode 8
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'theta_BSS','theta_BSS_t2'};
appl.name = 'BSS_Coverage';
appl.assignsuffix = '_BSS_Coverage';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm17','lm18'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear bnd
bnd.usage = {0,1};
bnd.weak = {0,'D_BSS*(-theta_BSSTx_test*theta_BSSTx-
theta_BSSTy_test*theta_BSSTy)+theta_BSS_test*(r_BSS+abs(Jan_BSS)/2/F/Ns*(1-theta_O-
theta_CO-theta_BSS))'};
bnd.ind = [1,1,2,1,2,1,1,1,1];
appl.bnd = bnd;
fem.appl{8} = appl;
fem.frame = {'ref'};
fem.border = 1;
clear units;
units.basesystem = 'SI';
fem.units = units;

% Subdomain settings
clear equ
equ.ind = [1];
equ.dim = {'Vio'};

% Subdomain expressions
equ.expr = {'sigma_ionic','33400[S/m]*exp((-10300)[K]/T)'};
fem.equ = equ;

% Boundary settings
clear bnd
bnd.ind = [1,2,3,2,3,1,1,1,1];
bnd.dim = {'Vio','Vca','Vca','Van','Van','theta_O','theta_CO', ...
```

```
  'theta_BSS'};

% Boundary expressions
bnd.expr = {'sigma_el_ca',{'','1/(-1.145028e-09+1.275961e-17*T^3-4.720065e-
16*T^2+7.877040999999999e-11*T)', ...
  ''}, ...
  'sigma_el_an',{'','','1/(-4.843579e-08+2.814022e-17*T^3-1.600249e-13*T^2+5.552497e-
10*T)'}};
fem.bnd = bnd;

% Point settings
clear pnt
pnt.ind = [1,1,2,3,1,1,1,1,1];

% Point expressions
pnt.expr = {'Van_0',{'','','0[V]'}, ...
  'Vca_0',{'','0.7[V]',''}};
fem.pnt = pnt;

% Coupling variable elements
clear elemcpl

% Integration coupling variables
clear elem
elem.elem = 'elcplscalar';
elem.g = {'1'};
src = cell(1,1);
clear bnd
bnd.expr = {{{},'Jcathode/4/F',{},{}},{{},{},'r3+r4',{}},{{},{}, ...
  'Jan_co2/2/F',{}},{{},{},'r2-r_2',{}},{{},{},'r1-r_1',{}},{{},{}, ...
  'Jan_o2/4/F',{}},{{},{},'r5',{}}};
bnd.ipoints = {{{},'4',{},'4'},{{},{},'4',{}},{{},{},'4',{}},{{},{},'4',{}},{{}, ...
  {},'4',{}},{{},{},'4',{}},{{},{},'4',{}}};
bnd.frame = {{{},'ref',{},'ref'},{{},{},'ref',{}},{{},{},'ref',{}},{{},{}, ...
  'ref',{}},{{},{},'ref',{}},{{},{},'ref',{}},{{},{},'ref',{}}};
bnd.ind = {{'1','6','9'},{'2','4'},{'3','5'},{'7','8'}};
src{1} = {{},bnd,{}};
elem.src = src;
geomdim = cell(1,1);
geomdim{1} = {};
elem.geomdim = geomdim;
elem.var = {'Int1','Int2','Int3','Int4','Int5','Int6','Int7'};
elem.global = {'1','2','3','4','5','6','7'};
elem.maxvars = {};
elemcpl{1} = elem;
fem.elemcpl = elemcpl;

% Global expressions
fem.globalexpr = {'Voc','(0.5*(mu_co_0-mu_co2_0-
mu_BSS)+0.5*R*T*log(Po2^(3/2)/Po2^(1/2))+0.5*R*T*log(Pco/Pco2))/(1*F)', ...
  'n_c','Voc-(Vca-Vio)', ...
  'Jo_c','gamma_c*(Po2/Pref)^(0.25)*exp(-Eact_c/(R*T))', ...
  'Jcathode','3*Jo_c*(exp(a_c*2*F*n_c/(R*T))-exp(-(1-a_c)*2*F*n_c/(R*T)))', ...
  'n_a','Van-Vio', ...
  'Jo_a1','gamma_a1*Pco/Pref*Pco2/Pref*exp(-Eact_a/(R*T))', ...
  'Jo_a2','A_a2*exp(-Eact_a/(R*T))', ...
  'Jo_a3','gamma_a3*(Po2/Pref)^(-0.25)*exp(-Eact_a/(R*T))', ...
  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, ...
  'Jan_BSS','Jo_a2*(exp(a_a*2*F*n_a/(R*T))-exp(-(1-a_a)*2*F*n_a/(R*T)))', ...
  'Jan_o2','Jo_a3*(exp(a_a*2*F*n_a/(R*T))-exp(-(1-a_a)*2*F*n_a/(R*T)))', ...
  'Janode','Jan_co2+Jan_BSS+Jan_o2', ...
  'c_o2','Po2/(R*T)', ...
  'c_CO','Pco/(R*T)', ...
  'k1','S_o2*sqrt(0.5*R*T/(pi*M_o2))/Ns', ...
  'k_1','ko_1*exp(-E_1/(R*T))', ...
  'k2','S_CO*sqrt(0.5*R*T/(pi*M_CO))/Ns', ...
  'k_2','ko_2*exp(-E_2/(R*T))', ...
  'k3','ko3*exp(-E3/(R*T))', ...
  'k5','1e-2-k4', ...
  'r1','2*k1*c_o2*(1-theta_O-theta_CO-theta_BSS)^2', ...
  'r_1','k_1*theta_O^2/(1-theta_O)^2', ...
```

```
  'r2','k2*c_CO*(1-theta_O-theta_CO-theta_BSS)', ...
  'r_2','k_2*theta_CO', ...
  'r3','k3*theta_O*theta_CO', ...
  'r4','k4*theta_CO*theta_BSS', ...
  'r5','k5*theta_BSS^2/(1-theta_BSS)^2', ...
  'r_O','r1-r_1-r3', ...
  'r_CO','r2-r_2-r3-r4', ...
  'r_BSS','-r4-r5'};

% ODE Settings
clear ode
ode.dim={'Pco2','Pco','Po2'};
ode.f={'50-Pco2','Pco_in-Pco2-Pco','Po2_in-Pco2/2-Po2'};
ode.init={'1e-1','Pco_in','Po2_in'};
ode.dinit={'0','0','0'};
clear units;
units.basesystem = 'SI';
ode.units = units;
fem.ode=ode;

% Multiphysics
fem=multiphysics(fem);

% Extend mesh
fem.xmesh=meshextend(fem);

% Solve problem
fem.sol=femstatic(fem, ...
                  'init',fem0.sol, ...
'solcomp',{'theta_O','Pco','Van','Vio','Pco2','theta_CO','Vca','Po2','theta_BSS'}, ...
'outcomp',{'theta_O','Pco','Van','Vio','Pco2','theta_CO','Vca','Po2','theta_BSS'}, ...
                  'blocksize','auto', ...
                  'maxiter',100);

% Save current fem structure for restart purposes
fem0=fem;

% Restart using previous solution

% Application mode 1
clear appl
appl.mode.class = 'FlPDEC';
appl.dim = {'Vio','Vio_t'};
appl.name = 'Velectrolyte';
appl.assignsuffix = '_Velectrolyte';
clear bnd
bnd.g = {0,'Janode','-Jcathode'};
bnd.type = 'neu';
bnd.ind = [1,3,2,3,2,1,1,1,1];
appl.bnd = bnd;
clear equ
equ.f = 0;
equ.da = 0;
equ.c = 'sigma_ionic';
equ.ind = [1];
appl.equ = equ;
fem.appl{1} = appl;

% Application mode 2
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'Vca','Vca_t'};
appl.name = 'Cathode_TPB_flux';
appl.assignsuffix = '_Cathode_TPB_flux';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm3','lm4'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear bnd
```

```
bnd.usage = {0,1};
bnd.weak = {0,'sigma_el_ca*(-VcaTx_test*Vca-VcaTy_test*Vca)+Vca_test*Jcathode'};
bnd.ind = [1,2,1,2,1,1,1,1,1];
appl.bnd = bnd;
fem.appl{2} = appl;

% Application mode 3
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'Vca','Vca_t2'};
appl.name = 'Vcathode';
appl.assignsuffix = '_Vcathode';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm5','lm6'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear pnt
pnt.constrf = {0,'test(Vca_0-Vca)'};
pnt.constr = {0,'Vca_0-Vca'};
pnt.ind = [1,1,2,1,1,1,2,1,1];
appl.pnt = pnt;
clear bnd
bnd.usage = {0,1};
bnd.weak = {0,'sigma_el_ca*(-VcaTx_test*VcaTx-VcaTy_test*VcaTy)'};
bnd.ind = [1,2,1,2,1,1,1,1,1];
appl.bnd = bnd;
fem.appl{3} = appl;

% Application mode 4
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'Van','Van_t'};
appl.name = 'Anode_TPB_flux';
appl.assignsuffix = '_Anode_TPB_flux';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm7','lm8'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear bnd
bnd.usage = {0,1};
bnd.weak = {0,'sigma_el_an*(-VanTx_test*Van-VanTy_test*Van)-Van_test*Janode'};
bnd.ind = [1,1,2,1,2,1,1,1,1];
appl.bnd = bnd;
fem.appl{4} = appl;

% Application mode 5
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'Van','Van_t2'};
appl.name = 'Vanode';
appl.assignsuffix = '_Vanode';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm9','lm10'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear pnt
pnt.constrf = {0,'test(Van_0-Van)'};
pnt.constr = {0,'Van_0-Van'};
pnt.ind = [1,1,1,2,1,1,1,1,1];
appl.pnt = pnt;
clear bnd
bnd.usage = {0,1};
bnd.weak = {0,'sigma_el_an*(-VanTx_test*VanTx-VanTy_test*VanTy)'};
bnd.ind = [1,1,2,1,2,1,1,1,1];
appl.bnd = bnd;
```

```
fem.appl{5} = appl;

% Application mode 6
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'theta_O','theta_O_t'};
appl.name = 'Oxygen_Coverage';
appl.assignsuffix = '_Oxygen_Coverage';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm11','lm12'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear bnd
bnd.usage = {0,1};
bnd.weak = {0,'D_O*(-theta_OTx_test*theta_OTx-
theta_OTy_test*theta_OTy)+theta_O_test*r_O'};
bnd.ind = [1,1,2,1,2,1,1,1,1];
appl.bnd = bnd;
fem.appl{6} = appl;

% Application mode 7
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'theta_CO','theta_CO_t'};
appl.name = 'CO_Coverage';
appl.assignsuffix = '_CO_Coverage';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm13','lm14'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear bnd
bnd.usage = {0,1};
bnd.weak = {0,'D_CO*(-theta_COTx_test*theta_COTx-
theta_COTy_test*theta_COTy)+theta_CO_test*r_CO'};
bnd.ind = [1,1,2,1,2,1,1,1,1];
appl.bnd = bnd;
fem.appl{7} = appl;

% Application mode 8
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'theta_BSS','theta_BSS_t2'};
appl.name = 'BSS_Coverage';
appl.assignsuffix = '_BSS_Coverage';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm17','lm18'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear bnd
bnd.usage = {0,1};
bnd.weak = {0,'D_BSS*(-theta_BSSTx_test*theta_BSSTx-
theta_BSSTy_test*theta_BSSTy)+theta_BSS_test*(r_BSS+abs(Jan_BSS)/2/F/Ns*(1-theta_O-
theta_CO-theta_BSS))'};
bnd.ind = [1,1,2,1,2,1,1,1,1];
appl.bnd = bnd;
fem.appl{8} = appl;
fem.frame = {'ref'};
fem.border = 1;
clear units;
units.basesystem = 'SI';
fem.units = units;


% Subdomain settings
clear equ
```

```
equ.ind = [1];
equ.dim = {'Vio'};

% Subdomain expressions
equ.expr = {'sigma_ionic','33400[S/m]*exp((-10300)[K]/T)'};
fem.equ = equ;

% Boundary settings
clear bnd
bnd.ind = [1,2,3,2,3,1,1,1,1];
bnd.dim = {'Vio','Vca','Vca','Van','Van','theta_O','theta_CO', ...
  'theta_BSS'};

% Boundary expressions
bnd.expr = {'sigma_el_ca',{'','1/(-1.145028e-09+1.275961e-17*T^3-4.720065e-
16*T^2+7.877040999999999e-11*T)', ...
  ''}, ...
  'sigma_el_an',{'','','1/(-4.843579e-08+2.814022e-17*T^3-1.600249e-13*T^2+5.552497e-
10*T)'}};
fem.bnd = bnd;

% Point settings
clear pnt
pnt.ind = [1,1,2,3,1,1,1,1,1];

% Point expressions
pnt.expr = {'Van_0',{'','','0[V]'}, ...
  'Vca_0',{'','0.7[V]',''}};
fem.pnt = pnt;

% Coupling variable elements
clear elemcpl
% Integration coupling variables
clear elem
elem.elem = 'elcplscalar';
elem.g = {'1'};
src = cell(1,1);
clear bnd
bnd.expr = {{{},'Jcathode/4/F',{},{}},{{},{},'r3+r4',{}},{{},{}, ...
  'Jan_co2/2/F',{}},{{},{},'r2-r_2',{}},{{},{},'r1-r_1',{}},{{},{}, ...
  'Jan_o2/4/F',{}},{{},{},'r5',{}}};
bnd.ipoints = {{{},'4',{},'4'},{{},{},'4',{}},{{},{},'4',{}},{{},{},'4',{}},{{}, ...
  {},'4',{}},{{},{},'4',{}},{{},{},'4',{}}};
bnd.frame = {{{},'ref',{},'ref'},{{},{},'ref',{}},{{},{},'ref',{}},{{},{}, ...
  'ref',{}},{{},{},'ref',{}},{{},{},'ref',{}},{{},{},'ref',{}}};
bnd.ind = {{'1','6','9'},{'2','4'},{'3','5'},{'7','8'}};
src{1} = {{},bnd,{}};
elem.src = src;
geomdim = cell(1,1);
geomdim{1} = {};
elem.geomdim = geomdim;
elem.var = {'Int1','Int2','Int3','Int4','Int5','Int6','Int7'};
elem.global = {'1','2','3','4','5','6','7'};
elem.maxvars = {};
elemcpl{1} = elem;
fem.elemcpl = elemcpl;

% Global expressions
fem.globalexpr = {'Voc','(0.5*(mu_co_0-mu_co2_0-
mu_BSS)+0.5*R*T*log(Po2^(3/2)/Po2^(1/2))+0.5*R*T*log(Pco/Pco2))/(1*F)', ...
  'n_c','Voc-(Vca-Vio)', ...
  'Jo_c','gamma_c*(Po2/Pref)^(0.25)*exp(-Eact_c/(R*T))', ...
  'Jcathode','3*Jo_c*(exp(a_c*2*F*n_c/(R*T))-exp(-(1-a_c)*2*F*n_c/(R*T)))', ...
  'n_a','Van-Vio', ...
  'Jo_a1','gamma_a1*Pco/Pref*Pco2/Pref*exp(-Eact_a/(R*T))', ...
  'Jo_a2','A_a2*exp(-Eact_a/(R*T))', ...
  'Jo_a3','gamma_a3*(Po2/Pref)^(-0.25)*exp(-Eact_a/(R*T))', ...
  'Jan_co2','Jo_a1*(exp(a_a*2*F*n_a/(R*T))-exp(-(1-a_a)*2*F*n_a/(R*T)))', ...
  'Jan_BSS','Jo_a2*(exp(a_a*2*F*n_a/(R*T))-exp(-(1-a_a)*2*F*n_a/(R*T)))', ...
  'Jan_o2','Jo_a3*(exp(a_a*2*F*n_a/(R*T))-exp(-(1-a_a)*2*F*n_a/(R*T)))', ...
  'Janode','Jan_co2+Jan_BSS+Jan_o2', ...
```

```
  'c_o2','Po2/(R*T)', ...
  'c_CO','Pco/(R*T)', ...
  'k1','S_o2*sqrt(0.5*R*T/(pi*M_o2))/Ns', ...
  'k_1','ko_1*exp(-E_1/(R*T))', ...
  'k2','S_CO*sqrt(0.5*R*T/(pi*M_CO))/Ns', ...
  'k_2','ko_2*exp(-E_2/(R*T))', ...
  'k3','ko3*exp(-E3/(R*T))', ...
  'k5','1e-2-k4', ...
  'r1','2*k1*c_o2*(1-theta_O-theta_CO-theta_BSS)^2', ...
  'r_1','k_1*theta_O^2/(1-theta_O)^2', ...
  'r2','k2*c_CO*(1-theta_O-theta_CO-theta_BSS)', ...
  'r_2','k_2*theta_CO', ...
  'r3','k3*theta_O*theta_CO', ...
  'r4','k4*theta_CO*theta_BSS', ...
  'r5','k5*theta_BSS^2/(1-theta_BSS)^2', ...
  'r_O','r1-r_1-r3', ...
  'r_CO','r2-r_2-r3-r4', ...
  'r_BSS','-r4-r5'};

% ODE Settings
clear ode
ode.dim={'Pco2','Pco','Po2'};
ode.f={'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'};
ode.init={'1e-1','Pco_in','Po2_in'};
ode.dinit={'0','0','0'};
clear units;
units.basesystem = 'SI';
ode.units = units;
fem.ode=ode;

% Multiphysics
fem=multiphysics(fem);

% Extend mesh
fem.xmesh=meshextend(fem);

% Solve problem
fem.sol=femstatic(fem, ...
                  'init',fem0.sol, ...
'solcomp',{'theta_O','Pco','Van','Vio','Pco2','Vca','theta_CO','Po2','theta_BSS'}, ...
'outcomp',{'theta_O','Pco','Van','Vio','Pco2','Vca','theta_CO','Po2','theta_BSS'}, ...
                  'blocksize','auto', ...
                  'maxiter',100);

% Save current fem structure for restart purposes
fem0=fem;

% Rate due to NEMCA effect calculation
r_NEMCA=postint(fem,'(r3+r4)*Ns*width', ...
           'unit','', ...
           'recover','off', ...
           'dl',[3,5], ...
           'edim',1);

% Rate due to Faradaic effect calculation
r_Faradaic_1=postint(fem,'Jan_co2/2/F*width', ...
           'unit','', ...
           'recover','off', ...
           'dl',[3,5], ...
           'edim',1);


% Total rate calculation
r_Total=r_NEMCA+r_Faradaic;

end
```

A.2 The multi-scale framework files (Chapter 4)

A.2.1 MATLAB communication function

```
function [theta1 r1 t1 r_CO2_macro r_CO2_micro fem0] =
Multiscale_EPOC(Tin,Po2_in,Pco_in,Feed)

% Parameters
Site_density=2.5407e-5;
Nav=6.0221367e23;
Ns=Site_density*Nav;
R=8.3145;
Pco2_in=0.0;
par=[5.38e-1;7.69e-5;2.4e13;2.43139e5;6.5e13;0.99618e5;2.7e6;3.5186e4;5.73e-3;4.27e-
3;8.4e-2];
lsize=[1800;400];

% Initialization of variables
Init_time=0;
Time_incr=1e-6;
End_time=1e-5;
Pout(1)=Pco_in;
Pout(2)=Po2_in;
Pout(3)=Pco2_in;
Pout(4)=101325-Pout(1)-Pout(2)-Pout(3);
rate0=[0.0;0.0;0.0;0.0];
theta0=[1.0;0.0;0.0;0.0];
BSS_in=[0;0;0];
BSS_rest=0;

% Initialization of indexes
k=1;
l=0;
m=1;

while (End_time<1.0e-2)

if (Init_time==0)

% call kMC Simulator
[theta r
t]=Single_kMC(Ns,par,rate0,lsize,theta0,Init_time,End_time,Time_incr,Tin,Pout,BSS_in);
r_CO2_micro=mean2(r(end-2:end,3));

% Update of Partial Pressures
Pout(3)=(r_CO2_micro)*R*Tin/Feed;
Pout(2)=Po2_in-Pout(3)/2;
Pout(1)=Pco_in-Pout(3);

% call COMSOL script
[fem0 BSS_gen
r_CO2_macro]=MacroPotentials_init(Init_time,Time_incr,End_time,Pout(1),Pout(2),Pout(3),T
in,Site_density,Feed);

else

% call kMC Simulator
[theta r
t]=Single_kMC(Ns,par,rate0,lsize,theta0,Init_time,End_time,Time_incr,Tin,Pout,BSS_in);
r_CO2_micro=mean2(r(end-2:end,3));
```

```
% call COMSOL script
[fem0 BSS_gen
r_CO2_macro]=MacroPotentials_loop(fem0,Init_time,Time_incr,End_time,Pout(1),Pout(2),Pout
(3),Tin,Site_density,Feed);

end

% Update of Partial Pressures
Pout(3)=(r_CO2_macro+r_CO2_micro)*R*Tin/Feed;
Pout(2)=Po2_in-Pout(3)/2;
Pout(1)=Pco_in-Pout(3);
Pout(4)=101325-Pout(1)-Pout(2)-Pout(3);

% Update of time variables
Init_time=End_time;
End_time=End_time+1e-5;

% Update of initial conditions
theta0=theta(end,:);
rate0=r(end,:)+r_CO2_macro;

% BSS Flux for kMC simulation
BSS_gen=BSS_gen*Nav*1e-5;
BSS_gen=BSS_gen+BSS_rest;
BSS_rest=BSS_gen-floor(BSS_gen);
BSS_in(1)=floor(rand*floor(BSS_gen));
BSS_in(2)=floor(rand*BSS_in(1));
BSS_in(3)=floor(BSS_gen)-BSS_in(1)-BSS_in(2);

% Storage of output data
theta1(k:l+length(theta(:,1))-1,:)=theta(1:length(theta(:,1))-1,:);
r1(k:l+length(r(:,1))-1,:)=r(1:length(r(:,1))-1,:)+r_CO2_macro;
t1(k:l+length(theta(:,1))-1,1)=t(1:length(theta(:,1))-1,1);
P1(:,m)=Pout(:);

% Update of indexes
k=k+length(theta(:,1))-1;
l=k-1;
m=m+1;

end

% Storage of last value of output data
theta1(k,:)=theta(end,:);
r1(k,:)=r(end,:);
t1(k,1)=t(end,1);

end
```

A.2.2 Macroscopic model m.files

```
function [fem0 BSS_gen r_CO2_macro] =
MacroPotentials_init(Init_time,Time_incr,End_time,Pco_in,Po2_in,Pco2_in,Tin,Site_density
,Feed)

flclear fem

% COMSOL version
clear vrsn
vrsn.name = 'COMSOL 3.5';
vrsn.ext = 'a';
vrsn.major = 0;
vrsn.build = 603;
vrsn.rcs = '$Name:  $';
vrsn.date = '$Date: 2008/12/03 17:02:19 $';
fem.version = vrsn;

% Geometry
g1=block3('4.6e-7','1.023e-7','5e-
6','base','corner','pos',{'0','0','0'},'axis',{'0','0','1'},'rot','0');
g2=curve3([4.0E-7,4.0E-7],[0,1.023E-7],[0,0]);

% Analyzed geometry
clear c s
c.objs={g2};
c.name={'B1'};
c.tags={'g2'};

s.objs={g1};
s.name={'BLK1'};
s.tags={'g1'};

fem.draw=struct('c',c,'s',s);
fem.geom=geomcsg(fem);

% Initialize mesh
fem.mesh=meshinit(fem, ...
                  'hmaxfac',[3,1e-8,4,1e-8], ...
                  'hmaxsub',[1,1e-7]);

% Constants
fem.const = {'T',Tin, ...
  'F','96485 [s*A/mol]', ...
  'R','8.3145 [J/mol/K]', ...
  'gamma_c','6.91e8 [A/m^2]', ...
  'gamma_a1','2.91e11 [A/m^2]', ...
  'gamma_a2','3.42e4 [A/m^2]', ...
  'gamma_a3','5.01e7 [A/m^2]', ...
  'Pref','101325 [Pa]', ...
  'Eact_c','110e3 [J/mol]', ...
  'Eact_a','120e3 [J/mol]', ...
  'a_c','0.5 [1]', ...
  'a_a','0.5 [1]', ...
  'Ns',Site_density, ...
  'mu_co_0','-137268.672 [J/mol]', ...
  'mu_co2_0','-394383.84 [J/mol]', ...
  'mu_BSS','-236.4e3 [J/mol]', ...
  'Po2_in',Po2_in, ...
  'Pco_in',Pco_in, ...
  'Pco2_in',Pco2_in, ...
  'Nav','6.0221367e23 [mol^-1]', ...
  'Feed',Feed};

% Application mode 1
clear appl
appl.mode.class = 'FlPDEC';
appl.dim = {'Vio','Vio_t'};
appl.name = 'YSZ_Potential';
```

```
appl.assignsuffix = '_YSZ_Potential';
clear edg
edg.weak = {0,xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'};
edg.ind = [1,1,2,1,3,1,2,3,2,1,1,1,1,3,1];
appl.edg = edg;
clear bnd
bnd.type = 'neu';
bnd.ind = [1,1,1,1,1,1,1];
appl.bnd = bnd;
clear equ
equ.f = 0;
equ.da = 1e4;
equ.c = 'sigma_ionic';
equ.ind = [1];
appl.equ = equ;
fem.appl{1} = appl;

% Application mode 2
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'Vca','Vca_t'};
appl.name = 'Au_Potential';
appl.sshape = 2;
appl.assignsuffix = '_Au_Potential';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm3','lm4'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear edg
edg.constrf = {0,'test(Vca_0-Vca)',0};
edg.constr = {0,'Vca_0-Vca',0};
edg.weak = {0,0,'sigma_el_ca*(-VcaTx_test*Vca-VcaTy_test*Vca)-Vca_test*Jcathode'};
edg.ind = [1,2,3,1,1,1,3,1,3,1,1,1,1,1,1];
appl.edg = edg;
clear bnd
bnd.dweak = {0,'1e4*Vca_test*Vca_time'};
bnd.usage = {0,1};
bnd.weak = {0,'sigma_el_ca*(-VcaTx_test*VcaTx-VcaTy_test*VcaTy)'};
bnd.ind = [1,1,2,1,1,1,1];
appl.bnd = bnd;
fem.appl{2} = appl;

% Application mode 3
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'Van','Van_t'};
appl.name = 'Pt_Potential';
appl.sshape = 2;
appl.assignsuffix = '_Pt_Potential';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm5','lm6'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear edg
edg.constrf = {0,'test(Van_0-Van)',0};
edg.constr = {0,'Van_0-Van',0};
edg.weak = {0,0,'sigma_el_an*(-VanTx_test*Van-VanTy_test*Van)+Van_test*Janode'};
edg.ind = [1,1,1,2,3,1,1,3,1,1,1,1,1,3,1];
appl.edg = edg;
clear bnd
bnd.dweak = {0,'1e4*Van_test*Van_time'};
bnd.usage = {0,1};
bnd.weak = {0,'sigma_el_an*(-VanTx_test*VanTx-VanTy_test*VanTy)'};
bnd.ind = [1,1,1,2,1,1,1];
appl.bnd = bnd;
fem.appl{3} = appl;
fem.frame = {'ref'};
```

```
fem.border = 1;
clear units;
units.basesystem = 'SI';
fem.units = units;

% Subdomain settings
clear equ
equ.ind = [1];
equ.dim = {'Vio'};

% Subdomain expressions
equ.expr = {'sigma_ionic','33400[S/m]*exp((-10300)[K]/T)'};
fem.equ = equ;

% Boundary settings
clear bnd
bnd.ind = [1,1,2,3,1,1,1];
bnd.dim = {'Vio','Vca','Van'};

% Boundary expressions
bnd.expr = {'sigma_el_an',{'','','1/(-4.843579e-08+2.814022e-17*T^3-1.600249e-
13*T^2+5.552497e-10*T)'}, ...
  'sigma_el_ca',{'','1/(-1.145028e-09+1.275961e-17*T^3-4.720065e-
16*T^2+7.877040999999999e-11*T)', ...
  ''}};
fem.bnd = bnd;

% Edge settings
clear edg
edg.ind = [1,2,1,3,1,1,1,1,1,1,1,1,1,1,1];

% Edge expressions
edg.expr = {'Van_0',{'','','0 [V]'}, ...
  'Vca_0',{'','0.7 [V]',''}};
fem.edg = edg;

% Coupling variable elements
clear elemcpl
% Integration coupling variables
clear elem
elem.elem = 'elcplscalar';
elem.g = {'1'};
src = cell(1,1);
clear bnd
bnd.expr = {{{},'Jan_co2/2/F'}};
bnd.ipoints = {{{},'4'}};
bnd.frame = {{{},'ref'}};
bnd.ind = {{'1','2','3','5','6','7'},{'4'}};
src{1} = {{},{},bnd,{}};
elem.src = src;
geomdim = cell(1,1);
geomdim{1} = {};
elem.geomdim = geomdim;
elem.var = {'Int1'};
elem.global = {'1'};
elemcpl{1} = elem;
fem.elemcpl = elemcpl;

% Global expressions
fem.globalexpr = {'Voc','(0.5*(mu_co_0-mu_co2_0-
mu_BSS)+0.25*R*T*log(Po2^3/Po2)+0.5*R*T*log(Pco/Pco2))/(1*F)', ...
  'n_c','Voc-(Vca-Vio)', ...
  'Jo_c','gamma_c*exp(-Eact_c/(R*T))', ...
  'Jcathode','3*Jo_c*(exp(a_c*2*F*n_c/(R*T))-exp(-(1-a_c)*2*F*n_c/(R*T)))', ...
  'n_a','Van-Vio', ...
  'Jo_a1','gamma_a1*Pco*exp(-Eact_a/(R*T))/(Pref*(Pco2/Pref)^0.5)', ...
  'Jo_a2','gamma_a2*exp(-Eact_a/(R*T))', ...
  'Jo_a3','gamma_a3*(Po2/Pref)^(-0.25)*exp(-Eact_a/(R*T))', ...
  'Jan_co2','Jo_a1*(exp(a_a*2*F*n_a/(R*T))-exp(-(1-a_a)*2*F*n_a/(R*T)))', ...
  'Jan_BSS','Jo_a2*(exp(a_a*2*F*n_a/(R*T))-exp(-(1-a_a)*2*F*n_a/(R*T)))', ...
  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, ...
```

```
  'Janode','Jan_co2+Jan_BSS+Jan_o2'};

% ODE Settings
clear ode
ode.dim={'Pco2','Pco','Po2'};
ode.f={'Int1*R*T/Feed+Pco2_in-Pco2','(Pco_in-Pco2)-Pco','(Po2_in-Pco2/2)-Po2'};
ode.init={'Pco2_in','Pco_in','Po2_in'};
ode.dinit={'0','0','0'};
clear units;
units.basesystem = 'SI';
ode.units = units;
fem.ode=ode;

% Multiphysics
fem=multiphysics(fem);

% Extend mesh
fem.xmesh=meshextend(fem);

% Solve problem
fem.sol=femtime(fem, ...
                xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, ...
                'solcomp',{'Pco','Van','Vio','Pco2','Vca','Po2'}, ...
                'outcomp',{'Pco','Van','Vio','Pco2','Vca','Po2'}, ...
                'blocksize','auto', ...
                'tlist',[colon(Init_time,Time_incr,End_time)], ...
                'tout','tlist', ...
                'linsolver','gmres');

% Save current fem structure for restart purposes
fem0=fem;

% BSS generation calculation
BSS_gen=postint(fem,'Jan_BSS/2/F', ...
                'unit','', ...
                'recover','off', ...
                'dl',4, ...
                'edim',2, ...
                'solnum','end');

% CO2 Faradaic rate calculation
r_CO2_macro=postint(fem,'Jan_co2/2/F', ...
                    'unit','', ...
                    'recover','off', ...
                    'dl',4, ...
                    'edim',2, ...
                    'solnum','end');
```

```
function [fem0 BSS_gen r_CO2_macro] =
MacroPotentials_loop(fem0,Init_time,Time_incr,End_time,Pco_in,Po2_in,Pco2_in,Tin,Site_de
nsity,Feed)

flclear fem

% COMSOL version
clear vrsn
vrsn.name = 'COMSOL 3.5';
vrsn.ext = 'a';
vrsn.major = 0;
vrsn.build = 603;
vrsn.rcs = '$Name:  $';
vrsn.date = '$Date: 2008/12/03 17:02:19 $';
fem.version = vrsn;

% Geometry
g1=block3('4.6e-7','1.023e-7','5e-
6','base','corner','pos',{'0','0','0'},'axis',{'0','0','1'},'rot','0');
g2=curve3([4.0E-7,4.0E-7],[0,1.023E-7],[0,0]);

% Analyzed geometry
clear c s
c.objs={g2};
c.name={'B1'};
c.tags={'g2'};

s.objs={g1};
s.name={'BLK1'};
s.tags={'g1'};

fem.draw=struct('c',c,'s',s);
fem.geom=geomcsg(fem);

% Initialize mesh
fem.mesh=meshinit(fem, ...
                  'hmaxfac',[3,1e-8,4,1e-8], ...
                  'hmaxsub',[1,1e-7]);

% Constants
fem.const = {'T',Tin, ...
  'F','96485 [s*A/mol]', ...
  'R','8.3145 [J/mol/K]', ...
  'gamma_c','6.91e8 [A/m^2]', ...
  'gamma_a1','2.91e11 [A/m^2]', ...
  'gamma_a2','3.42e4 [A/m^2]', ...
  'gamma_a3','5.01e7 [A/m^2]', ...
  'Pref','101325 [Pa]', ...
  'Eact_c','110e3 [J/mol]', ...
  'Eact_a','120e3 [J/mol]', ...
  'a_c','0.5 [1]', ...
  'a_a','0.5 [1]', ...
  'Ns',Site_density, ...
  'mu_co_0','-137268.672 [J/mol]', ...
  'mu_co2_0','-394383.84 [J/mol]', ...
  'mu_BSS','-236.4e3 [J/mol]', ...
  'Po2_in',Po2_in, ...
  'Pco_in',Pco_in, ...
  'Pco2_in',Pco2_in, ...
  'Nav','6.0221367e23 [mol^-1]', ...
  'Feed',Feed};

% Application mode 1
clear appl
appl.mode.class = 'FlPDEC';
appl.dim = {'Vio','Vio_t'};
appl.name = 'YSZ_Potential';
appl.assignsuffix = '_YSZ_Potential';
clear edg
edg.weak = {0,'sigma_ionic*(-VioTx_test*Vio-VioTy_test*Vio-
VioTz_test*Vio)+Vio_test*Jcathode', ...
```

```
  'sigma_ionic*(-VioTx_test*Vio-VioTy_test*Vio-VioTz_test*Vio)-Vio_test*Janode'};
edg.ind = [1,1,2,1,3,1,2,3,2,1,1,1,1,3,1];
appl.edg = edg;
clear bnd
bnd.type = 'neu';
bnd.ind = [1,1,1,1,1,1,1];
appl.bnd = bnd;
clear equ
equ.f = 0;
equ.da = 1e4;
equ.c = 'sigma_ionic';
equ.ind = [1];
appl.equ = equ;
fem.appl{1} = appl;


% Application mode 2
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'Vca','Vca_t'};
appl.name = 'Au_Potential';
appl.sshape = 2;
appl.assignsuffix = '_Au_Potential';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm3','lm4'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear edg
edg.constrf = {0,'test(Vca_0-Vca)',0};
edg.constr = {0,'Vca_0-Vca',0};
edg.weak = {0,0,'sigma_el_ca*(-VcaTx_test*Vca-VcaTy_test*Vca)-Vca_test*Jcathode'};
edg.ind = [1,2,3,1,1,1,3,1,3,1,1,1,1,1,1];
appl.edg = edg;
clear bnd
bnd.dweak = {0,'1e4*Vca_test*Vca_time'};
bnd.usage = {0,1};
bnd.weak = {0,'sigma_el_ca*(-VcaTx_test*VcaTx-VcaTy_test*VcaTy)'};
bnd.ind = [1,1,2,1,1,1,1];
appl.bnd = bnd;
fem.appl{2} = appl;

% Application mode 3
clear appl
appl.mode.class = 'FlPDEWBoundary';
appl.dim = {'Van','Van_t'};
appl.name = 'Pt_Potential';
appl.sshape = 2;
appl.assignsuffix = '_Pt_Potential';
clear prop
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm5','lm6'};
prop.weakconstr = weakconstr;
appl.prop = prop;
clear edg
edg.constrf = {0,'test(Van_0-Van)',0};
edg.constr = {0,'Van_0-Van',0};
edg.weak = {0,0,'sigma_el_an*(-VanTx_test*Van-VanTy_test*Van)+Van_test*Janode'};
edg.ind = [1,1,1,2,3,1,1,3,1,1,1,1,1,3,1];
appl.edg = edg;
clear bnd
bnd.dweak = {0,'1e4*Van_test*Van_time'};
bnd.usage = {0,1};
bnd.weak = {0,'sigma_el_an*(-VanTx_test*VanTx-VanTy_test*VanTy)'};
bnd.ind = [1,1,1,2,1,1,1];
appl.bnd = bnd;
fem.appl{3} = appl;
fem.frame = {'ref'};
fem.border = 1;
```

```
clear units;
units.basesystem = 'SI';
fem.units = units;

% Subdomain settings
clear equ
equ.ind = [1];
equ.dim = {'Vio'};

% Subdomain expressions
equ.expr = {'sigma_ionic','33400[S/m]*exp((-10300)[K]/T)'};
fem.equ = equ;

% Boundary settings
clear bnd
bnd.ind = [1,1,2,3,1,1,1];
bnd.dim = {'Vio','Vca','Van'};

% Boundary expressions
bnd.expr = {'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'}, ...
  'sigma_el_ca',{'','1/(-1.145028e-09+1.275961e-17*T^3-4.720065e-
16*T^2+7.877040999999999e-11*T)', ...
  ''}};
fem.bnd = bnd;

% Edge settings
clear edg
edg.ind = [1,2,1,3,1,1,1,1,1,1,1,1,1,1,1];

% Edge expressions
edg.expr = {'Van_0',{'','','0 [V]'}, ...
  'Vca_0',{'','0.7 [V]',''}};
fem.edg = edg;

% Coupling variable elements
clear elemcpl
% Integration coupling variables
clear elem
elem.elem = 'elcplscalar';
elem.g = {'1'};
src = cell(1,1);
clear bnd
bnd.expr = {{{},'Jan_co2/2/F'}};
bnd.ipoints = {{{},'4'}};
bnd.frame = {{{},'ref'}};
bnd.ind = {{'1','2','3','5','6','7'},{'4'}};
src{1} = {{},{},bnd,{}};
elem.src = src;
geomdim = cell(1,1);
geomdim{1} = {};
elem.geomdim = geomdim;
elem.var = {'Int1'};
elem.global = {'1'};
elemcpl{1} = elem;
fem.elemcpl = elemcpl;

% Global expressions
fem.globalexpr = {'Voc','(0.5*(mu_co_0-mu_co2_0-
mu_BSS)+0.25*R*T*log(Po2^3/Po2)+0.5*R*T*log(Pco/Pco2))/(1*F)', ...
  'n_c','Voc-(Vca-Vio)', ...
  'Jo_c','gamma_c*exp(-Eact_c/(R*T))', ...
  'Jcathode','3*Jo_c*(exp(a_c*2*F*n_c/(R*T))-exp(-(1-a_c)*2*F*n_c/(R*T)))', ...
  'n_a','Van-Vio', ...
  'Jo_a1','gamma_a1*Pco*exp(-Eact_a/(R*T))/(Pref*(Pco2/Pref)^0.5)', ...
  'Jo_a2','gamma_a2*exp(-Eact_a/(R*T))', ...
  'Jo_a3','gamma_a3*(Po2/Pref)^(-0.25)*exp(-Eact_a/(R*T))', ...
  'Jan_co2','Jo_a1*(exp(a_a*2*F*n_a/(R*T))-exp(-(1-a_a)*2*F*n_a/(R*T)))', ...
  'Jan_BSS','Jo_a2*(exp(a_a*2*F*n_a/(R*T))-exp(-(1-a_a)*2*F*n_a/(R*T)))', ...
  'Jan_o2','Jo_a3*(exp(a_a*2*F*n_a/(R*T))-exp(-(1-a_a)*2*F*n_a/(R*T)))', ...
  'Janode','Jan_co2+Jan_BSS+Jan_o2'};
```

```
% ODE Settings
clear ode
ode.dim={'Pco2','Pco','Po2'};
ode.f={'Int1*R*T/Feed+Pco2_in-Pco2','(Pco_in-Pco2)-Pco','(Po2_in-Pco2/2)-Po2'};
ode.init={'Pco2_in','Pco_in','Po2_in'};
ode.dinit={'0','0','0'};
clear units;
units.basesystem = 'SI';
ode.units = units;
fem.ode=ode;

% Multiphysics
fem=multiphysics(fem);


% Extend mesh
fem.xmesh=meshextend(fem);

% Solve problem
fem.sol=femtime(fem, ...
                'init',fem0.sol, ...
                'solcomp',{'Pco','Van','Vio','Pco2','Vca','Po2'}, ...
                'outcomp',{'Pco','Van','Vio','Pco2','Vca','Po2'}, ...
                'blocksize','auto', ...
                'tlist',[colon(Init_time,Time_incr,End_time)], ...
                'tout','tlist', ...
                xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx);

% Save current fem structure for restart purposes
fem0=fem;

% BSS generation calculation
BSS_gen=postint(fem,'Jan_BSS/2/F', ...
                'unit','', ...
                'recover','off', ...
                'dl',4, ...
                'edim',2, ...
                'solnum','end');

% CO2 Faradaic rate calculation
r_CO2_macro=postint(fem,'Jan_co2/2/F', ...
                    'unit','', ...
                    'recover','off', ...
                    'dl',4, ...
                    'edim',2, ...
                    'solnum','end');
```

A.2.3 Microscopic model files

A.2.3.1 Matlab function for kMC execution

```
function [theta r t]=
Single_kMC(Ns,par,rate0,lsize,theta0,Init_time,End_time,time_incr,Tin,Pin,BSS_in)

% writes 'parameters.dat' fortran input file
fid1=fopen('parameters.dat','w');
fprintf(fid1,'*********************************************************\n');
fprintf(fid1,'***  kMC LATTICE version 1.0 - Parameters input file  ***\n');
fprintf(fid1,'*********************************************************\n');
fprintf(fid1,'\n');
fprintf(fid1,'TEMPERATURE [K]:\n');
fprintf(fid1,'----------------\n');
fprintf(fid1,'%#e\n',Tin);
fprintf(fid1,'\n');
fprintf(fid1,'\n');
fprintf(fid1,'SURFACE SPECIFICATIONS:\n');
fprintf(fid1,'-----------------------\n');
fprintf(fid1,'Site density (sites/m^2): %#e\n',Ns);
fprintf(fid1,'\n');
fprintf(fid1,'\n');
fprintf(fid1,'GASEOUS CHEMICAL SPECIES: 4\n');
fprintf(fid1,'---------------------------\n');
fprintf(fid1,'name: CO\n');
fprintf(fid1,'molecular weight [kg/mol]: 2.8E-02\n');
fprintf(fid1,'sticking coefficient: %#e\n',par(1));
fprintf(fid1,'Partial Pressure [Pa]: %f\n',Pin(1));
fprintf(fid1,'Initial rate [mol/sec]: %#e\n',rate0(1));
fprintf(fid1,'-----\n');
fprintf(fid1,'name: O2\n');
fprintf(fid1,'molecular weight [kg/mol]: 32.0e-3\n');
fprintf(fid1,'sticking coefficient: %#e\n',par(2));
fprintf(fid1,'Partial Pressure [Pa]: %f\n',Pin(2));
fprintf(fid1,'Initial rate [mol/sec]: %#e\n',rate0(2));
fprintf(fid1,'-----\n');
fprintf(fid1,'name: CO2\n');
fprintf(fid1,'molecular weight [kg/mol]: 44.0e-3\n');
fprintf(fid1,'sticking coefficient: 0.0\n');
fprintf(fid1,'Partial Pressure [Pa]: %f\n',Pin(3));
fprintf(fid1,'Initial rate [mol/sec]: %#e\n',rate0(3));
fprintf(fid1,'-----\n');
fprintf(fid1,'name: He\n');
fprintf(fid1,'molecular weight [kg/mol]: 4.0e-3\n');
fprintf(fid1,'sticking coefficient: 0.0\n');
fprintf(fid1,'Partial Pressure [Pa]: %f\n',Pin(4));
fprintf(fid1,'Initial rate [mol/sec]: %#e\n',rate0(4));
fprintf(fid1,'---------------------------\n');
fprintf(fid1,'\n');
fprintf(fid1,'\n');
fprintf(fid1,'ADSORBED CHEMICAL SPECIES: 3\n');
fprintf(fid1,'----------------------------\n');
fprintf(fid1,'name: *CO\n');
fprintf(fid1,'name: *O\n');
fprintf(fid1,'name: *BSS\n');
fprintf(fid1,'----------------------------\n');
fprintf(fid1,'\n');
fprintf(fid1,'\n');
fprintf(fid1,'CHEMICAL REACTIONS: 8\n');
fprintf(fid1,' 1) ----------------------------------\n');
fprintf(fid1,'Dissociative adsorption\n');
fprintf(fid1,'Gas: O2\n');
fprintf(fid1,'Adsorbed: *O\n');
fprintf(fid1,' 2) ----------------------------------\n');
```

```
fprintf(fid1,'Associative desorption\n');
fprintf(fid1,'Adsorbed: *O\n');
fprintf(fid1,'Gas: O2\n');
fprintf(fid1,'Preexponential factor: %#e\n',par(3));
fprintf(fid1,'Activation energy: %#e\n',par(4));
fprintf(fid1,' 3) ----------------------------------\n');
fprintf(fid1,'Unimolecular adsorption\n');
fprintf(fid1,'Gas: CO\n');
fprintf(fid1,'Adsorbed: *CO\n');
fprintf(fid1,' 4) ----------------------------------\n');
fprintf(fid1,'Unimolecular desorption\n');
fprintf(fid1,'Adsorbed: *CO\n');
fprintf(fid1,'Gas: CO\n');
fprintf(fid1,'Preexponential factor: %#e\n',par(5));
fprintf(fid1,'Activation energy: %#e\n',par(6));
fprintf(fid1,' 5) ----------------------------------\n');
fprintf(fid1,'Bimolecular surface reaction 1\n');
fprintf(fid1,'Adsorbed,1: *CO\n');
fprintf(fid1,'Adsorbed,2: *O\n');
fprintf(fid1,'Gaseous products: 1\n');
fprintf(fid1,'Gas: CO2\n');
fprintf(fid1,'Preexponential factor: %#e\n',par(7));
fprintf(fid1,'Activation energy: %#e\n',par(8));
fprintf(fid1,' 6) ----------------------------------\n');
fprintf(fid1,'Bimolecular surface reaction 1\n');
fprintf(fid1,'Adsorbed,1: *CO\n');
fprintf(fid1,'Adsorbed,2: *BSS\n');
fprintf(fid1,'Gaseous products: 1\n');
fprintf(fid1,'Gas: CO2\n');
fprintf(fid1,'Preexponential factor: %#e\n',par(9));
fprintf(fid1,'Activation energy: 0.0\n');
fprintf(fid1,' 7) ----------------------------------\n');
fprintf(fid1,'Associative desorption\n');
fprintf(fid1,'Adsorbed: *BSS\n');
fprintf(fid1,'Gas: O2\n');
fprintf(fid1,'Preexponential factor: %#e\n',par(10));
fprintf(fid1,'Activation energy: 0.0\n');
fprintf(fid1,' 8) ----------------------------------\n');
fprintf(fid1,'Surface diffusion\n');
fprintf(fid1,'Adsorbed: *BSS\n');
fprintf(fid1,'Preexponential factor: %#e\n',par(11));
fprintf(fid1,'Activation energy: 0.0\n');
fprintf(fid1,'---------------------------------------\n');
fprintf(fid1,'*******************************************************\n');
fclose(fid1);

% writes 'lattice.dat' fortran input file
fid2=fopen('lattice.dat','w');
fprintf(fid2,'*************************************************************\n');
fprintf(fid2,'***  kMC LATTICE version 1.0: Lattice CREATION input file  ***\n');
fprintf(fid2,'*************************************************************\n');
fprintf(fid2,'\n');
fprintf(fid2,'Lattice size [X]: %d\n',lsize(1));
fprintf(fid2,'Lattice size [Y]: %d\n',lsize(2));
fprintf(fid2,'------------------------\n');
fprintf(fid2,'\n');
fprintf(fid2,'Number of adsorbed chemical species: 3\n');
fprintf(fid2,'---------------------------------------\n');
fprintf(fid2,'*CO   P0\n');
fprintf(fid2,'*O    P0\n');
fprintf(fid2,'*BSS  P0\n');
fprintf(fid2,'---------------------------------------\n');
fprintf(fid2,'\n');
fprintf(fid2,'Initial surface coverages:\n');
fprintf(fid2,'-------------------------\n');
for i=2:4
    fprintf(fid2,'%#e\n',theta0(i));
end
fprintf(fid2,'-------------------------\n');
fprintf(fid2,'*************************************************************\n');
fclose(fid2);
```

```
% writes 'control.dat' fortran input file
fid3=fopen('control.dat','w');
fprintf(fid3,'*****************************************************\n');
fprintf(fid3,'***  kMC LATTICE version 3.0: Control input file  ***\n');
fprintf(fid3,'*****************************************************\n');
fprintf(fid3,'\n');
fprintf(fid3,'Random seed:\n');
fprintf(fid3,'------------\n');
fprintf(fid3,'-100000\n');
fprintf(fid3,'\n');
fprintf(fid3,'Number of simulations:\n');
fprintf(fid3,'----------------------\n');
fprintf(fid3,'3\n');
fprintf(fid3,'\n');
fprintf(fid3,'Initial time, time increment, end time:\n');
fprintf(fid3,'---------------------------------------\n');
fprintf(fid3,'%#e\n',Init_time);
fprintf(fid3,'%#e\n',time_incr);
fprintf(fid3,'%#e\n',End_time);
fprintf(fid3,'\n');
fprintf(fid3,'Time step for every configuration storage:\n');
fprintf(fid3,'-----------------------------------------\n');
fprintf(fid3,'0.001\n');
fprintf(fid3,'\n');
fprintf(fid3,'Time interval for boundary flux updates:\n');
fprintf(fid3,'----------------------------------------\n');
fprintf(fid3,'0.00001\n');
fprintf(fid3,'\n');
fprintf(fid3,'Boundary conditions, in/outgoing particles:\n');
fprintf(fid3,'-----------------------------------------\n');
fprintf(fid3,'Top:\n');
fprintf(fid3,'-------\n');
fprintf(fid3,'NATURAL\n');
fprintf(fid3,'source\n');
fprintf(fid3,'0\n');
fprintf(fid3,'0\n');
fprintf(fid3,'%d\n',BSS_in(1));
fprintf(fid3,'-------------\n');
fprintf(fid3,'Bottom:\n');
fprintf(fid3,'-------\n');
fprintf(fid3,'NATURAL\n');
fprintf(fid3,'source\n');
fprintf(fid3,'0\n');
fprintf(fid3,'0\n');
fprintf(fid3,'%d\n',BSS_in(2));
fprintf(fid3,'-------------\n');
fprintf(fid3,'Left:\n');
fprintf(fid3,'-------\n');
fprintf(fid3,'NATURAL\n');
fprintf(fid3,'source\n');
fprintf(fid3,'0\n');
fprintf(fid3,'0\n');
fprintf(fid3,'%d\n',BSS_in(3));
fprintf(fid3,'-------------\n');
fprintf(fid3,'Right:\n');
fprintf(fid3,'-------\n');
fprintf(fid3,'NATURAL\n');
fprintf(fid3,'source\n');
fprintf(fid3,'0\n');
fprintf(fid3,'0\n');
fprintf(fid3,'0\n');
fprintf(fid3,'-----------------------------------------\n');
fprintf(fid3,'\n');
fprintf(fid3,'Time interval for DOMAIN Generation/Consumption updates:\n');
fprintf(fid3,'-------------------------------------------------------\n');
fprintf(fid3,'0.0001\n');
fprintf(fid3,'\n');
fprintf(fid3,'Enable DOMAIN Generation/Consumption (y/n): n\n');
fprintf(fid3,'-------------------------------------------\n');
fprintf(fid3,'0\n');
```

```
fprintf(fid3,'0\n');
fprintf(fid3,'0\n');
fprintf(fid3,'-------------------------------------------\n');
fprintf(fid3,'*******************************************************\n');
fclose(fid3);

% execution of the kMC simulation
ifail=system('./Single_kMC.exe');
if ifail~=0,
  fprintf(1,'ifail=%d\n',ifail); error 'kmc init simulator caused an error'
end

% get values from rate and coverage .OUT files
load -ascii COVERAGES.OUT
load -ascii RATES.OUT

% clean input & output files
ifail=system('rm *.OUT *.dat');
if ifail~=0,
    fprintf(1,'ifail=%d\n',ifail); error 'error while cleaning input & output files'
end

% get the last values of coverage and rate profiles as well as end time
theta=COVERAGES(:,2:end);
r=RATES(:,2:end);
t=COVERAGES(:,1);

end
```

A.2.3.1 kMC Fortran files (main program, subroutines & variables)

```
*******************************************************************************
    PROGRAM Single_kMC
*-----------------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: i,k,nsim
    INTEGER :: comptt,mu,reactions_gap
    REAL(KIND=8) :: sum,gamma_tot,tau
    REAL(KIND=8) :: ran2,ran
*-----------------------------------------------------------------------------*

    WRITE(*,*)
    WRITE(*,*)
    WRITE(*,'(''*************************************************************'')')
    WRITE(*,*)
    WRITE(*,'(''**********               SIMULATION STARTED            **********'')')
    WRITE(*,*)
    WRITE(*,'(''*************************************************************'')')
    WRITE(*,'(''>                              Single_kMC.exe     <'')')
    WRITE(*,'(''>                Developed by Ioannis S. Fragkopoulos   <'')')
    WRITE(*,'(''>                      Based on codes of F. Goujon      <'')')
    WRITE(*,'(''>               Single kMC Lattice with external FLUXES     <'')')
    WRITE(*,'(''>      and particle Generation/Consumption on/from Lattice   <'')')
    WRITE(*,'(''>         Reese et al., J. Comput. Phys., 2001, 173, 302-321     <'')')
    WRITE(*,'(''*************************************************************'')')
    WRITE(*,*)

    CALL CPU_TIME(cpu_begin)

    CALL read_init_lattice ()
```

```
      CALL read_parameters ()

      CALL read_control ()

      CALL init_lattice ()

      CALL init_kreac ()

      theta(:,:) = 0.0
      rate(:,:) = 0.0
      NBGAS(:) = 0

      DO nsim=1,nb_simul

       CALL init_zero ()

       CALL init_class ()

       CALL read_lattice ()

       CALL init_class_sites ()

       IF (nsim.eq.1) CALL write_classes ()

       WRITE(*,'(''-------------------------------------------------'')')
       WRITE(*,'('' Simulation:'',i5,''/'',i3)')nsim,nb_simul
       WRITE(*,'('' Time(s) Number of occupied sites'')')
       WRITE(*,'('' --------- -------------------------- '')')

       comptt = 0

       DO WHILE (time.lt.end_time)

        IF ((bound_fluxes).and.(time.gt.time_fluxupdate)) THEN

                CALL update_bound_fluxes ()
                CALL make_bound_fluxes ()

        ENDIF

        IF ((bound_fluxes).and.(time.gt.btime_next)) THEN

                CALL make_bound_fluxes ()

        ENDIF

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

         CALL update_domain_fluxes ()
         CALL make_domain_fluxes ()

        ENDIF

        IF ((domain_fluxes).and.(time.gt.dom_time_next)) THEN

         CALL make_domain_fluxes ()

        ENDIF

        CALL calc_proba ()
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

        gamma_tot = 0.0

        DO i=1,nb_reac
         gamma_tot = gamma_tot+gamma(i)
```

```
      ENDDO

      IF (gamma_tot.gt.1e-7) THEN

       tau = 1.0/(gamma_tot*REAL(nb_sites,8))
     &          *LOG(1/ran2(idum))

       mu = 1
       sum = 0.0
       ran = ran2(idum)*REAL(gamma_tot,8)

       DO i=1,nb_reac-1

        sum = sum+gamma(i)

        IF ((ran.gt.sum).and.(ran.lt.sum+gamma(i+1))) THEN

          mu = i+1

        ENDIF

       ENDDO

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

       IF ((type_reac(mu).eq.1).or.(type_reac(mu).eq.2).or.
     &      (type_reac(mu).eq.8).or.(type_reac(mu).eq.9)) THEN

        CALL make_reaction_1site(mu)

       ELSE

        CALL make_reaction_2site(mu)

       ENDIF

       time = time+tau

       IF (clean) THEN
         CALL init_class_sites ()
       ENDIF

      ELSE

       time = end_time

       WRITE(*,*)
       WRITE(*,'(''-------------------------------------------------'')')
       WRITE(*,'(''                 WARNING!                    '')')
       WRITE(*,'('' No reaction is possible to take place      '')')
       WRITE(*,'('' Total Transition Probability is very low   '')')
       WRITE(*,'(''-------------------------------------------------'')')
       WRITE(*,*)

      ENDIF

      CALL write_slabs ()

      comptt = comptt+1
      IF (mod(comptt,100).eq.0) THEN
       WRITE(*,'(e14.6,7i8)')time,(NBADS(k),k=0,nb_ads)
      ENDIF

      IF (time.ge.ttt*conf_tstep) THEN
       CALL write_conf_files ()
       ttt = ttt + 1
```

```
        END IF

      ENDDO

    ENDDO

    CALL write_configuration ()

    CALL write_nbreaction ()

    CALL write_ttt ()

    CALL write_coverage ()

    CALL write_rates ()

    CALL CPU_TIME(cpu_end)

    time_seconds = INT(cpu_end-cpu_begin)
    time_minutes = time_seconds/60
    time_hours = time_minutes/60
    time_minutes = time_minutes-time_hours*60
    time_seconds = time_seconds-time_hours*3600-time_minutes*60

    WRITE(*,*)

    IF ((time_minutes.lt.10).and.(time_seconds.lt.10)) THEN

      WRITE(*,'(''Total simulation time: '',i3,'':0'',i1,'':0'',i1)')
     &        time_hours,time_minutes,time_seconds

    ELSE IF ((time_minutes.lt.10).and.(time_seconds.ge.10)) THEN

      WRITE(*,'(''Total simulation time: '',i3,'':0'',i1,'':'',i2)')
     &        time_hours,time_minutes,time_seconds

    ELSE IF ((time_minutes.ge.10).and.(time_seconds.lt.10)) THEN

      WRITE(*,'(''Total simulation time: '',i3,'':'',i2,'':0'',i1)')
     &        time_hours,time_minutes,time_seconds

    ELSE

      WRITE(*,'(''Total simulation time: '',i3,'':'',i2,'':'',i2)')
     &        time_hours,time_minutes,time_seconds

    ENDIF

    WRITE(*,*)
    WRITE(*,'(''************************************************************'')')
    WRITE(*,'(''>                                  Single_kMC.exe       <'')')
    WRITE(*,'(''>                    Developed by Ioannis S. Fragkopoulos       <'')')
    WRITE(*,'(''>                           Based on codes of F. Goujon        <'')')
    WRITE(*,'(''>                  Single kMC Lattice with external FLUXES        <'')')
    WRITE(*,'(''>       and particle Generation/Consumption on/from Lattice     <'')')
    WRITE(*,'(''>           Reese et al., J. Comput. Phys., 2001, 173, 302-321     <'')')
    WRITE(*,'(''************************************************************'')')
    WRITE(*,*)
    WRITE(*,'(''**********                SIMULATION COMPLETED             **********'')')
    WRITE(*,*)
    WRITE(*,'(''************************************************************'')')
    WRITE(*,*)

    END PROGRAM
*-----------------------------------------------------------------------*
*************************************************************************
```

```
*******************************************************************************
    SUBROUTINE neighbour (xsite,ysite,xneigh,yneigh,neigh)
*------------------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: xsite,ysite,xneigh,yneigh,neigh
*------------------------------------------------------------------------------*
    xneigh = xsite
    yneigh = ysite

    IF (neigh.eq.top) yneigh = yneigh+1
    IF (neigh.eq.bottom) yneigh = yneigh-1
    IF (neigh.eq.left) xneigh = xneigh-1
    IF (neigh.eq.right) xneigh = xneigh+1

    IF (boundary(top).eq.'PERIODIC') THEN

      IF (yneigh.eq.lsize(y)+1) yneigh = 1
      IF (yneigh.eq.0) yneigh = lsize(y)

    ENDIF

    IF (boundary(left).eq.'PERIODIC') THEN

      IF (xneigh.eq.0) xneigh = lsize(x)
      IF (xneigh.eq.lsize(x)+1) xneigh = 1

    ENDIF

    RETURN
    END
*------------------------------------------------------------------------------*
*******************************************************************************

*******************************************************************************
    SUBROUTINE make_bound_fluxes ()
*------------------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: i,k,side,rsite,xn,yn
    INTEGER,DIMENSION(0:nb_ads_max) :: nbB
    REAL(KIND=8) :: ran2
    LOGICAL :: ok
*------------------------------------------------------------------------------*
    DO side=top,right

      nbB(:) = 0

      IF ((side.eq.top).and.(source(side))) THEN

        DO i=1,lsize(x)
          nbB(lattice(i,lsize(y))) = nbB(lattice(i,lsize(y)))+1
        ENDDO

        DO i=1,nb_ads

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

          IF (nb_left(side,i).gt.0) THEN

            IF (nbB(0).gt.0) THEN

              ok = .false.
```

```
        DO WHILE (.not.ok)

          rsite = INT(ran2(idum)*REAL(lsize(x),8))+1

          IF (lattice(rsite,lsize(y)).eq.0) THEN

            lattice(rsite,lsize(y)) = i
            NBADS(i) = NBADS(i)+1
            NBADS(0) = NBADS(0)-1
            nbB(0) = nbB(0)-1

            nb_left(side,i) = nb_left(side,i)-1
            incoming(side,i) = incoming(side,i)-1

            IF ((nbB(0).eq.0).or.(nb_left(side,i).eq.0)
     &                         .or.(incoming(side,i).eq.0)) ok = .true.

            CALL update_class_onesite(rsite,lsize(y))
            CALL update_class_twosite(rsite,lsize(y))

            DO k=top,right
              CALL neighbour(rsite,lsize(y),xn,yn,k)
              CALL update_class_twosite(xn,yn)
            ENDDO

          ENDIF

        ENDDO

      ENDIF

    ELSE IF (nb_left(side,i).lt.0) THEN

      IF (nbB(i).gt.0) THEN

        ok = .false.

        DO WHILE (.not.ok)

          rsite = INT(ran2(idum)*REAL(lsize(x),8))+1

          IF (lattice(rsite,lsize(y)).eq.i) THEN

            lattice(rsite,lsize(y)) = 0
            NBADS(i) = NBADS(i)-1
            NBADS(0) = NBADS(0)+1
            nbB(i) = nbB(i)-1

            nb_left(side,i) = nb_left(side,i)+1
            incoming(side,i) = incoming(side,i)+1

            IF ((nbB(i).eq.0).or.(nb_left(side,i).eq.0)
     &                         .or.(incoming(side,i).eq.0)) ok = .true.

            CALL update_class_onesite(rsite,lsize(y))
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

            DO k=top,right
              CALL neighbour(rsite,lsize(y),xn,yn,k)
              CALL update_class_twosite(xn,yn)
            ENDDO

          ENDIF

        ENDDO
```

```
          ENDIF

        ENDIF

      ENDIF

    ENDDO

  ELSE IF ((side.eq.bottom).and.(source(side))) THEN

    DO i=1,lsize(x)
      nbB(lattice(i,1)) = nbB(lattice(i,1))+1
    ENDDO

    DO i=1,nb_ads

      IF ((nb_left(side,i).ne.0).and.(incoming(side,i).ne.0)) THEN

        IF (nb_left(side,i).gt.0) THEN

          IF (nbB(0).gt.0) THEN

            ok = .false.

            DO WHILE (.not.ok)

              rsite = INT(ran2(idum)*REAL(lsize(x),8))+1

              IF (lattice(rsite,1).eq.0) THEN

                lattice(rsite,1) = i
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                NBADS(0) = NBADS(0)-1
                nbB(0) = nbB(0)-1

                nb_left(side,i) = nb_left(side,i)-1
                incoming(side,i) = incoming(side,i)-1

                IF ((nbB(0).eq.0).or.(nb_left(side,i).eq.0)
   &                             .or.(incoming(side,i).eq.0)) ok = .true.

                CALL update_class_onesite(rsite,1)
                CALL update_class_twosite(rsite,1)

                DO k=top,right
                  CALL neighbour(rsite,1,xn,yn,k)
                  CALL update_class_twosite(xn,yn)
                ENDDO

              ENDIF

            ENDDO

          ENDIF

        ELSE IF (nb_left(side,i).lt.0) THEN

          IF (nbB(i).gt.0) THEN

            ok = .false.

            DO WHILE (.not.ok)

              rsite = INT(ran2(idum)*REAL(lsize(x),8))+1

              IF (lattice(rsite,1).eq.i) THEN
```

```
            lattice(rsite,1) = 0
            NBADS(i) = NBADS(i)-1
            NBADS(0) = NBADS(0)+1
            nbB(i) = nbB(i)-1

            nb_left(side,i) = nb_left(side,i)+1
            incoming(side,i) = incoming(side,i)+1

            IF ((nbB(i).eq.0).or.(nb_left(side,i).eq.0)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

            CALL update_class_onesite(rsite,1)
            CALL update_class_twosite(rsite,1)

            DO k=top,right
              CALL neighbour(rsite,1,xn,yn,k)
              CALL update_class_twosite(xn,yn)
            ENDDO

          ENDIF

        ENDDO

      ENDIF

    ENDIF

  ENDIF

ENDDO

ELSE IF ((side.eq.left).and.(source(side))) THEN

DO i=1,lsize(y)
  nbB(lattice(1,i)) = nbB(lattice(1,i))+1
ENDDO

DO i=1,nb_ads

  IF ((nb_left(side,i).ne.0).and.(incoming(side,i).ne.0)) THEN

    IF (nb_left(side,i).gt.0) THEN

      IF (nbB(0).gt.0) THEN

        ok = .false.

        DO WHILE (.not.ok)

          rsite = INT(ran2(idum)*REAL(lsize(y),8))+1

          IF (lattice(1,rsite).eq.0) THEN

            lattice(1,rsite) = i
            NBADS(i) = NBADS(i)+1
            NBADS(0) = NBADS(0)-1
            nbB(0) = nbB(0)-1

            nb_left(side,i) = nb_left(side,i)-1
            incoming(side,i) = incoming(side,i)-1

            IF ((nbB(0).eq.0).or.(nb_left(side,i).eq.0)
   &                     .or.(incoming(side,i).eq.0)) ok = .true.

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
              CALL update_class_twosite(1,rsite)

              DO k=top,right
               CALL neighbour(1,rsite,xn,yn,k)
               CALL update_class_twosite(xn,yn)
              ENDDO

             ENDIF

            ENDDO

           ENDIF

          ELSE IF (nb_left(side,i).lt.0) THEN

           IF (nbB(i).gt.0) THEN

            ok = .false.

            DO WHILE (.not.ok)

             rsite = INT(ran2(idum)*REAL(lsize(y),8))+1

             IF (lattice(1,rsite).eq.i) THEN

              lattice(1,rsite) = 0
              NBADS(i) = NBADS(i)-1
              NBADS(0) = NBADS(0)+1
              nbB(i) = nbB(i)-1

              nb_left(side,i) = nb_left(side,i)+1
              incoming(side,i) = incoming(side,i)+1

              IF ((nbB(i).eq.0).or.(nb_left(side,i).eq.0)
&                              .or.(incoming(side,i).eq.0)) ok = .true.

              CALL update_class_onesite(1,rsite)
              CALL update_class_twosite(1,rsite)

              DO k=top,right
               CALL neighbour(1,rsite,xn,yn,k)
               CALL update_class_twosite(xn,yn)
              ENDDO

             ENDIF

            ENDDO

           ENDIF

          ENDIF

         ENDIF

        ENDDO

       ELSE IF ((side.eq.right).and.(source(side))) THEN

        DO i=1,lsize(y)
         nbB(lattice(lsize(x),i)) = nbB(lattice(lsize(x),i))+1
        ENDDO

        DO i=1,nb_ads

         IF ((nb_left(side,i).ne.0).and.(incoming(side,i).ne.0)) THEN
```

```
        IF (nb_left(side,i).gt.0) THEN

         IF (nbB(0).gt.0) THEN

          ok = .false.

          DO WHILE (.not.ok)

           rsite = INT(ran2(idum)*REAL(lsize(y),8))+1

           IF (lattice(lsize(x),rsite).eq.0) THEN

            lattice(lsize(x),rsite) = i
            NBADS(i) = NBADS(i)+1
            NBADS(0) = NBADS(0)-1
            nbB(0) = nbB(0)-1

            nb_left(side,i) = nb_left(side,i)-1
            incoming(side,i) = incoming(side,i)-1

            IF ((nbB(0).eq.0).or.(nb_left(side,i).eq.0)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

            CALL update_class_onesite(lsize(x),rsite)
            CALL update_class_twosite(lsize(x),rsite)

            DO k=top,right
             CALL neighbour(lsize(x),rsite,xn,yn,k)
             CALL update_class_twosite(xn,yn)
            ENDDO

           ENDIF

          ENDDO

         ENDIF

        ELSE IF (nb_left(side,i).lt.0) THEN

         IF (nbB(i).gt.0) THEN

          ok = .false.

          DO WHILE (.not.ok)

           rsite = INT(ran2(idum)*REAL(lsize(y),8))+1

           IF (lattice(lsize(x),rsite).eq.i) THEN

            lattice(lsize(x),rsite) = 0
            NBADS(i) = NBADS(i)-1
            NBADS(0) = NBADS(0)+1
            nbB(i) = nbB(i)-1

            nb_left(side,i) = nb_left(side,i)+1
            incoming(side,i) = incoming(side,i)+1

            IF ((nbB(i).eq.0).or.(nb_left(side,i).eq.0)
   &                  .or.(incoming(side,i).eq.0)) ok = .true.

            CALL update_class_onesite(lsize(x),rsite)
            CALL update_class_twosite(lsize(x),rsite)

            DO k=top,right
             CALL neighbour(lsize(x),rsite,xn,yn,k)
             CALL update_class_twosite(xn,yn)
```

```
              ENDDO

             ENDIF

            ENDDO

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

          ENDIF

         ENDIF

        ENDDO

       ENDIF

     ENDDO

     DO side=top,right
      DO i=1,nb_ads
        nb_left(side,i) = nb_left(side,i)+nb_to_add(side,i)
      ENDDO
     ENDDO

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
     t_b=t_b+1

     RETURN
     END
*-------------------------------------------------------------------------*
*****************************************************************************

*****************************************************************************
     SUBROUTINE update_bound_fluxes ()
*-------------------------------------------------------------------------*
     IMPLICIT NONE
     INCLUDE 'variables.inc'

     INTEGER :: i,side
*-------------------------------------------------------------------------*
     DO side=top,right

      IF (source(side)) THEN
       DO i=1,nb_ads
        incoming(side,i) = incoming0(side,i)
       ENDDO
      ENDIF

     ENDDO

     DO side=top,right

      IF (source(side)) THEN

       DO i=1,nb_ads

        IF (incoming(side,i).eq.0) THEN

          nb_left(side,i) = 0
          nb_to_add(side,i) = nb_left(side,i)

        ELSEIF (incoming(side,i).gt.0) THEN

          nb_left(side,i) = CEILING(REAL(incoming(side,i),8)*int_boundary/time_boundary)
          nb_to_add(side,i) = nb_left(side,i)
```

```
        ELSE

          nb_left(side,i) = INT(REAL(incoming(side,i),8)*int_boundary/time_boundary)-1
          nb_to_add(side,i) = nb_left(side,i)

        ENDIF

       ENDDO

      ENDIF

    ENDDO

    time_fluxup_slots = time_fluxup_slots + 1
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

    RETURN
    END
*-------------------------------------------------------------------------------*
*********************************************************************************

*********************************************************************************
    SUBROUTINE make_domain_fluxes ()
*-------------------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: i,j,k,xn,yn
    INTEGER :: xcoord,ycoord
    REAL(KIND=8) :: ran2
    LOGICAL :: ok
*-------------------------------------------------------------------------------*
    DO i=1,nb_ads

      IF ((dom_nb_left(i).ne.0).and.(dom_incoming(i).ne.0)) THEN

        IF (dom_nb_left(i).gt.0) THEN

          IF (NBADS(0).gt.0) THEN

            ok = .false.

            DO WHILE (.not.ok)

              xcoord = INT(ran2(idum)*REAL(lsize(x),8))+1
              ycoord = INT(ran2(idum)*REAL(lsize(y),8))+1

              IF (lattice(xcoord,ycoord).eq.0) THEN

                lattice(xcoord,ycoord) = i
                NBADS(i) = NBADS(i)+1
                NBADS(0) = NBADS(0)-1

                dom_nb_left(i) = dom_nb_left(i)-1
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

                IF ((NBADS(0).eq.0).or.(dom_nb_left(i).eq.0).or.
   &                          (dom_incoming(i).eq.0)) ok = .true.

                CALL update_class_onesite(xcoord,ycoord)
                CALL update_class_twosite(xcoord,ycoord)

                DO k=top,right
                  CALL neighbour(xcoord,ycoord,xn,yn,k)
                  CALL update_class_twosite(xn,yn)
                ENDDO
```

```
        ENDIF

       ENDDO

      ENDIF

     ELSEIF (dom_nb_left(i).lt.0) THEN

      IF (NBADS(i).gt.0) THEN

       ok = .false.

       DO WHILE (.not.ok)

        xcoord = INT(ran2(idum)*REAL(lsize(x),8))+1
        ycoord = INT(ran2(idum)*REAL(lsize(y),8))+1

        IF (lattice(xcoord,ycoord).eq.i) THEN

         lattice(xcoord,ycoord) = i
         NBADS(i) = NBADS(i)-1
         NBADS(0) = NBADS(0)+1

         dom_nb_left(i) = dom_nb_left(i)+1
         dom_incoming(i) = dom_incoming(i)+1

         IF ((NBADS(i).eq.0).or.(dom_nb_left(i).eq.0).or.
   &                  (dom_incoming(i).eq.0)) ok = .true.

         CALL update_class_onesite(xcoord,ycoord)
         CALL update_class_twosite(xcoord,ycoord)

         DO k=top,right
          CALL neighbour(xcoord,ycoord,xn,yn,k)
          CALL update_class_twosite(xn,yn)
         ENDDO

        ENDIF

       ENDDO

      ENDIF

     ENDIF

    ENDIF

   ENDDO

   DO i=1,nb_ads
     dom_nb_left(i) = dom_nb_left(i)+dom_nb_to_add(i)
   ENDDO

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   t_d=t_d+1

   RETURN
   END
*---------------------------------------------------------------------------*
*******************************************************************************

*******************************************************************************
   SUBROUTINE update_domain_fluxes ()
*---------------------------------------------------------------------------*
   IMPLICIT NONE
```

```
      INCLUDE 'variables.inc'

      INTEGER :: i
*------------------------------------------------------------------------------*
      DO i=1,nb_ads
       dom_incoming(i) = dom_incoming0(i)
      ENDDO

      DO i=1,nb_ads

       IF (dom_incoming(i).eq.0) THEN

        dom_nb_left(i) = 0
        dom_nb_to_add(i) = dom_nb_left(i)

       ELSEIF (dom_incoming(i).gt.0) THEN

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
        dom_nb_to_add(i) = dom_nb_left(i)

       ELSE

        dom_nb_left(i) = INT(REAL(dom_incoming(i),8)*dom_int_boundary/time_domain)-1
        dom_nb_to_add(i) = dom_nb_left(i)

       ENDIF

      ENDDO

      dom_time_fluxup_slots = dom_time_fluxup_slots + 1
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

      RETURN
      END
*------------------------------------------------------------------------------*
********************************************************************************

********************************************************************************
      LOGICAL FUNCTION outside (xsite,ysite)
*------------------------------------------------------------------------------*
      IMPLICIT NONE
      INCLUDE 'variables.inc'

      INTEGER :: xsite,ysite
*------------------------------------------------------------------------------*
      outside = .false.

      IF ((xsite.le.0).or.(xsite.ge.lsize(x)+1).or.
     &   (ysite.le.0).or.(ysite.ge.lsize(y)+1)) outside = .true.

      RETURN
      END
*------------------------------------------------------------------------------*
********************************************************************************

********************************************************************************
      SUBROUTINE init_lattice ()
*------------------------------------------------------------------------------*
      IMPLICIT NONE
      INCLUDE 'variables.inc'

      INTEGER :: i,j,k
      REAL(KIND=8) :: ran2,ran
      INTEGER:: xlatt,ylatt
      INTEGER:: nb_rest_tot,sum
      INTEGER,DIMENSION(nb_ads_max):: nb_rest
```

```
*-------------------------------------------------------------------------------*
    finish=.false.
    lattice(:,:) = 0
    nb_rest(:)=0

       DO WHILE (.not.finish)

            xlatt=INT(ran2(idum)*REAL(lsize(x),8))+1
            ylatt=INT(ran2(idum)*REAL(lsize(y),8))+1

            IF (lattice(xlatt,ylatt).eq.0) THEN

             nb_rest_tot=0

             DO i=1,nb_ads
               nb_rest(i)=(NB_fin(i)-NB(i))*priority(i)
               nb_rest_tot=nb_rest_tot+nb_rest(i)
             ENDDO

             IF (nb_rest_tot.ne.0) THEN

               insert_type=1
               sum=0
               ran=INT(ran2(idum)*REAL(nb_rest_tot,8))+1

               DO i=1,nb_ads-1

                 sum=sum+nb_rest(i)

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

               ENDDO

             ELSE

               insert_type=0

             ENDIF

             lattice(xlatt,ylatt)=insert_type
             NB(insert_type)=NB(insert_type)+1

            ENDIF

            finish=.true.

            DO i=1,nb_ads
             IF (NB(i).lt.NB_fin(i)) finish=.false.
            ENDDO

       ENDDO

    OPEN(21,FILE='./Configuration_Files/config_00000.out',STATUS='unknown')
    REWIND(21)

     WRITE(21,'(''                   Time: 0.0E+00  [sec]          '')')
     WRITE(21,'(''          ------------------------------------------'')')

     WRITE(21,'(i6)') lsize(x)
     WRITE(21,'(i6)') lsize(y)

     DO i=1,lsize(x)
      DO j=1,lsize(y)
        WRITE(21,'(2i4)') lattice(i,j)
      ENDDO
     ENDDO
```

```
    CLOSE(21)

    RETURN
    END
*-------------------------------------------------------------------------*
*****************************************************************************

*****************************************************************************
    SUBROUTINE init_zero ()
*-------------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: i,side,max,max2
    REAL(KIND=8)  :: ran2
*-------------------------------------------------------------------------*
    time = begin_time
    ttt = 1

    IF (bound_fluxes) THEN

     DO side=top,right

       IF (source(side)) THEN
        DO i=1,nb_ads
          incoming(side,i) = incoming0(side,i)
        ENDDO
       ENDIF

     ENDDO

     max = 1

     DO side=top,right

      IF (source(side)) THEN
        DO i=1,nb_ads
          IF (ABS(incoming(side,i)).gt.max) max = ABS(incoming(side,i))
        ENDDO
      ENDIF

     ENDDO

     int_boundary = time_boundary/(REAL(max,8))

     DO side=top,right

       IF (source(side)) THEN

        DO i=1,nb_ads

          IF (incoming(side,i).eq.0) THEN

            nb_left(side,i) = 0
            nb_to_add(side,i) = nb_left(side,i)

          ELSEIF (incoming(side,i).gt.0) THEN

            nb_left(side,i) = CEILING(REAL(incoming(side,i),8)*int_boundary/time_boundary)
            nb_to_add(side,i) = nb_left(side,i)

          ELSE

            nb_left(side,i) = INT(REAL(incoming(side,i),8)*int_boundary/time_boundary)-1
            nb_to_add(side,i) = nb_left(side,i)
```

```
          ENDIF

         ENDDO

        ENDIF

       ENDDO

       btime_next = begin_time+ran2(idum)*REAL(int_boundary,8)
       t_b=1
       time_fluxup_slots = 1
       time_fluxupdate = time_boundary

     ENDIF

     IF (domain_fluxes) THEN

       DO i=1,nb_ads
        dom_incoming(i) = dom_incoming0(i)
       ENDDO

       max2 = 1
       DO i=1,nb_ads
        IF (ABS(dom_incoming(i)).gt.max2) max2 = ABS(dom_incoming(i))
       ENDDO

       dom_int_boundary = time_domain/(REAL(max2,8))

       DO i=1,nb_ads

        IF (dom_incoming(i).eq.0) THEN

          dom_nb_left(i) = 0
          dom_nb_to_add(i) = dom_nb_left(i)

        ELSEIF (dom_incoming(i).gt.0) THEN

          dom_nb_left(i) = CEILING(REAL(dom_incoming(i),8)*dom_int_boundary/time_domain)
          dom_nb_to_add(i) = dom_nb_left(i)

        ELSE

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
          dom_nb_to_add(i) = dom_nb_left(i)

        ENDIF

       ENDDO

       dom_time_next= begin_time +ran2(idum)*REAL(dom_int_boundary,8)
       t_d=1
       dom_time_fluxup_slots = 1
       dom_time_fluxupdate = begin_time+time_domain

     ENDIF

     RETURN
     END
*-------------------------------------------------------------------------*
*****************************************************************************

*****************************************************************************
     SUBROUTINE init_class ()
*-------------------------------------------------------------------------*
     IMPLICIT NONE
```

```
      INCLUDE 'variables.inc'

      INTEGER :: i,j,p1,p2
*------------------------------------------------------------------------------*
      DO i=1,nb_reac

        IF (type_reac(i).eq.1) THEN

          IF (class(0,-1,-1).eq.0) THEN
            nb_class = nb_class+1
            class(0,-1,-1) = nb_class
            CALL store_class_coords(nb_class,0,-1,-1)
          ENDIF

        ELSE IF ((type_reac(i).eq.2).or.(type_reac(i).eq.8)) THEN

          p1 = param_reac(i,1)

          IF (class(p1,-1,-1).eq.0) THEN
            nb_class = nb_class+1
            class(p1,-1,-1) = nb_class
            CALL store_class_coords(nb_class,p1,-1,-1)
          ENDIF

        ELSE IF (type_reac(i).eq.3) THEN

          IF (class(0,-1,-1).eq.0) THEN
            nb_class = nb_class+1
            class(0,-1,-1) = nb_class
            CALL store_class_coords(nb_class,0,-1,-1)
          ENDIF

          DO j=1,4

            IF (class(0,0,j).eq.0) THEN
              nb_class = nb_class+1
              class(0,0,j) = nb_class
              CALL store_class_coords(nb_class,0,0,j)
            ENDIF

          ENDDO

        ELSE IF (type_reac(i).eq.4) THEN

          p1 = param_reac(i,1)

          IF (class(p1,-1,-1).eq.0) THEN
            nb_class = nb_class+1
            class(p1,-1,-1) = nb_class
            CALL store_class_coords(nb_class,p1,-1,-1)
          ENDIF

          DO j=1,4

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              nb_class = nb_class+1
              class(p1,p1,j) = nb_class
              CALL store_class_coords(nb_class,p1,p1,j)
            ENDIF

          ENDDO

        ELSE IF ((type_reac(i).eq.5).or.(type_reac(i).eq.10)
     &                            .or.(type_reac(i).eq.11)) THEN

          p1 = param_reac(i,1)
```

```
      p2 = param_reac(i,2)

      IF (class(p1,-1,-1).eq.0) THEN
       nb_class = nb_class+1
       class(p1,-1,-1) = nb_class
       CALL store_class_coords(nb_class,p1,-1,-1)
      ENDIF

      IF (class(p2,-1,-1).eq.0) THEN
       nb_class = nb_class+1
       class(p2,-1,-1) = nb_class
       CALL store_class_coords(nb_class,p2,-1,-1)
      ENDIF

      DO j=1,4

       IF (class(p1,p2,j).eq.0) THEN
         nb_class = nb_class+1
         class(p1,p2,j) = nb_class
         CALL store_class_coords(nb_class,p1,p2,j)
       ENDIF

       IF (class(p2,p1,j).eq.0) THEN
         nb_class = nb_class+1
         class(p2,p1,j) = nb_class
         CALL store_class_coords(nb_class,p2,p1,j)
       ENDIF

      ENDDO

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

      p1 = param_reac(i,1)

      IF (class(0,-1,-1).eq.0) THEN
       nb_class = nb_class+1
       class(0,-1,-1) = nb_class
       CALL store_class_coords(nb_class,0,-1,-1)
      ENDIF

      IF (class(p1,-1,-1).eq.0) THEN
       nb_class = nb_class+1
       class(p1,-1,-1) = nb_class
       CALL store_class_coords(nb_class,p1,-1,-1)
      ENDIF

      DO j=1,4

       IF (class(p1,0,j).eq.0) THEN
         nb_class = nb_class+1
         class(p1,0,j) = nb_class
         CALL store_class_coords(nb_class,p1,0,j)
       ENDIF

       IF (class(0,p1,j).eq.0) THEN
         nb_class = nb_class+1
         class(0,p1,j) = nb_class
         CALL store_class_coords(nb_class,0,p1,j)
       ENDIF

      ENDDO

     ELSE IF (type_reac(i).eq.9) THEN

      p1 = param_reac(i,2)
```

```
        IF (class(p1,-1,-1).eq.0) THEN
          nb_class = nb_class+1
          class(p1,-1,-1) = nb_class
          CALL store_class_coords(nb_class,p1,-1,-1)
        ENDIF

       ENDIF

      ENDDO

      nb_proba = 0
      proba_coord(:,:) = -1

      DO i=1,nb_class

       IF (class_coord(i,2).eq.-1) THEN

         nb_proba = nb_proba+1
         proba_coord(nb_proba,1) = class_coord(i,1)

       ELSE IF (class_coord(i,3).eq.1) THEN

         nb_proba = nb_proba+1
         proba_coord(nb_proba,1) = class_coord(i,1)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

       ENDIF

      ENDDO

      IF (nb_class.gt.nb_class_max) THEN
       WRITE(*,*)
       WRITE(*,'(''*** ERROR: number of classes too large:'')')
       WRITE(*,'('' - nb_class ='',i4)') nb_class
       WRITE(*,'('' - nb_class_max ='',i4)') nb_class_max
       WRITE(*,'(''You may increase nb_class_max and re-compile the program.'')')
       WRITE(*,'(''Program terminated.'')')
       WRITE(*,*)
       STOP
      ENDIF

      RETURN
      END
*-----------------------------------------------------------------------*
*************************************************************************

*************************************************************************
      SUBROUTINE init_class_sites ()
*-----------------------------------------------------------------------*
      IMPLICIT NONE
      INCLUDE 'variables.inc'

      INTEGER :: i,j
*-----------------------------------------------------------------------*
      nb_site_class(:) = 0
      realnb_site_class(:) = 0
      class_site(:,:,:) = 0

      site_clnb(:,:) = 0
      site_clid(:,:,:) = 0
      site_clsite(:,:,:) = 0

      enabled(:,:) = .false.
      clean = .false.
      gc = .false.
```

```
    DO i=1,lsize(x)
      DO j=1,lsize(y)

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
        CALL initclass_twosite(i,j)

      ENDDO
    ENDDO

    gc = .true.

    RETURN
    END
*-----------------------------------------------------------------------*
*************************************************************************

*************************************************************************
    SUBROUTINE initclass_onesite (xsite,ysite)
*-----------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: xsite,ysite
    INTEGER :: stype,ind
    REAL(KIND=8) :: proportion
    LOGICAL :: outside
*-----------------------------------------------------------------------*
    IF (.not.outside(xsite,ysite)) THEN

      stype = lattice(xsite,ysite)
      ind = class(stype,-1,-1)

      realnb_site_class(ind) = realnb_site_class(ind)+1
      nb_site_class(ind) = nb_site_class(ind)+1

      enabled(ind,nb_site_class(ind)) = .true.

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
      class_site(ind,nb_site_class(ind),y) = ysite

      IF (gc) THEN

        proportion = REAL(nb_site_class(ind),8)/REAL(nb_sites,8)

        IF (proportion.gt.1.0) THEN
          clean = .true.
        ENDIF

      ENDIF

      site_clnb(xsite,ysite) = site_clnb(xsite,ysite)+1
      site_clid(xsite,ysite,site_clnb(xsite,ysite)) = ind
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

      IF (site_clnb(xsite,ysite).gt.1e2) THEN
        clean = .true.
      ENDIF

    ENDIF

    RETURN
    END
*-----------------------------------------------------------------------*
*************************************************************************
```

```
********************************************************************************
      SUBROUTINE initclass_twosite (xsite,ysite)
*------------------------------------------------------------------------------*
      IMPLICIT NONE
      INCLUDE 'variables.inc'

      INTEGER :: xsite,ysite
      INTEGER :: k,n,side
      INTEGER :: stype,ind
      INTEGER :: xn,yn
      INTEGER :: nn,nb_types
      INTEGER,DIMENSION(4) :: nntype,nb_nntype
      LOGICAL :: new,outside,ok
      REAL(KIND=8) :: proportion
*------------------------------------------------------------------------------*
      IF (.not.outside(xsite,ysite)) THEN

        stype = lattice(xsite,ysite)
        nb_types = 0
        nntype(:) = -1
        nb_nntype(:) = 0

        DO n=top,right

          CALL neighbour(xsite,ysite,xn,yn,n)

          IF (.not.outside(xn,yn)) THEN

            nn = lattice(xn,yn)
            new = .true.

            DO k=1,nb_types

              IF (nntype(k).eq.nn) THEN
                new = .false.
                nb_nntype(k) = nb_nntype(k)+1
              ENDIF

            ENDDO

            IF (new) THEN
              nb_types = nb_types+1
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              nb_nntype(nb_types) = 1
            ENDIF

          ENDIF

        ENDDO

        DO k=1,nb_types

          ind = class(stype,nntype(k),nb_nntype(k))

          IF (ind.ne.0) THEN

            realnb_site_class(ind) = realnb_site_class(ind)+1
            nb_site_class(ind) = nb_site_class(ind)+1

            enabled(ind,nb_site_class(ind)) = .true.

            class_site(ind,nb_site_class(ind),x) = xsite
            class_site(ind,nb_site_class(ind),y) = ysite

            IF (gc) THEN
```

```
          proportion = REAL(nb_site_class(ind),8)/REAL(nb_sites,8)

          IF (proportion.gt.1.0) THEN
           clean = .true.
          ENDIF

         ENDIF

         site_clnb(xsite,ysite) = site_clnb(xsite,ysite)+1
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
         site_clsite(xsite,ysite,site_clnb(xsite,ysite)) = nb_site_class(ind)

         IF (site_clnb(xsite,ysite).gt.1e2) THEN
          clean = .true.
         ENDIF

        ENDIF

       ENDDO

     ENDIF

     RETURN
     END
*-----------------------------------------------------------------------*
*************************************************************************

*************************************************************************
     SUBROUTINE init_kreac ()
*-----------------------------------------------------------------------*
     IMPLICIT NONE
     INCLUDE 'variables.inc'

     INTEGER :: i
     REAL(KIND=8),DIMENSION(nb_gas) :: k_ads
*-----------------------------------------------------------------------*
     DO i=1,nb_gas

       k_ads(i) = sticking_gas(i)*(N_AVOG/site_density)*
    &                                press(i)/sqrt(2.0*PI*weight_gas(i)*R*temperature)

     ENDDO

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

       IF ((type_reac(i).eq.1).or.(type_reac(i).eq.3)) THEN

         k_reac(i) = k_ads(param_reac(i,1))

       ELSE IF (type_reac(i).eq.9) THEN

                k_reac(i) = k_ads(param_reac(i,1))*
    &                       dexp(-real(actenergy(i),8)/(R*temperature))

       ELSE

         k_reac(i) = preexp(i)*dexp(-real(actenergy(i),8)/(R*temperature))

       ENDIF

     ENDDO

     RETURN
     END
*-----------------------------------------------------------------------*
*************************************************************************
```

```
**************************************************************************
    SUBROUTINE calc_proba ()
*-------------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: i,j
    INTEGER :: s1,s2,ind
    REAL(KIND=8) :: numer
*-------------------------------------------------------------------------*
    proba(:,:) = 0.0

    DO i=1,nb_proba

     s1 = proba_coord(i,1)
     s2 = proba_coord(i,2)

     IF (s2.eq.-1) THEN

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
       proba(s1,s2) = REAL(realnb_site_class(ind),8)/REAL(nb_sites,8)

      ELSE

       numer = 0.0

       DO j=1,4

        ind = class(s2,s1,j)
        numer = numer+j*realnb_site_class(ind)

       ENDDO

       ind = class(s2,-1,-1)

       IF (realnb_site_class(ind).ne.0) THEN
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
       ENDIF

      ENDIF

    ENDDO

    RETURN
    END
*-------------------------------------------------------------------------*
**************************************************************************

**************************************************************************
    SUBROUTINE calc_gamma ()
*-------------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: i,p1,p2
*-------------------------------------------------------------------------*
    DO i=1,nb_reac

     IF (type_reac(i).eq.1) THEN

       gamma(i) = k_reac(i)*proba(0,-1)

     ELSE IF ((type_reac(i).eq.2).or.(type_reac(i).eq.8)) THEN

       p1 = param_reac(i,1)
       gamma(i) = k_reac(i)*proba(p1,-1)
```

```
      ELSE IF (type_reac(i).eq.3) THEN

       gamma(i) = k_reac(i)*proba(0,-1)*proba(0,0)

      ELSE IF (type_reac(i).eq.4) THEN

       p1 = param_reac(i,1)
       gamma(i) = k_reac(i)*proba(p1,-1)*proba(p1,p1)

      ELSE IF ((type_reac(i).eq.5).or.(type_reac(i).eq.10)
     &                  .or.(type_reac(i).eq.11)) THEN

       p1 = param_reac(i,1)
       p2 = param_reac(i,2)
       gamma(i) = k_reac(i)*(proba(p1,-1)*proba(p2,p1)+proba(p2,-1)*proba(p1,p2))

       IF (p1.eq.p2) gamma(i) = gamma(i)/2

      ELSE IF ((type_reac(i).eq.6).or.(type_reac(i).eq.7)) THEN

       p1 = param_reac(i,1)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

      ELSE IF (type_reac(i).eq.9) THEN

       p1 = param_reac(i,1)
       p2 = param_reac(i,2)
       gamma(i) = k_reac(i)*proba(p2,-1)

       WRITE(*,*)
       WRITE(*,*)'   Check if the the sticking coefficient of the gas species    '
       WRITE(*,*)'    that takes place in the "Reactive Adsorption" type of      '
       WRITE(*,*)'        reaction, is the proper one and is not mixed       '
       WRITE(*,*)'   with any other, utilised for normal adsorption purposes     '
       WRITE(*,*)

      ENDIF

     ENDDO

     RETURN
     END
*-----------------------------------------------------------------------------*
*******************************************************************************

*******************************************************************************
     SUBROUTINE make_reaction_1site (rnum)
*-----------------------------------------------------------------------------*
     IMPLICIT NONE
     INCLUDE 'variables.inc'

     INTEGER :: rnum
     INTEGER :: i,p1,p2,p3
     INTEGER :: ind,nsite,xn,yn
     INTEGER :: xx1,yy1
     REAL(KIND=8) :: ran2
     LOGICAL :: ok,outside
*-----------------------------------------------------------------------------*
     p1 = param_reac(rnum,1)
     p2 = param_reac(rnum,2)
     p3 = param_reac(rnum,3)

     IF (type_reac(rnum).eq.1) THEN

      ind = class(0,-1,-1)
```

```
      ok = .false.

      DO WHILE (.not.ok)

        nsite = INT(ran2(idum)*nb_site_class(ind))+1
        xx1 = class_site(ind,nsite,x)
        yy1 = class_site(ind,nsite,y)

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

      ENDDO

      lattice(xx1,yy1) = p2
      NBGAS(p1) = NBGAS(p1)-1
      NBADS(p2) = NBADS(p2)+1
      NBADS(0) = NBADS(0)-1

    ELSE IF (type_reac(rnum).eq.2) THEN

      ind = class(p1,-1,-1)
      ok = .false.

      DO WHILE (.not.ok)

        nsite = INT(ran2(idum)*nb_site_class(ind))+1
        xx1 = class_site(ind,nsite,x)
        yy1 = class_site(ind,nsite,y)

        IF (enabled(ind,nsite).and.(.not.outside(xx1,yy1))) ok = .true.

      ENDDO

      lattice(xx1,yy1) = 0
      NBADS(p1) = NBADS(p1)-1
      NBGAS(p2) = NBGAS(p2)+1
      NBADS(0) = NBADS(0)+1

    ELSE IF (type_reac(rnum).eq.8) THEN

      ind = class(p1,-1,-1)
      ok = .false.

      DO WHILE (.not.ok)

        nsite = INT(ran2(idum)*nb_site_class(ind))+1
        xx1 = class_site(ind,nsite,x)
        yy1 = class_site(ind,nsite,y)

        IF (enabled(ind,nsite).and.(.not.outside(xx1,yy1))) ok = .true.

      ENDDO

      lattice(xx1,yy1) = p2
      NBADS(p1) = NBADS(p1)-1
      NBADS(p2) = NBADS(p2)+1

    ELSE IF (type_reac(rnum).eq.9) THEN

      ind = class(p2,-1,-1)
      ok = .false.

      DO WHILE (.not.ok)

        nsite = INT(ran2(idum)*nb_site_class(ind))+1
        xx1 = class_site(ind,nsite,x)
        yy1 = class_site(ind,nsite,y)
```

```
       IF (enabled(ind,nsite).and.(.not.outside(xx1,yy1))) ok = .true.

      ENDDO

      lattice(xx1,yy1) = p3
      NBGAS(p1) = NBGAS(p1)-1
      NBADS(p2) = NBADS(p2)-1
      NBADS(p3) = NBADS(p3)+1

     ELSE

      WRITE(*,*)
      WRITE(*,'(''ERROR: '',i2,'' is not 1-site chemical reaction'')')rnum
      WRITE(*,'(''Program terminated.'')')
      WRITE(*,*)
      STOP

     ENDIF

     CALL update_class_onesite(xx1,yy1)
     CALL update_class_twosite(xx1,yy1)

     DO i=top,right
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
      CALL update_class_twosite(xn,yn)
     ENDDO

     RETURN
     END
*-------------------------------------------------------------------------*
*****************************************************************************

*****************************************************************************
     SUBROUTINE make_reaction_2site (rnum)
*-------------------------------------------------------------------------*
     IMPLICIT NONE
     INCLUDE 'variables.inc'

     INTEGER :: rnum,ind,side
     INTEGER :: i,j,k,p1,p2,p3,s1,s2
     INTEGER :: site1,site2
     INTEGER :: nbc, nsite, cl, neigh
     INTEGER,DIMENSION(2) :: suppr
     INTEGER :: xx1, yy1, xn, yn, xx2, yy2
     REAL(KIND=8) :: ran,ran2,sum
     INTEGER,DIMENSION(8) :: idclass
     INTEGER,DIMENSION(8) :: Cxym
     INTEGER :: Cxytot
     LOGICAL :: ok,outside
*-------------------------------------------------------------------------*
     idclass(:) = -1
     p1 = param_reac(rnum,1)
     p2 = param_reac(rnum,2)
     p3 = param_reac(rnum,3)

     IF (type_reac(rnum).eq.3) THEN

      s1 = 0
      s2 = 0

     ELSE IF (type_reac(rnum).eq.4) THEN

      s1 = p1
      s2 = p1
```

```
ELSE IF (type_reac(rnum).eq.5) THEN

 s1 = p1
 s2 = p2

ELSE IF (type_reac(rnum).eq.6) THEN

 s1 = p1
 s2 = 0

ELSE IF (type_reac(rnum).eq.7) THEN

 s1 = p1
 s2 = 0

ELSE IF (type_reac(rnum).eq.10) THEN

 s1 = p1
 s2 = p2

ELSE IF (type_reac(rnum).eq.11) THEN

 s1 = p1
 s2 = p2

ELSE

 WRITE(*,*)
 WRITE(*,'(''ERROR: '',i2,'' is not 2-site chemical reaction'')')rnum
 WRITE(*,'(''Program terminated.'')')
 WRITE(*,*)
 STOP

ENDIF

nbc = 0
Cxytot = 0

DO i=1,4

 ind = class(s1,s2,i)

 IF (realnb_site_class(ind).gt.0) THEN
  nbc = nbc+1
  idclass(nbc) = ind
  Cxym(nbc) = i*realnb_site_class(ind)
  Cxytot = Cxytot+Cxym(nbc)
 ENDIF

 ind = class(s2,s1,i)

 IF (realnb_site_class(ind).gt.0) THEN
  nbc = nbc+1
  idclass(nbc) = ind
  Cxym(nbc) = i*realnb_site_class(ind)
  Cxytot = Cxytot+Cxym(nbc)
 ENDIF

ENDDO

cl = 1
sum = 0.0
ran = ran2(idum)*REAL(Cxytot,8)

DO i=1,nbc-1
```

```
      sum = sum+REAL(Cxym(i),8)

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

    ENDDO

    ind = idclass(cl)
    ok = .false.

    DO WHILE (.not.ok)
     nsite = INT(ran2(idum)*nb_site_class(ind))+1
     IF (enabled(ind,nsite)) ok = .true.
    ENDDO

    xx1 = class_site(ind,nsite,x)
    yy1 = class_site(ind,nsite,y)
    site1 = lattice(xx1,yy1)
    site2 = class_coord(ind,2)

    neigh = INT(ran2(idum)*4)+1
    ok = .false.

    DO WHILE (.not.ok)

     neigh = mod(neigh,4)+1

     CALL neighbour(xx1,yy1,xn,yn,neigh)

     IF ((.not.outside(xn,yn)).and.
    &     (lattice(xn,yn).eq.site2)) ok = .true.

    ENDDO

    xx2 = xn
    yy2 = yn

    IF (type_reac(rnum).eq.3) THEN

     lattice(xx1,yy1) = p2
     lattice(xx2,yy2) = p2

     NBGAS(p1) = NBGAS(p1)-1
     NBADS(0) = NBADS(0)-2
     NBADS(p2) = NBADS(p2)+2

    ELSE IF (type_reac(rnum).eq.4) THEN

     lattice(xx1,yy1) = 0
     lattice(xx2,yy2) = 0

     NBADS(p1) = NBADS(p1)-2
     NBADS(0) = NBADS(0)+2
     NBGAS(p2) = NBGAS(p2)+1

    ELSE IF (type_reac(rnum).eq.5) THEN

     lattice(xx1,yy1) = 0
     lattice(xx2,yy2) = 0

     NBADS(p1) = NBADS(p1)-1
     NBADS(p2) = NBADS(p2)-1
     NBADS(0) = NBADS(0)+2

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
       k = param_reac(rnum,3+j)
       NBGAS(k) = NBGAS(k)+1
```

```
  ENDDO

ELSE IF (type_reac(rnum).eq.6) THEN

  IF (ran2(idum).lt.0.5) THEN

    lattice(xx1,yy1) = p2
    lattice(xx2,yy2) = p3

    NBADS(p1) = NBADS(p1)-1
    NBADS(0) = NBADS(0)-1
    NBADS(p2) = NBADS(p2)+1
    NBADS(p3) = NBADS(p3)+1

  ELSE

    lattice(xx1,yy1) = p3
    lattice(xx2,yy2) = p2

    NBADS(p1) = NBADS(p1)-1
    NBADS(0) = NBADS(0)-1
    NBADS(p2) = NBADS(p2)+1
    NBADS(p3) = NBADS(p3)+1

  ENDIF

ELSE IF (type_reac(rnum).eq.7) THEN

  IF (lattice(xx1,yy1).eq.p1) THEN

    lattice(xx1,yy1) = 0
    lattice(xx2,yy2) = p1

  ELSEIF (lattice(xx2,yy2).eq.p1) THEN

    lattice(xx1,yy1) = p1
    lattice(xx2,yy2) = 0

  ENDIF

ELSE IF (type_reac(rnum).eq.10) THEN

  IF (ran2(idum).lt.0.5) THEN

    lattice(xx1,yy1) = p3
    lattice(xx2,yy2) = 0

    NBADS(p1) = NBADS(p1)-1
    NBADS(p2) = NBADS(p2)-1
    NBADS(p3) = NBADS(p3)+1
    NBADS(0) = NBADS(0)+1

  ELSE

    lattice(xx1,yy1) = 0
    lattice(xx2,yy2) = p3

    NBADS(p1) = NBADS(p1)-1
    NBADS(p2) = NBADS(p2)-1
    NBADS(p3) = NBADS(p3)+1
    NBADS(0) = NBADS(0)+1

  ENDIF

ELSE
```

```
      IF (ran2(idum).lt.0.5) THEN

        lattice(xx1,yy1) = param_reac(rnum,3)
        lattice(xx2,yy2) = param_reac(rnum,4)

        NBADS(p1) = NBADS(p1)-1
        NBADS(p2) = NBADS(p2)-1
        NBADS(param_reac(rnum,3)) = NBADS(param_reac(rnum,3))+1
        NBADS(param_reac(rnum,4)) = NBADS(param_reac(rnum,4))+1

      ELSE

        lattice(xx1,yy1) = param_reac(rnum,4)
        lattice(xx2,yy2) = param_reac(rnum,3)

        NBADS(p1) = NBADS(p1)-1
        NBADS(p2) = NBADS(p2)-1
        NBADS(param_reac(rnum,3)) = NBADS(param_reac(rnum,3))+1
        NBADS(param_reac(rnum,4)) = NBADS(param_reac(rnum,4))+1

      ENDIF

    ENDIF

    CALL update_class_onesite(xx1,yy1)
    CALL update_class_onesite(xx2,yy2)

    DO i=top,right
      CALL neighbour(xx1,yy1,xn,yn,i)
      CALL update_class_twosite(xn,yn)
      CALL neighbour(xx2,yy2,xn,yn,i)
      CALL update_class_twosite(xn,yn)
    ENDDO

    RETURN
    END
*-----------------------------------------------------------------------------*
*******************************************************************************

*******************************************************************************
    SUBROUTINE update_class_onesite (xsite,ysite)
*-----------------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: xsite,ysite
    INTEGER :: i,ind,num
    LOGICAL :: outside
*-----------------------------------------------------------------------------*
    IF (.not.outside(xsite,ysite)) THEN

      DO i=1,site_clnb(xsite,ysite)
        ind = site_clid(xsite,ysite,i)
        num = site_clsite(xsite,ysite,i)

        IF (enabled(ind,num).and.(class_coord(ind,2).eq.-1)) THEN
          enabled(ind,num) = .false.
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
        ENDIF

      ENDDO

      CALL initclass_onesite(xsite,ysite)

    ENDIF
```

```
      RETURN
      END
*-----------------------------------------------------------------------*
*************************************************************************

*************************************************************************
      SUBROUTINE update_class_twosite (xsite,ysite)
*-----------------------------------------------------------------------*
      IMPLICIT NONE
      INCLUDE 'variables.inc'

      INTEGER :: xsite,ysite
      INTEGER :: i,ind,num
      LOGICAL :: outside
*-----------------------------------------------------------------------*
      IF (.not.outside(xsite,ysite)) THEN

       DO i=1,site_clnb(xsite,ysite)
         ind = site_clid(xsite,ysite,i)
         num = site_clsite(xsite,ysite,i)

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
           enabled(ind,num) = .false.
           realnb_site_class(ind) = realnb_site_class(ind)-1
         ENDIF

       ENDDO

       CALL initclass_twosite(xsite,ysite)

      ENDIF

      RETURN
      END
*-----------------------------------------------------------------------*
*************************************************************************

*************************************************************************
      SUBROUTINE read_init_lattice ()
*-----------------------------------------------------------------------*
      IMPLICIT NONE
      INCLUDE 'variables.inc'

      INTEGER :: i,j,k,NB_tot
      CHARACTER :: pr*2,info*80
      LOGICAL :: existe
*-----------------------------------------------------------------------*
      INQUIRE(FILE='lattice.dat',EXIST=existe)

      IF (.not.existe) THEN

       WRITE(*,*)
       WRITE(*,'(''*** ERROR: Parameter file ''lattice.dat'' was not found'')')
       WRITE(*,'(''          Program terminated.'')')
       WRITE(*,*)
       STOP

      ENDIF

      OPEN(18,FILE='lattice.dat',STATUS='unknown')
      REWIND(18)

      READ(18,*)
      READ(18,'(a80)') info

      IF (info(26:28).ne.'1.0') THEN
```

```
 WRITE(*,*)
 WRITE(*,'(''*** ERROR: Version incompatibility "lattice.dat" '')')
 WRITE(*,'('' Program terminated.'')')
 WRITE(*,*)
 STOP

ENDIF

READ(18,*)
READ(18,*)
READ(18,'(17x,i10)') lsize(x)
READ(18,'(17x,i10)') lsize(y)
READ(18,*)

nb_sites = lsize(x)*lsize(y)

name(0) = '*   '

READ(18,*)
READ(18,'(36x,i10)')nb_ads
READ(18,*)

priority(:) = 1

DO i=1,nb_ads

 READ(18,*)name(i),pr

 IF (pr.eq.'P0') priority(i) = 1
 IF (pr.eq.'P1') priority(i) = 10
 IF (pr.eq.'P2') priority(i) = 1000
 IF (pr.eq.'P3') priority(i) = 100000
 IF (pr.eq.'P4') priority(i) = 10000000

ENDDO

READ(18,*)
READ(18,*)
READ(18,*)
READ(18,*)

NB_fin(:) = 0
NB(:) = 0
theta0(:) = 0
NB(0) = nb_sites
theta0(0) = 1.0

WRITE(*,'(''-------------------------------------------------'')')
WRITE(*,'('' Initially on lattice '')')
WRITE(*,'('' ---------------------------- '')')

DO i=1,nb_ads

 READ(18,*)theta0(i)

 theta0(0) = theta0(0)-theta0(i)
 NB_fin(i) = ANINT(theta0(i)*REAL(nb_sites,8))
 NB(0) = NB(0)-NB_fin(i)

 WRITE(*,'(''- Number of adsorbed species'',i2,''('',a4,'')'','' ='',i10)')i,name(i),NB_fin(i)

ENDDO

WRITE(*,'(''- Number of active empty sites('',a4,'')'','' ='',i10)')name(0),NB(0)
```

```
    NB_tot=NB(0)

    DO i=1,nb_ads

    NB_tot= NB_tot+NB_fin(i)

    ENDDO

    WRITE(*,'('' ----------  '')')
    WRITE(*,'('' Total number of lattice sites:'',i14)')NB_tot
    WRITE(*,'(''-------------------------------------------------'')')

    nb_class = 0
    class(:,:,:) = 0
    class_coord(:,:)= -1

    CLOSE(18)

    RETURN
    END
*-----------------------------------------------------------------------*
*************************************************************************

*************************************************************************
    SUBROUTINE read_parameters ()
*-----------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: i,j
    INTEGER :: nametype_ads,nametype_gas
    CHARACTER :: specname*4,info*80
    LOGICAL :: existe
*-----------------------------------------------------------------------*
    INQUIRE(FILE='parameters.dat',EXIST=existe)

    IF (.not.existe) THEN

     WRITE(*,*)
     WRITE(*,'(''*** ERROR: Parameter file "parameters.dat" was not found'')')
     WRITE(*,'('' Program terminated.'')')
     WRITE(*,*)
     STOP

    ENDIF

    OPEN(22,file='parameters.dat',status='unknown')
    REWIND(22)

    READ(22,*)
    READ(22,'(a80)') info

    IF (info(26:28).ne.'1.0') THEN

     WRITE(*,*)
     WRITE(*,'(''*** ERROR: Version incompatibility "parameters.dat" '')')
     WRITE(*,'('' Program terminated.'')')
     WRITE(*,*)
     STOP

    ENDIF

    READ(22,*)
    READ(22,*)
    READ(22,*)
    READ(22,*)
```

```
READ(22,'(e20.10)') temperature
READ(22,*)
READ(22,*)
READ(22,*)
READ(22,*)
READ(22,'(26x,e20.10)') site_density
READ(22,*)
READ(22,*)
READ(22,'(26x,i2)') nb_gas

IF (nb_gas.ge.6) THEN

 WRITE(*,*)
 WRITE(*,*)'*************************************************************************'
 WRITE(*,*)' WARNING! --> (nb_gas>5) You should change the write_rates subroutine '
 WRITE(*,*)'*************************************************************************'
 WRITE(*,'(''Program terminated.'')')
 WRITE(*,*)
 STOP

ENDIF

READ(22,*)

DO i=1,nb_gas

 READ(22,'(6x,a4)') name_gas(i)
 READ(22,'(27x,e20.10)') weight_gas(i)
 READ(22,'(22x,e20.10)') sticking_gas(i)
 READ(22,'(23x,e20.10)') press(i)
 READ(22,'(24x,e20.10)') rate0(i)
 READ(22,*)

ENDDO

READ(22,*)
READ(22,*)
READ(22,'(27x,i2)') nb_ads

IF (nb_ads.ge.6) THEN

 WRITE(*,*)
 WRITE(*,*)'*************************************************************************'
 WRITE(*,*)' WARNING! --> (nb_ads>5) You should change the write_coverage subroutine '
 WRITE(*,*)'*************************************************************************'
 WRITE(*,'(''Program terminated.'')')
 WRITE(*,*)
 STOP

ENDIF

READ(22,*)

DO i=1,nb_ads

 READ(22,'(6x,a4)') name_ads(i)

ENDDO

READ(22,*)
READ(22,*)
READ(22,*)

READ(22,'(20x,i2)') nb_reac

DO i=1,nb_reac
```

```
READ(22,*)
READ(22,'(a80)') info

IF (info(1:23).eq.'Unimolecular adsorption') THEN

 type_reac(i) = 1
 READ(22,'(5x,a4)') specname
 param_reac(i,1) = nametype_gas(specname)
 READ(22,'(10x,a4)') specname
 param_reac(i,2) = nametype_ads(specname)

ELSE IF (info(1:23).eq.'Unimolecular desorption') THEN

 type_reac(i) = 2
 READ(22,'(10x,a4)') specname
 param_reac(i,1) = nametype_ads(specname)
 READ(22,'(5x,a4)') specname
 param_reac(i,2) = nametype_gas(specname)
 READ(22,'(23x,e20.10)') preexp(i)
 READ(22,'(19x,e20.10)') actenergy(i)

ELSE IF (info(1:23).eq.'Dissociative adsorption') THEN

 type_reac(i) = 3
 READ(22,'(5x,a4)') specname
 param_reac(i,1) = nametype_gas(specname)
 READ(22,'(10x,a4)') specname
 param_reac(i,2) = nametype_ads(specname)

ELSE IF (info(1:22).eq.'Associative desorption') THEN

 type_reac(i) = 4
 READ(22,'(10x,a4)') specname
 param_reac(i,1) = nametype_ads(specname)
 READ(22,'(5x,a4)') specname
 param_reac(i,2) = nametype_gas(specname)
 READ(22,'(23x,e20.10)') preexp(i)
 READ(22,'(19x,e20.10)') actenergy(i)

ELSE IF (info(1:30).eq.'Bimolecular surface reaction 1') THEN

 type_reac(i) = 5
 READ(22,'(12x,a4)') specname
 param_reac(i,1) = nametype_ads(specname)
 READ(22,'(12x,a4)') specname
 param_reac(i,2) = nametype_ads(specname)
 READ(22,'(18x,i2)') param_reac(i,3)

 IF (param_reac(i,3).ge.3) THEN

  WRITE(*,*)
  WRITE(*,*)'*****************************************************************************'
  WRITE(*,*)'     ERROR! --> You need to increase the number of nb_param_max           '
  WRITE(*,*)'                Currently nb_param_max=5 at (variables.inc)           '
  WRITE(*,*)'*****************************************************************************'
  WRITE(*,'(''Program terminated.'')')
  WRITE(*,*)
  STOP

 ENDIF

 DO j=1,param_reac(i,3)
  READ(22,'(5x,a4)') specname
  param_reac(i,3+j) = nametype_gas(specname)
 ENDDO
```

```
  READ(22,'(23x,e20.10)') preexp(i)
  READ(22,'(19x,e20.10)') actenergy(i)

ELSE IF (info(1:30).eq.'Bimolecular surface reaction 2') THEN

  type_reac(i) = 10
  READ(22,'(21x,a4)') specname
  param_reac(i,1) = nametype_ads(specname)
  READ(22,'(21x,a4)') specname
  param_reac(i,2) = nametype_ads(specname)
  READ(22,'(18x,a4)') specname
  param_reac(i,3) = nametype_ads(specname)
  READ(22,'(23x,e20.10)') preexp(i)
  READ(22,'(19x,e20.10)') actenergy(i)

ELSE IF (info(1:30).eq.'Bimolecular surface reaction 3') THEN

  type_reac(i) = 11
  READ(22,'(21x,a4)') specname
  param_reac(i,1) = nametype_ads(specname)
  READ(22,'(21x,a4)') specname
  param_reac(i,2) = nametype_ads(specname)
  READ(22,'(20x,a4)') specname
  param_reac(i,3) = nametype_ads(specname)
  READ(22,'(20x,a4)') specname
  param_reac(i,4) = nametype_ads(specname)
  READ(22,'(23x,e20.10)') preexp(i)
  READ(22,'(19x,e20.10)') actenergy(i)

ELSE IF (info(1:26).eq.'Unimolecular decomposition') THEN

  type_reac(i) = 6
  READ(22,'(19x,a4)') specname
  param_reac(i,1) = nametype_ads(specname)
  READ(22,'(20x,a4)') specname
  param_reac(i,2) = nametype_ads(specname)
  READ(22,'(20x,a4)') specname
  param_reac(i,3) = nametype_ads(specname)
  READ(22,'(23x,e20.10)') preexp(i)
  READ(22,'(19x,e20.10)') actenergy(i)

ELSE IF (info(1:17).eq.'Surface diffusion') THEN

  type_reac(i) = 7
  READ(22,'(10x,a4)') specname
  param_reac(i,1) = nametype_ads(specname)
  READ(22,'(23x,e20.10)') preexp(i)
  READ(22,'(19x,e20.10)') actenergy(i)

ELSE IF (info(1:29).eq.'Unimolecular surface reaction') THEN

  type_reac(i) = 8
  READ(22,'(19x,a4)') specname
  param_reac(i,1) = nametype_ads(specname)
  READ(22,'(18x,a4)') specname
  param_reac(i,2) = nametype_ads(specname)
  READ(22,'(23x,e20.10)') preexp(i)
  READ(22,'(19x,e20.10)') actenergy(i)

ELSE IF (info(1:19).eq.'Reactive adsorption') THEN

  type_reac(i) = 9
  READ(22,'(18x,a4)') specname
  param_reac(i,1) = nametype_gas(specname)
  READ(22,'(19x,a4)') specname
```

```
       param_reac(i,2) = nametype_ads(specname)
       READ(22,'(18x,a4)') specname
       param_reac(i,3) = nametype_ads(specname)
       READ(22,'(23x,e20.10)') preexp(i)
       READ(22,'(19x,e20.10)') actenergy(i)

      ELSE

       WRITE(*,*)
       WRITE(*,'(''*** ERROR: unknown reaction name in parameters.dat file:'')')
       WRITE(*,'(a80)') info
       WRITE(*,'(''Program terminated.'')')
       WRITE(*,*)
       STOP

      ENDIF

    ENDDO

    CLOSE(22)

    RETURN
    END
*-----------------------------------------------------------------------------*
*******************************************************************************

*******************************************************************************
    SUBROUTINE read_control ()
*-----------------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: i,side
    CHARACTER :: info*80, str*6, domflux
    LOGICAL :: existe
*-----------------------------------------------------------------------------*
    INQUIRE(FILE='control.dat',EXIST=existe)

    IF (.not.existe) THEN

      WRITE(*,*)
      WRITE(*,'(''*** ERROR: Parameter file "control.dat" was not found'')')
      WRITE(*,'('' Program terminated.'')')
      WRITE(*,*)
      STOP

    ENDIF

    OPEN(23,file='control.dat',status='unknown')
    REWIND(23)

    incoming(:,:) = 0
    boundary(:) = 'PERIODIC'
    source(:) = .false.
    bound_fluxes = .false.

    READ(23,*)
    READ(23,'(a80)') info

    IF (info(26:28).ne.'3.0') THEN

      WRITE(*,*)
      WRITE(*,'(''*** ERROR: Version incompatibility with standard input'')')
      WRITE(*,'(''Please compile the appropriate Version or update the control.dat file'')')
      WRITE(*,'(''Program terminated.'')')
      WRITE(*,*)
```

```
 STOP

ENDIF

READ(23,*)
READ(23,*)
READ(23,*)
READ(23,*)
READ(23,*) idum
READ(23,*)
READ(23,*)
READ(23,*)
READ(23,*) nb_simul
READ(23,*)
READ(23,*)
READ(23,*)
READ(23,'(e20.10)') begin_time
READ(23,'(e20.10)') time_incr
READ(23,'(e20.10)') end_time

READ(23,*)
READ(23,*)
READ(23,*)
READ(23,'(e20.10)')conf_tstep
READ(23,*)
READ(23,*)
READ(23,*)
READ(23,'(e20.10)') time_boundary
READ(23,*)
READ(23,*)

DO side=top,right

 READ(23,*)
 READ(23,*)
 READ(23,*)
 READ(23,*) boundary(side)

 IF (boundary(side).eq.'NATURAL') THEN

  bound_fluxes = .true.
  READ(23,'(a6)') str

  IF (str.eq.'source') THEN
   source(side) = .true.
  ENDIF

  DO i=1,nb_ads

   READ(23,*) incoming0(side,i)

  ENDDO

 ENDIF

ENDDO

IF (boundary(top).eq.'PERIODIC') boundary(bottom)='PERIODIC'
IF (boundary(bottom).eq.'PERIODIC') boundary(top)='PERIODIC'
IF (boundary(left).eq.'PERIODIC') boundary(right)='PERIODIC'
IF (boundary(right).eq.'PERIODIC') boundary(left)='PERIODIC'

READ(23,*)
READ(23,*)
READ(23,*)
READ(23,*)
```

```
      READ(23,'(e20.10)') time_domain
      READ(23,*)
      READ(23,'(44x,a1)') domflux
      READ(23,*)

      domain_fluxes=.false.

      IF (domflux.eq.'y') THEN

        domain_fluxes=.true.

        DO i=1,nb_ads

          READ(23,*) dom_incoming0(i)

        ENDDO

      ENDIF

      CLOSE(23)

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

      RETURN
      END
*-----------------------------------------------------------------------------*
*******************************************************************************

*******************************************************************************
      SUBROUTINE read_lattice ()
*-----------------------------------------------------------------------------*
      IMPLICIT NONE
      INCLUDE 'variables.inc'

      INTEGER :: i,j
      INTEGER :: stype
      LOGICAL :: existe
*-----------------------------------------------------------------------------*
      NBADS(:) = 0

      INQUIRE(FILE='./Configuration_Files/config_00000.out',EXIST=existe)

      IF (.not.existe) THEN
        WRITE(*,*)
        WRITE(*,'(''*** ERROR: Initial configuration file (config_00000.out) not found'')')
        WRITE(*,'(''Program terminated.'')')
        WRITE(*,*)
        STOP
      ENDIF

      OPEN(22,file='./Configuration_Files/config_00000.out',status='unknown')
      REWIND(22)

      READ(22,*)
      READ(22,*)
      READ(22,'(i6)') lsize(x)
      READ(22,'(i6)') lsize(y)

      IF (lsize(x).gt.lmaxX) THEN
        WRITE(*,*)
        WRITE(*,'(''*** ERROR: input lattice is too large along X:'')')
        WRITE(*,'('' - lsize[X] = '',i6)') lsize(x)
        WRITE(*,'('' - lmax[X] = '',i4)') lmaxX
        WRITE(*,'(''Re-compile the program with an appropriate value for lsize(x)'')')
        WRITE(*,'(''        OR change the maximum lattice size lmaxX'')')
        STOP
```

```
    ENDIF

    IF (lsize(y).gt.lmaxY) THEN
      WRITE(*,*)
      WRITE(*,'(''*** ERROR: input lattice is too large along Y:'')')
      WRITE(*,'('' - lsize[Y] = '',i6)') lsize(y)
      WRITE(*,'('' - lmax[Y] = '',i4)') lmaxY
      WRITE(*,'(''Re-compile the program with an appropriate value for lsize(y)'')')
      WRITE(*,'(''         OR change the maximum lattice size lmaxY'')')
      STOP
    ENDIF

    nb_sites = lsize(x)*lsize(y)
    NBADS(0) = nb_sites
    lattice(:,:) = 0

    DO i=1,lsize(x)
      DO j=1,lsize(y)

        READ(22,'(2i4)') lattice(i,j)

      ENDDO
    ENDDO

    CLOSE(22)

    DO i=1,lsize(x)
      DO j=1,lsize(y)

        stype = lattice(i,j)

        IF (stype.gt.nb_ads) THEN
          WRITE(*,*)
          WRITE(*,'(''ERROR: too many adsorbed species'')')
          WRITE(*,'(''Initial lattice is not compatible with parameter file.'')')
          WRITE(*,'(''Program terminated.'')')
          WRITE(*,*)
          STOP
        ENDIF

        IF (stype.gt.0) THEN
          NBADS(stype) = NBADS(stype)+1
          NBADS(0) = NBADS(0)-1
        ENDIF

      ENDDO
    ENDDO

    RETURN
    END
*-----------------------------------------------------------------------*
*************************************************************************

*************************************************************************
    INTEGER FUNCTION nametype_ads(species)
*-----------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    CHARACTER :: species*4
    INTEGER :: i,found
*-----------------------------------------------------------------------*
    found = 0

    DO i=1,nb_ads
      IF (name_ads(i).eq.species) found = i
```

```
      ENDDO

      IF (found.eq.0) THEN
        WRITE(*,*)
        WRITE(*,'(''Unspecified adsorbed species found in "parameters.dat":'',a4)') species
        WRITE(*,'(''Program terminated.'')')
        WRITE(*,*)
        STOP
      ENDIF

      nametype_ads = found

      RETURN
      END
*-----------------------------------------------------------------------*
*************************************************************************

*************************************************************************
      INTEGER FUNCTION nametype_gas(species)
*-----------------------------------------------------------------------*
      IMPLICIT NONE
      INCLUDE 'variables.inc'

      CHARACTER :: species*4
      INTEGER :: i,found
*-----------------------------------------------------------------------*
      found = 0

      DO i=1,nb_gas
        IF (name_gas(i).eq.species) found = i
      ENDDO

      IF (found.eq.0) THEN
        WRITE(*,*)
        WRITE(*,'(''Unspecified gaseous species found in "parameters.dat":'',a4)') species
        WRITE(*,'(''Program terminated.'')')
        WRITE(*,*)
        STOP
      ENDIF

      nametype_gas = found

      RETURN
      END
*-----------------------------------------------------------------------*
*************************************************************************

*************************************************************************
      SUBROUTINE store_class_coords(ind,site1,site2,numb)
*-----------------------------------------------------------------------*
      IMPLICIT NONE
      INCLUDE 'variables.inc'

      INTEGER :: ind,site1,site2,numb
*-----------------------------------------------------------------------*
      class_coord(ind,1) = site1
      class_coord(ind,2) = site2
      class_coord(ind,3) = numb

      RETURN
      END
*-----------------------------------------------------------------------*
*************************************************************************
```

```
*******************************************************************************
    REAL(KIND=8) FUNCTION ran2(idum)
*-------------------------------------------------------------------------*
    INTEGER ::idum
    INTEGER,PARAMETER :: IM1=2147483563, IM2=2147483399, IMM1=IM1-1
    INTEGER,PARAMETER :: IA1=40014, IA2=40692, IQ1=53668, IQ2=52774
    INTEGER,PARAMETER :: IR1=12211, IR2=3791, NTAB=32, NDIV=1+IMM1/NTAB
    REAL(KIND=8),PARAMETER :: AM=1./IM1, EPS=1.2E-7, RNMX=1.-EPS
    INTEGER :: idum2,j,k,iv(NTAB),iy

    SAVE iv,iy,idum2
    DATA idum2/123456789/, iv/NTAB*0/, iy/0/
*-------------------------------------------------------------------------*
    IF (idum.le.0) THEN

     idum = max(-idum,1)
     idum2 = idum

     DO j=NTAB+8,1,-1

       k = idum/IQ1
       idum = IA1*(idum-k*IQ1)-k*IR1
       IF (idum.lt.0) idum = idum+IM1
       IF (j.le.NTAB) iv(j) = idum

     ENDDO

     iy = iv(1)

    ENDIF

    k = idum/IQ1
    idum = IA1*(idum-k*IQ1)-k*IR1
    IF (idum.lt.0) idum = idum+IM1

    k = idum2/IQ2
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    IF (idum2.lt.0) idum2 = idum2+IM2

    j = 1+iy/NDIV
    iy = iv(j)-idum2
    iv(j) = idum
    IF (iy.lt.1) iy = iy+IMM1
    ran2 = min(AM*iy,RNMX)

    RETURN
    END
*-------------------------------------------------------------------------*
*******************************************************************************

*******************************************************************************
    SUBROUTINE write_classes ()
*-------------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: i
    CHARACTER :: s1*4,s2*4
*-------------------------------------------------------------------------*
    WRITE(*,*)
    WRITE(*,'(''-------------------------------------------------'')')
    WRITE(*,'('' There are '',i2,'' different classes:'')')nb_class
    WRITE(*,'('' ---------------------------------  '')')
    WRITE(*,'('' Initially  '')')
    WRITE(*,'('' ------------------  '')')
    WRITE(*,'('' "ind"  "class"  "realnb_site_class""')')
```

```
      WRITE(*,'('' ------- --------- ----------'')')

      DO i=1,nb_class

       IF (class_coord(i,1).eq.0) THEN
         s1 = '[*] '
       ELSE IF (class_coord(i,1).eq.-1) THEN
         s1 = '   '
       ELSE
         s1 = name_ads(class_coord(i,1))
       END IF

       IF (class_coord(i,2).eq.0) THEN
         s2 = '[*] '
       ELSE IF (class_coord(i,2).eq.-1) THEN
         s2 = '   '
       ELSE
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
       END IF

       WRITE(*,'('' - class '',i2,'':  '',2(a4,'',''1x),i1,'' -->'',i8,'' sites'')') i,
     &           s1,s2,class_coord(i,3),realnb_site_class(i)

      ENDDO

      WRITE(*,'(''----------------------------------------------------'')')
      WRITE(*,*)

      RETURN
      END
*-----------------------------------------------------------------------*
*************************************************************************

*************************************************************************
      SUBROUTINE write_nbreaction ()
*-----------------------------------------------------------------------*
      IMPLICIT NONE
      INCLUDE 'variables.inc'

      INTEGER :: i
      INTEGER(KIND=8) :: nbrtot
*-----------------------------------------------------------------------*
      WRITE(*,'(''----------------------------------------------------'')')
      WRITE(*,*)
      WRITE(*,'(''----------------------------------------------------'')')
      WRITE(*,'('' Number of Reaction Events:'')')
      WRITE(*,'('' ---------------------------- '')')

      nbrtot = 0

      DO i=1,nb_reac

       WRITE(*,'(''- Reaction of type '',i2,'': '',i13)')i,INT(nb_reaction(i)/nb_simul)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

      ENDDO

      WRITE(*,'('' ---------------------------- '')')
      WRITE(*,'(''Total Number of Events: '',i12)')nbrtot
      WRITE(*,'(''----------------------------------------------------'')')

      RETURN
      END
*-----------------------------------------------------------------------*
*************************************************************************
```

```
********************************************************************************
     SUBROUTINE write_slabs ()
*-----------------------------------------------------------------------------*
     IMPLICIT NONE
     INCLUDE 'variables.inc'

     INTEGER :: i,slab
*-----------------------------------------------------------------------------*
     slab = ceiling((time-begin_time)/time_incr)

     DO i=0,nb_ads

       theta(i,slab) = REAL(NBADS(i),8)

     ENDDO

     DO i=1,nb_gas
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
       NBGAS(i) = 0
     ENDDO

     RETURN
     END
*-----------------------------------------------------------------------------*
********************************************************************************

********************************************************************************
     SUBROUTINE write_conf_files ()
*-----------------------------------------------------------------------------*
     IMPLICIT NONE
     INCLUDE 'variables.inc'

     INTEGER :: k, n, i, j
     CHARACTER :: conf_file*38, ch
*-----------------------------------------------------------------------------*
     conf_file(1:29)='./Configuration_Files/config_'

     DO k=4,0,-1
       n = mod(INT(REAL(ttt)/10**k),10)
       ch = char(48+n)
       conf_file(34-k:34-k) = ch
     ENDDO

     conf_file(35:38) = '.out'

     OPEN(22,FILE=conf_file,STATUS='unknown')
     REWIND(22)

       WRITE(22,'(''                    Time:'',e10.3,  ''  [sec]'')')ttt*conf_tstep
       WRITE(22,'(''          ------------------------------------------'')')

       WRITE(22,'(i6)') lsize(x)
       WRITE(22,'(i6)') lsize(y)

       DO i=1,lsize(x)
         DO j=1,lsize(y)
            WRITE(22,'(2i4)') lattice(i,j)
         ENDDO
       ENDDO

     CLOSE(22)

     RETURN
     END
*-----------------------------------------------------------------------------*
********************************************************************************
```

```
*******************************************************************************
    SUBROUTINE write_ttt ()
*------------------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'
*------------------------------------------------------------------------------*
    OPEN(25,file='./Configuration_Files/ttt_updates.out',status='unknown')
    REWIND(25)

     WRITE(25,'(i6)') ttt

    CLOSE(25)

    RETURN
    END
*------------------------------------------------------------------------------*
*******************************************************************************

*******************************************************************************
    SUBROUTINE write_configuration ()
*------------------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: i, j
*------------------------------------------------------------------------------*
    OPEN(22,FILE='./Configuration_Files/confin.dat',STATUS='unknown')
    REWIND(22)

     WRITE(22,'(i6)') lsize(x)
     WRITE(22,'(i6)') lsize(y)

     DO i=1,lsize(x)
      DO j=1,lsize(y)
         WRITE(22,'(2i4)') lattice(i,j)
      ENDDO
     ENDDO

    CLOSE(22)

    RETURN
    END
*------------------------------------------------------------------------------*
*******************************************************************************

*******************************************************************************
    SUBROUTINE write_coverage ()
*------------------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: i,sl
    REAL(KIND=8) :: tt
*------------------------------------------------------------------------------*
    DO i=0,nb_ads

     theta(i,0) = theta0(i)

     DO sl=1,nb_pts

      theta(i,sl) = theta(i,sl)/REAL(nb_sites,8)

     ENDDO

    ENDDO
```

```
      OPEN(UNIT=22,FILE='COVERAGES.OUT',STATUS='unknown')
      REWIND(22)

      DO sl=0,nb_pts

       tt = begin_time + REAL(sl,8)*time_incr

       IF (nb_ads.eq.1) THEN
         WRITE(22,'(f13.7,2e15.6)') tt,(theta(i,sl),i=0,nb_ads)
       ELSEIF (nb_ads.eq.2) THEN
         WRITE(22,'(f13.7,3e15.6)') tt,(theta(i,sl),i=0,nb_ads)
       ELSEIF (nb_ads.eq.3) THEN
         WRITE(22,'(f13.7,4e15.6)') tt,(theta(i,sl),i=0,nb_ads)
       ELSEIF (nb_ads.eq.4) THEN
         WRITE(22,'(f13.7,5e15.6)') tt,(theta(i,sl),i=0,nb_ads)
       ELSE
         WRITE(22,'(f13.7,6e15.6)') tt,(theta(i,sl),i=0,nb_ads)
       ENDIF

      ENDDO

      CLOSE(22)

      RETURN
      END
*-----------------------------------------------------------------------*
*************************************************************************

*************************************************************************
      SUBROUTINE write_rates ()
*-----------------------------------------------------------------------*
      IMPLICIT NONE
      INCLUDE 'variables.inc'

      INTEGER :: i,sl
      REAL(KIND=8) :: tt
*-----------------------------------------------------------------------*
      DO i=1,nb_gas

       rate(i,0) = rate0(i)

       DO sl=1,nb_pts

         rate(i,sl) = rate(i,sl)/(N_AVOG*time_incr)/nb_simul

       ENDDO

      ENDDO

      OPEN(UNIT=23,FILE='RATES.OUT',STATUS='unknown')
      REWIND(23)

      DO sl=0,nb_pts

       tt = begin_time + REAL(sl,8)*time_incr

       IF (nb_gas.eq.1) THEN
         WRITE(23,'(f13.7,2e15.6)') tt,(rate(i,sl),i=1,nb_gas)
       ELSEIF (nb_gas.eq.2) THEN
         WRITE(23,'(f13.7,3e15.6)') tt,(rate(i,sl),i=1,nb_gas)
       ELSEIF (nb_gas.eq.3) THEN
         WRITE(23,'(f13.7,4e15.6)') tt,(rate(i,sl),i=1,nb_gas)
       ELSEIF (nb_gas.eq.4) THEN
         WRITE(23,'(f13.7,5e15.6)') tt,(rate(i,sl),i=1,nb_gas)
       ELSE
         WRITE(23,'(f13.7,6e15.6)') tt,(rate(i,sl),i=1,nb_gas)
```

```
      ENDIF

    ENDDO

    CLOSE(23)

    DO i=1,nb_gas

     rate_more(i,0) = rate(i,0)*site_density/nb_sites

     DO sl=1,nb_pts

        rate_more(i,sl) = rate(i,sl)*site_density/nb_sites

     ENDDO

    ENDDO

    OPEN(UNIT=25,FILE='RATES_mol.s^-1.m^-2.OUT',STATUS='unknown')
    REWIND(25)

    DO sl=0,nb_pts

     tt = begin_time + REAL(sl,8)*time_incr

     IF (nb_gas.eq.1) THEN
       WRITE(25,'(f13.7,2e15.6)') tt,(rate_more(i,sl),i=1,nb_gas)
     ELSEIF (nb_gas.eq.2) THEN
       WRITE(25,'(f13.7,3e15.6)') tt,(rate_more(i,sl),i=1,nb_gas)
     ELSEIF (nb_gas.eq.3) THEN
       WRITE(25,'(f13.7,4e15.6)') tt,(rate_more(i,sl),i=1,nb_gas)
     ELSEIF (nb_gas.eq.4) THEN
       WRITE(25,'(f13.7,5e15.6)') tt,(rate_more(i,sl),i=1,nb_gas)
     ELSE
       WRITE(25,'(f13.7,6e15.6)') tt,(rate_more(i,sl),i=1,nb_gas)
     ENDIF

    ENDDO

    CLOSE(25)

    RETURN
    END
*-------------------------------------------------------------------------------*
*********************************************************************************

*********************************************************************************
*----------------------          Parameters          ----------------------*

    INTEGER,PARAMETER :: x=1 , y=2
    INTEGER,PARAMETER :: top=1, bottom=2, left=3, right=4
    INTEGER,PARAMETER :: nb_ads_max=5, nb_gas_max=5
    INTEGER,PARAMETER :: lmaxX=2000, lmaxY=500
    INTEGER,PARAMETER :: nb_site_max=lmaxX*lmaxY
    INTEGER,PARAMETER :: nb_reac_max=11, nb_param_max=5
    INTEGER,PARAMETER :: nb_class_max=40, nbclsite_max=100
    INTEGER,PARAMETER :: nb_pts_max=100000
    REAL(KIND=8) :: pi=3.14159265359, R=8.314472, N_Avog=6.0221367e23

*----------------------        GENERAL variables        ----------------------*

    INTEGER :: nb_pts, idum, nb_simul, ttt, t_b, t_d
    INTEGER :: time_hours,time_minutes,time_seconds
    REAL(KIND=8) :: time, begin_time, end_time, conf_tstep, time_incr
    REAL(KIND=8) :: cpu_begin,cpu_end
    REAL(KIND=8) :: temperature, site_density
```

```
*---------------------      SYSTEM variables      ----------------------*

    INTEGER :: insert_type
    INTEGER,DIMENSION(nb_ads_max) :: priority
    CHARACTER,DIMENSION(0:nb_ads_max) :: name*4
    LOGICAL :: finish

*---------------------      SPECIES variables      ----------------------*

    INTEGER :: nb_gas, nb_ads
    CHARACTER,DIMENSION(nb_gas_max) :: name_gas*4
    CHARACTER,DIMENSION(nb_ads_max) :: name_ads*4
    REAL(KIND=8),DIMENSION(nb_gas_max) :: weight_gas,sticking_gas,press

*---------------------      LATTICE variables      ----------------------*

    INTEGER :: nb_sites
    INTEGER,DIMENSION(2) :: lsize
    INTEGER :: nb_class,nb_proba
    INTEGER,DIMENSION(nb_class_max,3) :: class_coord
    INTEGER(KIND=1),DIMENSION(0:nb_ads_max,-1:nb_ads_max,-1:4) :: class
    INTEGER,DIMENSION(nb_class_max,2) :: proba_coord
    INTEGER,DIMENSION(lmaxX,lmaxY) :: lattice
    INTEGER(KIND=8),DIMENSION(0:nb_ads_max) :: NB_fin, NB
    INTEGER,DIMENSION(nb_class_max) :: nb_site_class
    INTEGER,DIMENSION(nb_class_max) :: realnb_site_class
    INTEGER,DIMENSION(nb_class_max,nb_site_max,2) :: class_site
    INTEGER(KIND=1),DIMENSION(lmaxX,lmaxY) :: site_clnb
    INTEGER(KIND=1),DIMENSION(lmaxX,lmaxY,nbclsite_max) :: site_clid
    INTEGER,DIMENSION(lmaxX,lmaxY,nbclsite_max) :: site_clsite
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    INTEGER,DIMENSION(nb_ads_max) :: dom_nb_left, dom_nb_to_add
    REAL(KIND=8) :: time_domain, dom_int_boundary, dom_time_next
    REAL(KIND=8) :: dom_time_fluxup_slots, dom_time_fluxupdate
    LOGICAL(KIND=1),DIMENSION(nb_class_max,nb_site_max) :: enabled
    LOGICAL(KIND=1):: clean, gc, domain_fluxes

*---------------------      REACTIONS variables      ----------------------*

    INTEGER :: nb_reac
    INTEGER,DIMENSION(nb_reac_max) :: type_reac
    INTEGER,DIMENSION(nb_reac_max,nb_param_max) :: param_reac
    INTEGER(KIND=8),DIMENSION(nb_reac_max) :: nb_reaction

*---------------------      KINETICS variables      ----------------------*

    REAL(KIND=8),DIMENSION(nb_reac_max) :: preexp,actenergy
    REAL(KIND=8),DIMENSION(nb_reac_max) :: k_reac
    REAL(KIND=8),DIMENSION(0:nb_ads_max,-1:nb_ads_max) :: proba
    REAL(KIND=8),DIMENSION(nb_reac_max) :: gamma

*---------------------      DATAS variables      ----------------------*

    REAL(KIND=8),DIMENSION(0:nb_ads_max) :: theta0
    REAL(KIND=8),DIMENSION(nb_gas_max) :: rate0
    INTEGER,DIMENSION(0:nb_ads_max) :: NBADS
    INTEGER(KIND=8),DIMENSION(nb_gas_max) :: NBGAS
    REAL(KIND=8),DIMENSION(0:nb_ads_max,0:nb_pts_max) :: theta
    REAL(KIND=8),DIMENSION(nb_gas_max,0:nb_pts_max) :: rate
    REAL(KIND=8),DIMENSION(nb_gas_max,0:nb_pts_max) :: rate_more

*---------------------      BOUND variables      ----------------------*

    LOGICAL :: bound_fluxes
    CHARACTER,DIMENSION(4) :: boundary*8
```

```
      LOGICAL,DIMENSION(4) :: source
      REAL(KIND=8) :: time_boundary, int_boundary, btime_next
      REAL(KIND=8) :: time_fluxup_slots, time_fluxupdate
      INTEGER,DIMENSION(4,nb_ads_max) :: nb_left, nb_to_add
      INTEGER,DIMENSION(4,nb_ads_max) :: incoming,incoming0

*--------------------          COMMON          ----------------------*

      COMMON / GENERAL / time,begin_time,time_incr,end_time,conf_tstep,cpu_begin,
     &          cpu_end,time_hours,time_minutes,time_seconds,idum,nb_simul,
     &          nb_pts,temperature,site_density,ttt,t_b,t_d

      COMMON / SYSTEM / insert_type, priority,name,finish

      COMMON / SPECIES / nb_gas,nb_ads,name_gas,name_ads,weight_gas,sticking_gas,
     &          press

      COMMON / LATTICE / nb_sites,lsize,nb_class,nb_proba,class_coord,class,
     &          proba_coord,lattice,NB_fin,NB,nb_site_class,
     &          realnb_site_class,class_site,site_clnb,site_clid,
     &          site_clsite,dom_incoming0,dom_incoming,dom_nb_left,
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
     &          dom_time_next,dom_time_fluxup_slots,dom_time_fluxupdate,
     &          enabled,clean,gc,domain_fluxes

      COMMON / REACTIONS / nb_reaction,nb_reac,type_reac,param_reac

      COMMON / KINETICS / preexp,actenergy,k_reac,proba,gamma

      COMMON / DATAS / theta0,rate0,NBADS,NBGAS,theta,rate,rate_more

      COMMON / BOUND / time_boundary,int_boundary,btime_next,time_fluxupdate,
     &          time_fluxup_slots,nb_left,nb_to_add,incoming,incoming0,
     &          bound_fluxes,boundary,source

*-------------------------------------------------------------------------*
***************************************************************************
```

A.3 The gap-tooth framework files (Chapter 5)

A.3.1 'No'gap-tooth fortran files

A.3.1.1 Main program

```
********************************************************************************
    PROGRAM  NoGap_Tooth
*-------------------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: nx, ny, k, i, side
    INTEGER :: comptt, mu, t_step
    REAL(KIND=8) :: gamma_tot, sum, tau
    REAL(KIND=8) :: ran2, ran
*-------------------------------------------------------------------------------*

    WRITE(*,*)
    WRITE(*,*)
    WRITE(*,'(''************************************************************'')')
    WRITE(*,*)
    WRITE(*,'(''**********                SIMULATION STARTED              **********'')')
    WRITE(*,*)
    WRITE(*,'(''************************************************************'')')
    WRITE(*,'(''>                          NoGap_Tooth.exe            <'')')
    WRITE(*,'(''>                       By Ioannis S. Fragkopoulos     <'')')
    WRITE(*,'(''>      Gap-Tooth kMC Simulation with internal interactions   <'')')
    WRITE(*,'(''>                    and external boundary fluxes         <'')')
    WRITE(*,'(''>        Reese et al., J. Comput. Phys., 2001, 173, 302-321   <'')')
    WRITE(*,'(''>         Gear et al., Physics Letters A, 2003, 316, 190-195     <'')')
    WRITE(*,'(''************************************************************'')')
    WRITE(*,*)

    CALL CPU_TIME(cpu_begin)

    CALL read_parameters ()

    CALL read_patches ()

    CALL read_fluxes ()

    CALL calc_kreac ()

    CALL init_classes ()

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

    CALL ext_fluxes_distr ()

    CALL init_ext_fluxes ()

    theta(:,:,:,:) = 0.0
    rate(:,:,:,:) = 0.0
    NBGAS(:) = 0
    outgoing(:,:,:,:,:)=0
    ingoing(:,:,:,:,:)=0

    DO t_step=0,time_steps-1
```

```
    DO nsim = 1,nb_simul

     DO ny = 1,teeth(y)

      DO nx = 1,teeth(x)

       WRITE(*,'(''-----      Number of occupied sites      -----'')')
       WRITE(*,'(''        ----------------------------        '')')
       WRITE(*,'(''                Tooth [x]:'',i4)')nx
       WRITE(*,'(''                Tooth [y]:'',i4)')ny
       WRITE(*,'(''-----------------------------------------------'')')
       WRITE(*,'(''           Simulation:'',i5,''/'',i3)')nsim,nb_simul
       WRITE(*,'(''    Time(s)       Number of occupied sites'')')
       WRITE(*,'(''   ---------      -------------------------  '')')

       time = begin_time + t_step*time_int_fluxes
       time_end = (t_step+1)*time_int_fluxes

       CALL read_lattice (nx,ny)

       CALL init_class_sites (nx,ny)

         comptt = 0
         int_fluxes=.true.
         int_side_fluxes(:)=.false.

       DO WHILE (time.lt.time_end)

        CALL internal_fluxes(nx,ny)

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

        CALL calc_proba ()

        CALL calc_gamma ()

        gamma_tot = 0.0

        DO i=1,nb_reac
         gamma_tot = gamma_tot+gamma(i)
        ENDDO

        IF (gamma_tot.gt.1e-6) THEN

         tau = 1.0/(gamma_tot*REAL(nb_sites,8))
  &            *LOG(1/ran2(idum))

         mu = 1
         sum = 0.0
         ran = ran2(idum)*REAL(gamma_tot,8)

         DO i=1,nb_reac-1

          sum = sum+gamma(i)

          IF ((ran.gt.sum).and.(ran.lt.sum+gamma(i+1))) THEN

           mu = i+1

          ENDIF

         ENDDO

         nb_reaction(nx,ny,mu) = nb_reaction(nx,ny,mu)+1

         IF ((type_reac(mu).eq.1).or.(type_reac(mu).eq.2).or.
```

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

```
          CALL make_reaction_1site(nx,ny,mu)

         ELSE

          CALL make_reaction_2site(nx,ny,mu)

         ENDIF

         time = time + tau

         IF (clean) THEN
          CALL init_class_sites (nx,ny)
         ENDIF

        ELSE

         time = time_end

         WRITE(*,*)
         WRITE(*,'(''-----------------------------------------------'')')
         WRITE(*,'(''                 WARNING!                '')')
         WRITE(*,'(''     No reaction is possible to take place     '')')
         WRITE(*,'(''   Total Transition Probability is very low    '')')
         WRITE(*,'(''-----------------------------------------------'')')
         WRITE(*,*)

        ENDIF

         CALL write_slabs (nx,ny)

         comptt = comptt+1
         IF (mod(comptt,10).eq.0) THEN
          WRITE(*,'(e14.6,7i8)')time,(NBADS(k),k=0,nb_ads)
         ENDIF

        ENDDO

        WRITE(*,'(''-----------------------------------------------'')')

        IF (nsim.eq.nb_simul) THEN
         CALL write_configuration (nx,ny)
        ENDIF

       ENDDO

      ENDDO

     ENDDO

     CALL calc_int_fluxes ()

     outgoing(:,:,:,:,:)=0

    ENDDO

    CALL write_nbreaction ()

    CALL write_indiv_coverages ()

    CALL write_indiv_rates ()

    CALL CPU_TIME(cpu_end)

    time_seconds = INT(cpu_end-cpu_begin)
```

```
      time_minutes = time_seconds/60
      time_hours = time_minutes/60
      time_minutes = time_minutes-time_hours*60
      time_seconds = time_seconds-time_hours*3600-time_minutes*60

      WRITE(*,*)

      IF ((time_minutes.lt.10).and.(time_seconds.lt.10)) THEN

        WRITE(*,'(''Total simulation time: '',i3,'':0'',i1,'':0'',i1)')
     &        time_hours,time_minutes,time_seconds

      ELSE IF ((time_minutes.lt.10).and.(time_seconds.ge.10)) THEN

        WRITE(*,'(''Total simulation time: '',i3,'':0'',i1,'':'',i2)')
     &        time_hours,time_minutes,time_seconds

      ELSE IF ((time_minutes.ge.10).and.(time_seconds.lt.10)) THEN

        WRITE(*,'(''Total simulation time: '',i3,'':'',i2,'':0'',i1)')
     &        time_hours,time_minutes,time_seconds

      ELSE

        WRITE(*,'(''Total simulation time: '',i3,'':'',i2,'':'',i2)')
     &        time_hours,time_minutes,time_seconds

      ENDIF

      WRITE(*,*)
      WRITE(*,'(''************************************************************'')')
      WRITE(*,'(''>                          NoGap_Tooth.exe             <'')')
      WRITE(*,'(''>                       By Ioannis S. Fragkopoulos     <'')')
      WRITE(*,'(''>     Gap-Tooth kMC Simulation with internal interactions     <'')')
      WRITE(*,'(''>                    and external boundary fluxes          <'')')
      WRITE(*,'(''>        Reese et al., J. Comput. Phys., 2001, 173, 302-321     <'')')
      WRITE(*,'(''>         Gear et al., Physics Letters A, 2003, 316, 190-195     <'')')
      WRITE(*,'(''************************************************************'')')
      WRITE(*,*)
      WRITE(*,'(''**********            SIMULATION COMPLETED              **********'')')
      WRITE(*,*)
      WRITE(*,'(''************************************************************'')')
      WRITE(*,*)

      END PROGRAM
*-----------------------------------------------------------------------*
*************************************************************************
```

A.3.1.2 Subroutines

```
*******************************************************************************
    LOGICAL FUNCTION outside (xsite,ysite)
*-------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: xsite,ysite
*-------------------------------------------------------------------*
    outside = .false.

    IF ((xsite.le.0).or.(xsite.ge.lsize(x)+1)) outside = .true.
    IF ((ysite.le.0).or.(ysite.ge.lsize(y)+1)) outside = .true.

    RETURN
    END
*-------------------------------------------------------------------*
*******************************************************************************

*******************************************************************************
    SUBROUTINE neighbour (xsite,ysite,xneigh,yneigh,neigh)
*-------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: xsite,ysite,xneigh,yneigh,neigh
*-------------------------------------------------------------------*
    xneigh = xsite
    yneigh = ysite

    IF (neigh.eq.top) yneigh = yneigh+1
    IF (neigh.eq.bottom) yneigh = yneigh-1
    IF (neigh.eq.left) xneigh = xneigh-1
    IF (neigh.eq.right) xneigh = xneigh+1

    RETURN
    END
*-------------------------------------------------------------------*
*******************************************************************************

*******************************************************************************
    SUBROUTINE internal_fluxes(nx,ny)
*-------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: nx, ny, k, side
*-------------------------------------------------------------------*
    IF (int_fluxes) THEN

      tot_exch_side_part(:)=0
      tot_exch_part=0

      DO side=1,4

        DO k=1,nb_ads

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
          tot_exch_part = tot_exch_part + ingoing(nsim,nx,ny,side,k)

        ENDDO

        IF (tot_exch_side_part(side).gt.0) THEN
```

```
       int_side_fluxes(side)=.true.

      ENDIF

     ENDDO

   IF (tot_exch_part.gt.0) THEN
    CALL make_int_fluxes (nx,ny,nsim)
   ELSE
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   ENDIF

  ENDIF

  RETURN
  END
*------------------------------------------------------------------------------*
********************************************************************************

********************************************************************************
  SUBROUTINE calc_int_fluxes ()
*------------------------------------------------------------------------------*
*  Used in the case of no gap between the consecutive teeth
*------------------------------------------------------------------------------*
  IMPLICIT NONE
  INCLUDE 'variables.inc'

  INTEGER :: i, k, nx, ny
*------------------------------------------------------------------------------*
  DO ny = 1,teeth(y)

   DO nx = 1,teeth(x)

    DO k = 1,nb_ads

          IF (ny.lt.teeth(y)) THEN

           ingoing(:,nx,ny+1,2,k) = ingoing(nb_simul,nx,ny+1,2,k)
  &                          + outgoing(nb_simul,nx,ny,1,k)
           ingoing(:,nx,ny,1,k) = ingoing(nb_simul,nx,ny,1,k)
  &                          + outgoing(nb_simul,nx,ny+1,2,k)

          ENDIF

          IF (nx.lt.teeth(x)) THEN

           ingoing(:,nx+1,ny,3,k) = ingoing(nb_simul,nx+1,ny,3,k)
  &                          + outgoing(nb_simul,nx,ny,4,k)
           ingoing(:,nx,ny,4,k) = ingoing(nb_simul,nx,ny,4,k)
  &                          + outgoing(nb_simul,nx+1,ny,3,k)

          ENDIF

    ENDDO

   ENDDO

  ENDDO

  RETURN
  END
*------------------------------------------------------------------------------*
********************************************************************************
```

```
*******************************************************************************
      SUBROUTINE make_int_fluxes (nx,ny)
*-------------------------------------------------------------------------*
*    Boundary distribution of ingoing species
*-------------------------------------------------------------------------*
      IMPLICIT NONE
      INCLUDE 'variables.inc'

      INTEGER :: i,j,k,side,rsitex,rsitey,xn,yn,nx,ny
      INTEGER,DIMENSION(0:nb_ads_max) :: int_nbB
      REAL(KIND=8) :: ran2
      LOGICAL :: ok
*-------------------------------------------------------------------------*
      DO side=top,right

       int_nbB(:) = 0

       IF ((side.eq.top).and.(int_side_fluxes(side))) THEN

        j=lsize(y)

        DO i=1,lsize(x)
          int_nbB(lattice(i,j)) = int_nbB(lattice(i,j))+1
        ENDDO

        DO i=1,nb_ads

         IF (ingoing(nsim,nx,ny,side,i).ne.0) THEN

          IF (int_nbB(0).gt.0) THEN

           ok = .false.

           DO WHILE (.not.ok)

            rsitex = INT(ran2(idum)*REAL(lsize(x),8))+1
            rsitey = lsize(y)

            IF (lattice(rsitex,rsitey).eq.0) THEN

             lattice(rsitex,rsitey) = i
             NBADS(i) = NBADS(i)+1
             NBADS(0) = NBADS(0)-1
             int_nbB(0) = int_nbB(0)-1
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

             IF ((int_nbB(0).eq.0).or.
   &                 (ingoing(nsim,nx,ny,side,i).eq.0)) ok = .true.

             CALL update_class_onesite(rsitex,rsitey)
             CALL update_class_twosite(nx,ny,rsitex,rsitey)

             DO k=top,right
              CALL neighbour(rsitex,rsitey,xn,yn,k)
              CALL update_class_twosite(nx,ny,xn,yn)
             ENDDO

            ENDIF

           ENDDO

          ENDIF

         ENDIF

        ENDDO
```

```
      ELSE IF ((side.eq.bottom).and.(int_side_fluxes(side))) THEN

       j=1

       DO i=1,lsize(x)
         int_nbB(lattice(i,j)) = int_nbB(lattice(i,j))+1
       ENDDO

       DO i=1,nb_ads

        IF (ingoing(nsim,nx,ny,side,i).ne.0) THEN

          IF (int_nbB(0).gt.0) THEN

            ok = .false.

            DO WHILE (.not.ok)

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              rsitey = 1

              IF (lattice(rsitex,rsitey).eq.0) THEN

                lattice(rsitex,rsitey) = i
                NBADS(i) = NBADS(i)+1
                NBADS(0) = NBADS(0)-1
                int_nbB(0) = int_nbB(0)-1
                ingoing(nsim,nx,ny,side,i) = ingoing(nsim,nx,ny,side,i)-1


                IF ((int_nbB(0).eq.0).or.
     &                    (ingoing(nsim,nx,ny,side,i).eq.0)) ok = .true.

                CALL update_class_onesite(rsitex,rsitey)
                CALL update_class_twosite(nx,ny,rsitex,rsitey)

                DO k=top,right
                  CALL neighbour(rsitex,rsitey,xn,yn,k)
                  CALL update_class_twosite(nx,ny,xn,yn)
                ENDDO

              ENDIF

            ENDDO

          ENDIF

        ENDIF

       ENDDO

      ELSE IF ((side.eq.left).and.(int_side_fluxes(side))) THEN

       i=1

       DO j=1,lsize(y)
         int_nbB(lattice(i,j)) = int_nbB(lattice(i,j))+1
       ENDDO

       DO i=1,nb_ads

        IF (ingoing(nsim,nx,ny,side,i).ne.0) THEN

          IF (int_nbB(0).gt.0) THEN
```

```
          ok = .false.

          DO WHILE (.not.ok)

           rsitex = 1
           rsitey = INT(ran2(idum)*REAL(lsize(y),8))+1

           IF (lattice(rsitex,rsitey).eq.0) THEN

            lattice(rsitex,rsitey) = i
            NBADS(i) = NBADS(i)+1
            NBADS(0) = NBADS(0)-1
            int_nbB(0) = int_nbB(0)-1
             ingoing(nsim,nx,ny,side,i) = ingoing(nsim,nx,ny,side,i)-1

            IF ((int_nbB(0).eq.0).or.
     &                (ingoing(nsim,nx,ny,side,i).eq.0)) ok = .true.

            CALL update_class_onesite(rsitex,rsitey)
            CALL update_class_twosite(nx,ny,rsitex,rsitey)

            DO k=top,right
             CALL neighbour(rsitex,rsitey,xn,yn,k)
             CALL update_class_twosite(nx,ny,xn,yn)
            ENDDO

           ENDIF

          ENDDO

         ENDIF

        ENDIF

       ENDDO

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

       i=lsize(x)

       DO j=1,lsize(y)
          int_nbB(lattice(i,j)) = int_nbB(lattice(i,j))+1
       ENDDO

       DO i=1,nb_ads

        IF (ingoing(nsim,nx,ny,side,i).ne.0) THEN

         IF (int_nbB(0).gt.0) THEN

          ok = .false.

          DO WHILE (.not.ok)

           rsitex = lsize(x)
           rsitey = INT(ran2(idum)*REAL(lsize(y),8))+1

           IF (lattice(rsitex,rsitey).eq.0) THEN

            lattice(rsitex,rsitey) = i
            NBADS(i) = NBADS(i)+1
            NBADS(0) = NBADS(0)-1
            int_nbB(0) = int_nbB(0)-1
            ingoing(nsim,nx,ny,side,i) = ingoing(nsim,nx,ny,side,i)-1

            IF ((int_nbB(0).eq.0).or.
```

```
  &                     (ingoing(nsim,nx,ny,side,i).eq.0)) ok = .true.

            CALL update_class_onesite(rsitex,rsitey)
            CALL update_class_twosite(nx,ny,rsitex,rsitey)

            DO k=top,right
             CALL neighbour(rsitex,rsitey,xn,yn,k)
             CALL update_class_twosite(nx,ny,xn,yn)
            ENDDO

           ENDIF

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

         ENDIF

        ENDIF

       ENDDO

      ENDIF

     ENDDO

     RETURN
     END
*------------------------------------------------------------------------------*
*********************************************************************************

*********************************************************************************
     SUBROUTINE external_fluxes(nx,ny)
*------------------------------------------------------------------------------*
     IMPLICIT NONE
     INCLUDE 'variables.inc'

     INTEGER :: nx, ny
*------------------------------------------------------------------------------*
       IF ((bound_fluxes).and.(time.gt.time_fluxupdate(nx,ny,nsim))) THEN

        CALL update_ext_fluxes (nx,ny)
        CALL make_ext_fluxes (nx,ny)

       ENDIF

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

         CALL make_ext_fluxes (nx,ny)

       ENDIF

     RETURN
     END
*------------------------------------------------------------------------------*
*********************************************************************************

*********************************************************************************
     SUBROUTINE make_ext_fluxes (nx,ny)
*------------------------------------------------------------------------------*
     IMPLICIT NONE
     INCLUDE 'variables.inc'

     INTEGER :: i, k, side, rsite, xn, yn, nx, ny
     INTEGER,DIMENSION(0:nb_ads_max) :: nbB
     REAL(KIND=8) :: ran2
     LOGICAL :: ok
*------------------------------------------------------------------------------*
```

```
    DO side=top,right

     nbB(:) = 0

     IF ((side.eq.top).and.(source(side))) THEN

      DO i=1,lsize(x)
       nbB(lattice(i,lsize(y))) = nbB(lattice(i,lsize(y)))+1
      ENDDO

      DO i=1,nb_ads

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

        IF (nb_left(nx,ny,side,i,nsim).gt.0) THEN

         IF (nbB(0).gt.0) THEN

          ok = .false.

          DO WHILE (.not.ok)

           rsite = INT(ran2(idum)*REAL(lsize(x),8))+1

           IF (lattice(rsite,lsize(y)).eq.0) THEN

            lattice(rsite,lsize(y)) = i
            NBADS(i) = NBADS(i)+1
            NBADS(0) = NBADS(0)-1
            nbB(0) = nbB(0)-1

            nb_left(nx,ny,side,i,nsim) = nb_left(nx,ny,side,i,nsim)-1
            ext_flux_ind(nx,ny,side,i,nsim) = ext_flux_ind(nx,ny,side,i,nsim)-1

            IF ((nbB(0).eq.0).or.(nb_left(nx,ny,side,i,nsim).eq.0)
&                                        .or.(ext_flux_ind(nx,ny,side,i,nsim).eq.0)) ok = .true.

            CALL update_class_onesite(rsite,lsize(y))
            CALL update_class_twosite(nx,ny,rsite,lsize(y))

            DO k=top,right
             CALL neighbour(rsite,lsize(y),xn,yn,k)
             CALL update_class_twosite(nx,ny,xn,yn)
            ENDDO

           ENDIF

          ENDDO

         ENDIF

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

         IF (nbB(i).gt.0) THEN

          ok = .false.

          DO WHILE (.not.ok)

           rsite = INT(ran2(idum)*REAL(lsize(x),8))+1

           IF (lattice(rsite,lsize(y)).eq.i) THEN

            lattice(rsite,lsize(y)) = 0
            NBADS(i) = NBADS(i)-1
            NBADS(0) = NBADS(0)+1
```

```
              nbB(i) = nbB(i)-1

              nb_left(nx,ny,side,i,nsim) = nb_left(nx,ny,side,i,nsim)+1
              ext_flux_ind(nx,ny,side,i,nsim) = ext_flux_ind(nx,ny,side,i,nsim)+1

              IF ((nbB(i).eq.0).or.(nb_left(nx,ny,side,i,nsim).eq.0)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

              CALL update_class_onesite(rsite,lsize(y))
              CALL update_class_twosite(nx,ny,rsite,lsize(y))

              DO k=top,right
                CALL neighbour(rsite,lsize(y),xn,yn,k)
                CALL update_class_twosite(nx,ny,xn,yn)
              ENDDO

             ENDIF

            ENDDO

           ENDIF

          ENDIF

         ENDIF

        ENDDO

       ELSE IF ((side.eq.bottom).and.(source(side))) THEN

        DO i=1,lsize(x)
          nbB(lattice(i,1)) = nbB(lattice(i,1))+1
        ENDDO

        DO i=1,nb_ads

         IF ((nb_left(nx,ny,side,i,nsim).ne.0).and.(ext_flux_ind(nx,ny,side,i,nsim).ne.0)) THEN

          IF (nb_left(nx,ny,side,i,nsim).gt.0) THEN

           IF (nbB(0).gt.0) THEN

            ok = .false.

            DO WHILE (.not.ok)

             rsite = INT(ran2(idum)*REAL(lsize(x),8))+1

             IF (lattice(rsite,1).eq.0) THEN

              lattice(rsite,1) = i
              NBADS(i) = NBADS(i)+1
              NBADS(0) = NBADS(0)-1
              nbB(0) = nbB(0)-1

              nb_left(nx,ny,side,i,nsim) = nb_left(nx,ny,side,i,nsim)-1
              ext_flux_ind(nx,ny,side,i,nsim) = ext_flux_ind(nx,ny,side,i,nsim)-1

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   &                                  .or.(ext_flux_ind(nx,ny,side,i,nsim).eq.0)) ok = .true.

              CALL update_class_onesite(rsite,1)
              CALL update_class_twosite(nx,ny,rsite,1)

              DO k=top,right
                CALL neighbour(rsite,1,xn,yn,k)
```

```
            CALL update_class_twosite(nx,ny,xn,yn)
           ENDDO

          ENDIF

         ENDDO

        ENDIF

       ELSE IF (nb_left(nx,ny,side,i,nsim).lt.0) THEN

        IF (nbB(i).gt.0) THEN

         ok = .false.

         DO WHILE (.not.ok)

          rsite = INT(ran2(idum)*REAL(lsize(x),8))+1

          IF (lattice(rsite,1).eq.i) THEN

           lattice(rsite,1) = 0
           NBADS(i) = NBADS(i)-1
           NBADS(0) = NBADS(0)+1
           nbB(i) = nbB(i)-1

           nb_left(nx,ny,side,i,nsim) = nb_left(nx,ny,side,i,nsim)+1
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

           IF ((nbB(i).eq.0).or.(nb_left(nx,ny,side,i,nsim).eq.0)
   &                                    .or.(ext_flux_ind(nx,ny,side,i,nsim).eq.0)) ok = .true.

           CALL update_class_onesite(rsite,1)
           CALL update_class_twosite(nx,ny,rsite,1)

           DO k=top,right
            CALL neighbour(rsite,1,xn,yn,k)
            CALL update_class_twosite(nx,ny,xn,yn)
           ENDDO

          ENDIF

         ENDDO

        ENDIF

       ENDIF

      ENDIF

     ENDDO

    ELSE IF ((side.eq.left).and.(source(side))) THEN

     DO i=1,lsize(y)
      nbB(lattice(1,i)) = nbB(lattice(1,i))+1
     ENDDO

     DO i=1,nb_ads

      IF ((nb_left(nx,ny,side,i,nsim).ne.0).and.(ext_flux_ind(nx,ny,side,i,nsim).ne.0)) THEN

       IF (nb_left(nx,ny,side,i,nsim).gt.0) THEN

        IF (nbB(0).gt.0) THEN
```

```
          ok = .false.

          DO WHILE (.not.ok)

           rsite = INT(ran2(idum)*REAL(lsize(y),8))+1

           IF (lattice(1,rsite).eq.0) THEN

            lattice(1,rsite) = i
            NBADS(i) = NBADS(i)+1
            NBADS(0) = NBADS(0)-1
            nbB(0) = nbB(0)-1

            nb_left(nx,ny,side,i,nsim) = nb_left(nx,ny,side,i,nsim)-1
            ext_flux_ind(nx,ny,side,i,nsim) = ext_flux_ind(nx,ny,side,i,nsim)-1

            IF ((nbB(0).eq.0).or.(nb_left(nx,ny,side,i,nsim).eq.0)
&                                          .or.(ext_flux_ind(nx,ny,side,i,nsim).eq.0)) ok = .true.

            CALL update_class_onesite(1,rsite)
            CALL update_class_twosite(nx,ny,1,rsite)

            DO k=top,right
             CALL neighbour(1,rsite,xn,yn,k)
             CALL update_class_twosite(nx,ny,xn,yn)
            ENDDO

           ENDIF

          ENDDO

         ENDIF

        ELSE IF (nb_left(nx,ny,side,i,nsim).lt.0) THEN

         IF (nbB(i).gt.0) THEN

          ok = .false.

          DO WHILE (.not.ok)

           rsite = INT(ran2(idum)*REAL(lsize(y),8))+1

           IF (lattice(1,rsite).eq.i) THEN

            lattice(1,rsite) = 0
            NBADS(i) = NBADS(i)-1
            NBADS(0) = NBADS(0)+1
            nbB(i) = nbB(i)-1

            nb_left(nx,ny,side,i,nsim) = nb_left(nx,ny,side,i,nsim)+1
            ext_flux_ind(nx,ny,side,i,nsim) = ext_flux_ind(nx,ny,side,i,nsim)+1

            IF ((nbB(i).eq.0).or.(nb_left(nx,ny,side,i,nsim).eq.0)
&                                          .or.(ext_flux_ind(nx,ny,side,i,nsim).eq.0)) ok = .true.

            CALL update_class_onesite(1,rsite)
            CALL update_class_twosite(nx,ny,1,rsite)

            DO k=top,right
             CALL neighbour(1,rsite,xn,yn,k)
             CALL update_class_twosite(nx,ny,xn,yn)
            ENDDO

           ENDIF
```

```
            ENDDO

           ENDIF

          ENDIF

         ENDIF

        ENDDO

      ELSE IF ((side.eq.right).and.(source(side))) THEN

       DO i=1,lsize(y)
        nbB(lattice(lsize(x),i)) = nbB(lattice(lsize(x),i))+1
       ENDDO

       DO i=1,nb_ads

        IF ((nb_left(nx,ny,side,i,nsim).ne.0).and.(ext_flux_ind(nx,ny,side,i,nsim).ne.0)) THEN

         IF (nb_left(nx,ny,side,i,nsim).gt.0) THEN

          IF (nbB(0).gt.0) THEN

           ok = .false.

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

            rsite = INT(ran2(idum)*REAL(lsize(y),8))+1

            IF (lattice(lsize(x),rsite).eq.0) THEN

             lattice(lsize(x),rsite) = i
             NBADS(i) = NBADS(i)+1
             NBADS(0) = NBADS(0)-1
             nbB(0) = nbB(0)-1

             nb_left(nx,ny,side,i,nsim) = nb_left(nx,ny,side,i,nsim)-1
             ext_flux_ind(nx,ny,side,i,nsim) = ext_flux_ind(nx,ny,side,i,nsim)-1

             IF ((nbB(0).eq.0).or.(nb_left(nx,ny,side,i,nsim).eq.0)
     &                                    .or.(ext_flux_ind(nx,ny,side,i,nsim).eq.0)) ok = .true.

             CALL update_class_onesite(lsize(x),rsite)
             CALL update_class_twosite(nx,ny,lsize(x),rsite)

             DO k=top,right
              CALL neighbour(lsize(x),rsite,xn,yn,k)
              CALL update_class_twosite(nx,ny,xn,yn)
             ENDDO

            ENDIF

           ENDDO

          ENDIF

         ELSE IF (nb_left(nx,ny,side,i,nsim).lt.0) THEN

          IF (nbB(i).gt.0) THEN

           ok = .false.

           DO WHILE (.not.ok)

            rsite = INT(ran2(idum)*REAL(lsize(y),8))+1
```

```
            IF (lattice(lsize(x),rsite).eq.i) THEN

              lattice(lsize(x),rsite) = 0
              NBADS(i) = NBADS(i)-1
              NBADS(0) = NBADS(0)+1
              nbB(i) = nbB(i)-1

              nb_left(nx,ny,side,i,nsim) = nb_left(nx,ny,side,i,nsim)+1
              ext_flux_ind(nx,ny,side,i,nsim) = ext_flux_ind(nx,ny,side,i,nsim)+1

              IF ((nbB(i).eq.0).or.(nb_left(nx,ny,side,i,nsim).eq.0)
  &                                          .or.(ext_flux_ind(nx,ny,side,i,nsim).eq.0)) ok = .true.

              CALL update_class_onesite(lsize(x),rsite)
              CALL update_class_twosite(nx,ny,lsize(x),rsite)

              DO k=top,right
                CALL neighbour(lsize(x),rsite,xn,yn,k)
                CALL update_class_twosite(nx,ny,xn,yn)
              ENDDO

             ENDIF

           ENDDO

          ENDIF

         ENDIF

        ENDIF

       ENDDO

      ENDIF

    ENDDO

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
      DO i=1,nb_ads

        nb_left(nx,ny,side,i,nsim) = nb_left(nx,ny,side,i,nsim)+nb_to_add(nx,ny,side,i,nsim)

      ENDDO
    ENDDO

    btime_next(nx,ny,nsim) = btime_next(nx,ny,nsim) + int_boundary

    RETURN
    END
*-----------------------------------------------------------------------*
************************************************************************

************************************************************************
    SUBROUTINE update_ext_fluxes (nx,ny)
*-----------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: i,side, nx, ny
*-----------------------------------------------------------------------*
    DO side=top,right

      IF (source(side)) THEN

        DO i=1,nb_ads
```

```
      ext_flux_ind(nx,ny,side,i,nsim) = ext_flux_ind0(nx,ny,side,i)
     ENDDO

    ENDIF

   ENDDO

   DO side=top,right

    IF (source(side)) THEN

     DO i=1,nb_ads

      IF (ext_flux_ind(nx,ny,side,i,nsim).eq.0) THEN

       nb_left(nx,ny,side,i,nsim) = 0
       nb_to_add(nx,ny,side,i,nsim) = nb_left(nx,ny,side,i,nsim)

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

       nb_left(nx,ny,side,i,nsim) = CEILING(REAL(ext_flux_ind(nx,ny,side,i,nsim),8)*int_boundary/time_boundary)
       nb_to_add(nx,ny,side,i,nsim) = nb_left(nx,ny,side,i,nsim)

      ELSE

       nb_left(nx,ny,side,i,nsim) = INT(REAL(ext_flux_ind(nx,ny,side,i,nsim),8)*int_boundary/time_boundary)-1
       nb_to_add(nx,ny,side,i,nsim) = nb_left(nx,ny,side,i,nsim)

      ENDIF

     ENDDO

    ENDIF

   ENDDO

   time_fluxup_slots(nx,ny,nsim) = time_fluxup_slots(nx,ny,nsim) + 1
   time_fluxupdate(nx,ny,nsim) = REAL(time_fluxup_slots(nx,ny,nsim),8)*time_boundary

   RETURN
   END
*-----------------------------------------------------------------------------*
*******************************************************************************

*******************************************************************************
   SUBROUTINE init_classes ()
*-----------------------------------------------------------------------------*
   IMPLICIT NONE
   INCLUDE 'variables.inc'

   INTEGER :: i,j,p1,p2
*-----------------------------------------------------------------------------*
   nb_class = 0
   class(:,:,:) = 0
   class_coord(:,:)= -1

   DO i=1,nb_reac

    IF (type_reac(i).eq.1) THEN

     IF (class(0,-1,-1).eq.0) THEN
      nb_class = nb_class+1
      class(0,-1,-1) = nb_class
      CALL store_class_coords(nb_class,0,-1,-1)
     ENDIF
```

```
    ELSE IF ((type_reac(i).eq.2).or.(type_reac(i).eq.8)) THEN

      p1 = param_reac(i,1)

      IF (class(p1,-1,-1).eq.0) THEN
        nb_class = nb_class+1
        class(p1,-1,-1) = nb_class
        CALL store_class_coords(nb_class,p1,-1,-1)
      ENDIF

    ELSE IF (type_reac(i).eq.3) THEN

      IF (class(0,-1,-1).eq.0) THEN
        nb_class = nb_class+1
        class(0,-1,-1) = nb_class
        CALL store_class_coords(nb_class,0,-1,-1)
      ENDIF

      DO j=1,4

        IF (class(0,0,j).eq.0) THEN
          nb_class = nb_class+1
          class(0,0,j) = nb_class
          CALL store_class_coords(nb_class,0,0,j)
        ENDIF

      ENDDO

    ELSE IF (type_reac(i).eq.4) THEN

      p1 = param_reac(i,1)

      IF (class(p1,-1,-1).eq.0) THEN
        nb_class = nb_class+1
        class(p1,-1,-1) = nb_class
        CALL store_class_coords(nb_class,p1,-1,-1)
      ENDIF

      DO j=1,4

        IF (class(p1,p1,j).eq.0) THEN
          nb_class = nb_class+1
          class(p1,p1,j) = nb_class
          CALL store_class_coords(nb_class,p1,p1,j)
        ENDIF

      ENDDO

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  &                          .or.(type_reac(i).eq.11)) THEN

      p1 = param_reac(i,1)
      p2 = param_reac(i,2)

      IF (class(p1,-1,-1).eq.0) THEN
        nb_class = nb_class+1
        class(p1,-1,-1) = nb_class
        CALL store_class_coords(nb_class,p1,-1,-1)
      ENDIF

      IF (class(p2,-1,-1).eq.0) THEN
        nb_class = nb_class+1
        class(p2,-1,-1) = nb_class
        CALL store_class_coords(nb_class,p2,-1,-1)
      ENDIF
```

```
   DO j=1,4

    IF (class(p1,p2,j).eq.0) THEN
      nb_class = nb_class+1
      class(p1,p2,j) = nb_class
      CALL store_class_coords(nb_class,p1,p2,j)
    ENDIF

    IF (class(p2,p1,j).eq.0) THEN
      nb_class = nb_class+1
      class(p2,p1,j) = nb_class
      CALL store_class_coords(nb_class,p2,p1,j)
    ENDIF

   ENDDO

  ELSE IF ((type_reac(i).eq.6).or.(type_reac(i).eq.7)) THEN

   p1 = param_reac(i,1)

   IF (class(0,-1,-1).eq.0) THEN
    nb_class = nb_class+1
    class(0,-1,-1) = nb_class
    CALL store_class_coords(nb_class,0,-1,-1)
   ENDIF

   IF (class(p1,-1,-1).eq.0) THEN
    nb_class = nb_class+1
    class(p1,-1,-1) = nb_class
    CALL store_class_coords(nb_class,p1,-1,-1)
   ENDIF

   DO j=1,4

    IF (class(p1,0,j).eq.0) THEN
      nb_class = nb_class+1
      class(p1,0,j) = nb_class
      CALL store_class_coords(nb_class,p1,0,j)
    ENDIF

    IF (class(0,p1,j).eq.0) THEN
      nb_class = nb_class+1
      class(0,p1,j) = nb_class
      CALL store_class_coords(nb_class,0,p1,j)
    ENDIF

   ENDDO

  ELSE IF (type_reac(i).eq.9) THEN

   p1 = param_reac(i,2)

   IF (class(p1,-1,-1).eq.0) THEN
    nb_class = nb_class+1
    class(p1,-1,-1) = nb_class
    CALL store_class_coords(nb_class,p1,-1,-1)
   ENDIF

  ENDIF

 ENDDO

nb_proba = 0
proba_coord(:,:) = -1

DO i=1,nb_class
```

```
        IF (class_coord(i,2).eq.-1) THEN
          nb_proba = nb_proba+1
          proba_coord(nb_proba,1) = class_coord(i,1)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
          nb_proba = nb_proba+1
          proba_coord(nb_proba,1) = class_coord(i,1)
          proba_coord(nb_proba,2) = class_coord(i,2)
        ENDIF

      ENDDO

      IF (nb_class.gt.nb_class_max) THEN
        WRITE(*,*)
        WRITE(*,'(''*** ERROR: number of classes too large:'')')
        WRITE(*,'('' - nb_class ='',i4)') nb_class
        WRITE(*,'('' - nb_class_max ='',i4)') nb_class_max
        WRITE(*,'(''You may increase nb_class_max and re-compile the program.'')')
        WRITE(*,'(''Program terminated.'')')
        WRITE(*,*)
        STOP
      ENDIF

      RETURN
      END
*-----------------------------------------------------------------------*
*****************************************************************************

*****************************************************************************
      SUBROUTINE init_lattices ()
*-----------------------------------------------------------------------*
      IMPLICIT NONE
      INCLUDE 'variables.inc'

      INTEGER :: nx,ny,i
      INTEGER :: xlatt, ylatt
      INTEGER :: nb_rest_tot, sum
      INTEGER,DIMENSION(nb_ads_max) :: nb_rest
      REAL(KIND=8) :: ran2,ran
*-----------------------------------------------------------------------*

      WRITE(*,'(''----------------------------------------'')')
      WRITE(*,'(''- Creation of a'',i3,'' x'',i2,''  grid of teeth'')')teeth(x),teeth(y)
      WRITE(*,'(''- Teeth lattice size: '',i3,'' x '',i3)')lsize(x),lsize(y)
      WRITE(*,'(''----------------------------------------'')')
      WRITE(*,*)

      DO ny=1,teeth(y)
        DO nx = 1,teeth(x)

          finish = .false.
          lattice(:,:) = 0
          nb_rest(:) = 0
          NB(:) = 0
          NB_fin(:) = 0

          DO i=1,nb_ads
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
          ENDDO

          DO WHILE (.not.finish)

            xlatt = INT(ran2(idum)*REAL(lsize(x),8))+1
            ylatt = INT(ran2(idum)*REAL(lsize(y),8))+1
```

```
        IF (lattice(xlatt,ylatt).eq.0) THEN

        nb_rest_tot = 0

        DO i=1,nb_ads

         nb_rest(i) = (NB_fin(i)-NB(i))*priority(i)
         nb_rest_tot = nb_rest_tot+nb_rest(i)

        ENDDO

        IF (nb_rest_tot.ne.0) THEN

         insert_type = 1
         sum = 0

         ran = INT(ran2(idum)*REAL(nb_rest_tot),8)+1

         DO i=1,nb_ads-1
          sum = sum+nb_rest(i)
          IF ((ran.gt.sum).and.(ran.le.sum+nb_rest(i+1))) insert_type = i+1
         ENDDO

        ELSE

         insert_type = 0

        ENDIF

         lattice(xlatt,ylatt) = insert_type
         NB(insert_type) = NB(insert_type)+1

        ENDIF

        finish = .true.

        DO i=1,nb_ads
         IF (NB(i).lt.NB_fin(i)) finish = .false.
        ENDDO

       ENDDO

       CALL write_configuration (nx,ny)

      ENDDO
    ENDDO

    RETURN
    END
*-----------------------------------------------------------------------*
*************************************************************************

*************************************************************************
    SUBROUTINE ext_fluxes_distr ()
*-----------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: i, side, par, nx_tooth, ny_tooth
    LOGICAL,DIMENSION(teeth_max) :: selected
    REAL(KIND=8) :: ran2
    LOGICAL :: ok
*-----------------------------------------------------------------------*
    IF (bound_fluxes) THEN

       DO side=top,right
```

```
       IF (source(side)) THEN

         DO i=1,nb_ads

           IF (side.eq.top) THEN

            IF (ext_flux_gen0(side,i).gt.0) THEN

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

                ext_flux_ind0(:,teeth(y),side,i) = ext_flux_gen0(side,i)/teeth(x)

               ELSE

                ext_flux_ind0(:,teeth(y),side,i) = INT(ext_flux_gen0(side,i)/teeth(x))
                par = mod(ext_flux_gen0(side,i),teeth(x))
                ok=.false.
                selected(:)=.false.

                DO WHILE (.not.ok)

                  nx_tooth = INT(ran2(idum)*teeth(x))+1

                  IF (.not.selected(nx_tooth)) THEN
                     ext_flux_ind0(nx_tooth,teeth(y),side,i) = ext_flux_ind0(nx_tooth,teeth(y),side,i) + 1
                     par = par - 1
                  ENDIF

                   selected(nx_tooth)=.true.

                   IF (par.eq.0) ok=.true.

                 ENDDO

                ENDIF

              ELSEIF (ext_flux_gen0(side,i).lt.0) THEN

             IF (mod(ext_flux_gen0(side,i),teeth(x)).eq.0) THEN

                ext_flux_ind0(:,teeth(y),side,i) = ext_flux_gen0(side,i)/teeth(x)

               ELSE

                  ext_flux_ind0(:,teeth(y),side,i) = INT(ext_flux_gen0(side,i)/teeth(x))
                  par = mod(ext_flux_gen0(side,i),teeth(x))
                  ok=.false.
                  selected(:)=.false.

                  DO WHILE (.not.ok)

                    nx_tooth = INT(ran2(idum)*teeth(x))+1

                    IF (.not.selected(nx_tooth)) THEN
                      ext_flux_ind0(nx_tooth,teeth(y),side,i) = ext_flux_ind0(nx_tooth,teeth(y),side,i) - 1
                      par = par + 1
                     ENDIF

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

                     IF (par.eq.0) ok=.true.

                   ENDDO

                  ENDIF
```

```
                    ENDIF

                  ELSE IF (side.eq.bottom) THEN

                    IF (ext_flux_gen0(side,i).gt.0) THEN

                     IF (mod(ext_flux_gen0(side,i),teeth(x)).eq.0) THEN

                       ext_flux_ind0(:,1,side,i) = ext_flux_gen0(side,i)/teeth(x)

                     ELSE

                      ext_flux_ind0(:,1,side,i) = INT(ext_flux_gen0(side,i)/teeth(x))
                      par = mod(ext_flux_gen0(side,i),teeth(x))
                      ok=.false.
                      selected(:)=.false.

                      DO WHILE (.not.ok)

                       nx_tooth = INT(ran2(idum)*teeth(x))+1

                       IF (.not.selected(nx_tooth)) THEN
                        ext_flux_ind0(nx_tooth,1,side,i) = ext_flux_ind0(nx_tooth,1,side,i) + 1
                        par = par - 1
                       ENDIF

                       selected(nx_tooth)=.true.

                       IF (par.eq.0) ok=.true.

                      ENDDO

                     ENDIF

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

                     IF (mod(ext_flux_gen0(side,i),teeth(x)).eq.0) THEN

                       ext_flux_ind0(:,1,side,i) = ext_flux_gen0(side,i)/teeth(x)

                     ELSE

                      ext_flux_ind0(:,1,side,i) = INT(ext_flux_gen0(side,i)/teeth(x))
                      par = mod(ext_flux_gen0(side,i),teeth(x))
                      ok=.false.
                      selected(:)=.false.

                      DO WHILE (.not.ok)

                       nx_tooth = INT(ran2(idum)*teeth(x))+1

                       IF (.not.selected(nx_tooth)) THEN
                        ext_flux_ind0(nx_tooth,1,side,i) = ext_flux_ind0(nx_tooth,1,side,i) - 1
                        par = par + 1
                       ENDIF

                       selected(nx_tooth)=.true.

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

                      ENDDO

                     ENDIF

                    ENDIF
```

```
                    ELSE IF (side.eq.left) THEN

                      IF (ext_flux_gen0(side,i).gt.0) THEN

                       IF (mod(ext_flux_gen0(side,i),teeth(y)).eq.0) THEN

                         ext_flux_ind0(1,:,side,i) = ext_flux_gen0(side,i)/teeth(y)

                       ELSE

                        ext_flux_ind0(1,:,side,i) = INT(ext_flux_gen0(side,i)/teeth(y))
                        par = mod(ext_flux_gen0(side,i),teeth(y))
                        ok=.false.
                        selected(:)=.false.

                        DO WHILE (.not.ok)

                         ny_tooth = INT(ran2(idum)*teeth(y))+1

                         IF (.not.selected(ny_tooth)) THEN
                          ext_flux_ind0(1,ny_tooth,side,i) = ext_flux_ind0(1,ny_tooth,side,i) + 1
                          par = par - 1
                         ENDIF

                         selected(ny_tooth)=.true.

                         IF (par.eq.0) ok=.true.

                        ENDDO

                       ENDIF

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

                       IF (mod(ext_flux_gen0(side,i),teeth(y)).eq.0) THEN

                         ext_flux_ind0(1,:,side,i) = ext_flux_gen0(side,i)/teeth(y)

                       ELSE

                        ext_flux_ind0(1,:,side,i) = INT(ext_flux_gen0(side,i)/teeth(y))
                        par = mod(ext_flux_gen0(side,i),teeth(y))
                        ok=.false.
                        selected(:)=.false.

                        DO WHILE (.not.ok)

                         ny_tooth = INT(ran2(idum)*teeth(y))+1

                         IF (.not.selected(ny_tooth)) THEN
                          ext_flux_ind0(1,ny_tooth,side,i) = ext_flux_ind0(1,ny_tooth,side,i) - 1
                          par = par + 1
                         ENDIF

                         selected(ny_tooth)=.true.

                         IF (par.eq.0) ok=.true.

                        ENDDO

                       ENDIF

                      ENDIF

                    ELSE IF (side.eq.right) THEN
```

```
                IF (ext_flux_gen0(side,i).gt.0) THEN

                 IF (mod(ext_flux_gen0(side,i),teeth(y)).eq.0) THEN

                   ext_flux_ind0(teeth(x),:,side,i) = ext_flux_gen0(side,i)/teeth(y)

                 ELSE

                  ext_flux_ind0(teeth(x),:,side,i) = INT(ext_flux_gen0(side,i)/teeth(y))
                  par = mod(ext_flux_gen0(side,i),teeth(y))
                  ok=.false.
                  selected(:)=.false.

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

                    ny_tooth = INT(ran2(idum)*teeth(y))+1

                   IF (.not.selected(ny_tooth)) THEN
                    ext_flux_ind0(teeth(x),ny_tooth,side,i) = ext_flux_ind0(teeth(x),ny_tooth,side,i) + 1
                    par = par - 1
                   ENDIF

                   selected(ny_tooth)=.true.

                   IF (par.eq.0) ok=.true.

                  ENDDO

                 ENDIF

                ELSEIF (ext_flux_gen0(side,i).lt.0) THEN

                 IF (mod(ext_flux_gen0(side,i),teeth(y)).eq.0) THEN

                   ext_flux_ind0(teeth(x),:,side,i) = ext_flux_gen0(side,i)/teeth(y)

                 ELSE

                  ext_flux_ind0(teeth(x),:,side,i) = INT(ext_flux_gen0(side,i)/teeth(y))
                  par = mod(ext_flux_gen0(side,i),teeth(y))
                  ok=.false.
                  selected(:)=.false.

                  DO WHILE (.not.ok)

                    ny_tooth = INT(ran2(idum)*teeth(y))+1

                   IF (.not.selected(ny_tooth)) THEN
                    ext_flux_ind0(teeth(x),ny_tooth,side,i) = ext_flux_ind0(teeth(x),ny_tooth,side,i) - 1
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                   ENDIF

                   selected(ny_tooth)=.true.

                   IF (par.eq.0) ok=.true.

                  ENDDO

                 ENDIF

                ENDIF

             ENDIF

          ENDDO
```

```
            ENDIF

          ENDDO

        ENDIF

      RETURN
      END
*-----------------------------------------------------------------------------*
*******************************************************************************

*******************************************************************************
      SUBROUTINE init_ext_fluxes ()
*-----------------------------------------------------------------------------*
      IMPLICIT NONE
      INCLUDE 'variables.inc'

      INTEGER :: i, side, max, nx, ny, nb_sim
*-----------------------------------------------------------------------------*
      IF (bound_fluxes) THEN

       DO side=top,right

        IF (source(side)) THEN

         DO ny = 1,teeth(y)

          DO nx = 1,teeth(x)

           DO i=1,nb_ads
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
           ENDDO

          ENDDO

         ENDDO

        ENDIF

       ENDDO

       max = 1

       DO side=top,right

         IF (source(side)) THEN

         DO ny = 1,teeth(y)

          DO nx = 1,teeth(x)

           DO i=1,nb_ads
            IF (ABS(ext_flux_ind0(nx,ny,side,i)).gt.max) max = ABS(ext_flux_ind0(nx,ny,side,i))
           ENDDO

          ENDDO

         ENDDO

        ENDIF

       ENDDO

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
      DO side=top,right

       IF (source(side)) THEN

        DO ny = 1,teeth(y)

         DO nx = 1,teeth(x)

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

            IF (ext_flux_ind0(nx,ny,side,i).eq.0) THEN

              nb_left(nx,ny,side,i,:) = 0
              nb_to_add(nx,ny,side,i,:) = nb_left(nx,ny,side,i,nb_simul)

            ELSEIF (ext_flux_ind0(nx,ny,side,i).gt.0) THEN

              nb_left(nx,ny,side,i,:) = CEILING(REAL(ext_flux_ind0(nx,ny,side,i),8)*int_boundary/time_boundary)
              nb_to_add(nx,ny,side,i,:) = nb_left(nx,ny,side,i,nb_simul)

            ELSE

              nb_left(nx,ny,side,i,:) = INT(REAL(ext_flux_ind0(nx,ny,side,i),8)*int_boundary/time_boundary)-1
              nb_to_add(nx,ny,side,i,:) = nb_left(nx,ny,side,i,nb_simul)

            ENDIF

           ENDDO

          ENDDO

         ENDDO

        ENDIF

       ENDDO

       btime_next(:,:,:) = begin_time
       time_fluxup_slots(:,:,:) = 1
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

     ENDIF

     RETURN
     END
*-------------------------------------------------------------------------*
*****************************************************************************

*****************************************************************************
     SUBROUTINE init_class_sites (xbox,ybox)
*-------------------------------------------------------------------------*
     IMPLICIT NONE
     INCLUDE 'variables.inc'

     INTEGER :: i,j
     INTEGER :: xbox,ybox
*-------------------------------------------------------------------------*
     nb_site_class(:) = 0

     realnb_site_class(:) = 0
     class_site(:,:,:) = 0

     site_clnb(:,:) = 0
     site_clid(:,:,:) = 0
     site_clsite(:,:,:) = 0
```

```
    enabled(:,:) = .false.
    clean = .false.
    gc = .false.

      DO i=1,lsize(x)
        DO j=1,lsize(y)
          CALL initclass_onesite(i,j)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
        ENDDO
      ENDDO

    gc = .true.

    RETURN
    END
*-------------------------------------------------------------------------*
*****************************************************************************

*****************************************************************************
    SUBROUTINE initclass_onesite (xsite,ysite)
*-------------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: xsite,ysite
    INTEGER :: stype,ind
    REAL(KIND=8) :: proportion
    LOGICAL :: outside
*-------------------------------------------------------------------------*
    IF (.not.outside(xsite,ysite)) THEN

      stype = lattice(xsite,ysite)
      ind = class(stype,-1,-1)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
      nb_site_class(ind) = nb_site_class(ind)+1

      enabled(ind,nb_site_class(ind)) = .true.

      class_site(ind,nb_site_class(ind),x) = xsite
      class_site(ind,nb_site_class(ind),y) = ysite

      IF (gc) THEN

        proportion = REAL(nb_site_class(ind),8)/REAL(nb_sites,8)

        IF (proportion.gt.1.0) THEN
          clean = .true.
        ENDIF

      ENDIF

      site_clnb(xsite,ysite) = site_clnb(xsite,ysite)+1
      site_clid(xsite,ysite,site_clnb(xsite,ysite)) = ind
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

    ENDIF

    RETURN
    END
*-------------------------------------------------------------------------*
*****************************************************************************

*****************************************************************************
    SUBROUTINE initclass_twosite (xbox,ybox,xsite,ysite)
*-------------------------------------------------------------------------*
    IMPLICIT NONE
```

```
      INCLUDE 'variables.inc'

      INTEGER :: xbox,ybox,a
      INTEGER :: xsite,ysite
      INTEGER :: k,n,side
      INTEGER :: stype,ind
      INTEGER :: xn,yn
      INTEGER :: nn,nb_types
      INTEGER,DIMENSION(4) :: nntype,nb_nntype
      LOGICAL :: new,outside,ok
      REAL(KIND=8) :: proportion
*-------------------------------------------------------------------------------*
      IF (.not.outside(xsite,ysite)) THEN

       stype = lattice(xsite,ysite)

       nb_types = 0
       nntype(:) = -1
       nb_nntype(:) = 0

       DO n=top,right

        CALL neighbour(xsite,ysite,xn,yn,n)

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

          nn = lattice(xn,yn)
          new = .true.

          DO k=1,nb_types

           IF (nntype(k).eq.nn) THEN
             new = .false.
             nb_nntype(k) = nb_nntype(k)+1
           ENDIF

          ENDDO

          IF (new) THEN
            nb_types = nb_types+1
            nntype(nb_types) = nn
            nb_nntype(nb_types) = 1
          ENDIF

        ELSEIF ((outside(xn,yn)).and.(stype.ne.0)) THEN

          IF ((teeth(y).eq.1).and.(n.ne.1).and.(n.ne.2)) THEN

            IF (teeth(x).gt.1) THEN

              IF ((xbox.eq.1).and.(n.ne.3)) THEN

                nn = 0
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

                DO k=1,nb_types
                 IF (nntype(k).eq.nn) THEN
                   new = .false.
                   nb_nntype(k) = nb_nntype(k)+1
                 ENDIF
                ENDDO

               IF (new) THEN
                 nb_types = nb_types+1
                 nntype(nb_types) = nn
                 nb_nntype(nb_types) = 1
```

```
            ENDIF

         ELSEIF ((xbox.eq.teeth(x)).and.(n.ne.4)) THEN

            nn = 0
            new = .true.

            DO k=1,nb_types
              IF (nntype(k).eq.nn) THEN
                new = .false.
                nb_nntype(k) = nb_nntype(k)+1
              ENDIF
            ENDDO

            IF (new) THEN
              nb_types = nb_types+1
              nntype(nb_types) = nn
              nb_nntype(nb_types) = 1
            ENDIF

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

            nn = 0
            new = .true.

            DO k=1,nb_types
              IF (nntype(k).eq.nn) THEN
                new = .false.
                nb_nntype(k) = nb_nntype(k)+1
              ENDIF
            ENDDO

            IF (new) THEN
              nb_types = nb_types+1
              nntype(nb_types) = nn
              nb_nntype(nb_types) = 1
            ENDIF

          ENDIF

         ENDIF

         ELSEIF (teeth(y).gt.1) THEN

           IF ((teeth(x).eq.1).and.(n.ne.3).and.(n.ne.4)) THEN

            IF ((ybox.eq.1).and.(n.ne.2)) THEN

             nn = 0
             new = .true.

            DO k=1,nb_types
              IF (nntype(k).eq.nn) THEN
                new = .false.
                nb_nntype(k) = nb_nntype(k)+1
              ENDIF
            ENDDO

            IF (new) THEN
              nb_types = nb_types+1
              nntype(nb_types) = nn
              nb_nntype(nb_types) = 1
            ENDIF

         ELSEIF ((ybox.eq.teeth(y)).and.(n.ne.1)) THEN
```

```
         nn = 0
         new = .true.

         DO k=1,nb_types
           IF (nntype(k).eq.nn) THEN
             new = .false.
             nb_nntype(k) = nb_nntype(k)+1
           ENDIF
         ENDDO

         IF (new) THEN
           nb_types = nb_types+1
           nntype(nb_types) = nn
           nb_nntype(nb_types) = 1
         ENDIF

       ELSEIF ((ybox.gt.1).and.(ybox.lt.teeth(y))) THEN

         nn = 0
         new = .true.

         DO k=1,nb_types
           IF (nntype(k).eq.nn) THEN
             new = .false.
             nb_nntype(k) = nb_nntype(k)+1
           ENDIF
         ENDDO

         IF (new) THEN
           nb_types = nb_types+1
           nntype(nb_types) = nn
           nb_nntype(nb_types) = 1
         ENDIF

        ENDIF

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

        IF ((xbox.eq.1).and.(n.ne.3)) THEN

           IF ((ybox.eq.1).and.(n.ne.2)) THEN

            nn = 0
            new = .true.

            DO k=1,nb_types
             IF (nntype(k).eq.nn) THEN
               new = .false.
               nb_nntype(k) = nb_nntype(k)+1
             ENDIF
           ENDDO

           IF (new) THEN
             nb_types = nb_types+1
             nntype(nb_types) = nn
             nb_nntype(nb_types) = 1
           ENDIF

        ELSEIF ((ybox.eq.teeth(y)).and.(n.ne.1)) THEN

           nn = 0
           new = .true.

           DO k=1,nb_types
             IF (nntype(k).eq.nn) THEN
               new = .false.
```

```
      nb_nntype(k) = nb_nntype(k)+1
    ENDIF
  ENDDO

  IF (new) THEN
    nb_types = nb_types+1
    nntype(nb_types) = nn
    nb_nntype(nb_types) = 1
  ENDIF

ELSEIF ((ybox.gt.1).and.(ybox.lt.teeth(y))) THEN

  nn = 0
  new = .true.

  DO k=1,nb_types
    IF (nntype(k).eq.nn) THEN
      new = .false.
      nb_nntype(k) = nb_nntype(k)+1
    ENDIF
  ENDDO

  IF (new) THEN
    nb_types = nb_types+1
    nntype(nb_types) = nn
    nb_nntype(nb_types) = 1
  ENDIF

 ENDIF

ELSEIF ((xbox.eq.teeth(x)).and.(n.ne.4)) THEN

 IF ((ybox.eq.1).and.(n.ne.2)) THEN

  nn = 0
  new = .true.

  DO k=1,nb_types
    IF (nntype(k).eq.nn) THEN
      new = .false.
      nb_nntype(k) = nb_nntype(k)+1
    ENDIF
  ENDDO

  IF (new) THEN
    nb_types = nb_types+1
    nntype(nb_types) = nn
    nb_nntype(nb_types) = 1
  ENDIF

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

  nn = 0
  new = .true.

  DO k=1,nb_types
    IF (nntype(k).eq.nn) THEN
      new = .false.
      nb_nntype(k) = nb_nntype(k)+1
    ENDIF
  ENDDO

  IF (new) THEN
    nb_types = nb_types+1
    nntype(nb_types) = nn
    nb_nntype(nb_types) = 1
```

```
    ENDIF

  ELSEIF ((ybox.gt.1).and.(ybox.lt.teeth(y))) THEN

     nn = 0
     new = .true.

    DO k=1,nb_types
     IF (nntype(k).eq.nn) THEN
       new = .false.
       nb_nntype(k) = nb_nntype(k)+1
     ENDIF
    ENDDO

    IF (new) THEN
     nb_types = nb_types+1
     nntype(nb_types) = nn
     nb_nntype(nb_types) = 1
    ENDIF

  ENDIF

 ELSEIF ((xbox.gt.1).and.(xbox.lt.teeth(x))) THEN

    IF ((ybox.eq.1).and.(n.ne.2)) THEN

    nn = 0
    new = .true.

    DO k=1,nb_types
     IF (nntype(k).eq.nn) THEN
       new = .false.
       nb_nntype(k) = nb_nntype(k)+1
     ENDIF
    ENDDO

    IF (new) THEN
     nb_types = nb_types+1
     nntype(nb_types) = nn
     nb_nntype(nb_types) = 1
    ENDIF

  ELSEIF ((ybox.eq.teeth(y)).and.(n.ne.1)) THEN

    nn = 0
    new = .true.

    DO k=1,nb_types
     IF (nntype(k).eq.nn) THEN
       new = .false.
       nb_nntype(k) = nb_nntype(k)+1
     ENDIF
    ENDDO

    IF (new) THEN
     nb_types = nb_types+1
     nntype(nb_types) = nn
     nb_nntype(nb_types) = 1
    ENDIF

  ELSEIF ((ybox.gt.1).and.(ybox.lt.teeth(y))) THEN

    nn = 0
    new = .true.

    DO k=1,nb_types
```

```
              IF (nntype(k).eq.nn) THEN
                new = .false.
                nb_nntype(k) = nb_nntype(k)+1
              ENDIF
            ENDDO

            IF (new) THEN
              nb_types = nb_types+1
              nntype(nb_types) = nn
              nb_nntype(nb_types) = 1
            ENDIF

          ENDIF

        ENDIF

      ENDIF

    ENDIF

    ENDIF

    ENDDO

    DO k=1,nb_types

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

      IF (ind.ne.0) THEN

        realnb_site_class(ind) = realnb_site_class(ind)+1
        nb_site_class(ind) = nb_site_class(ind)+1

        enabled(ind,nb_site_class(ind)) = .true.

        class_site(ind,nb_site_class(ind),x) = xsite
        class_site(ind,nb_site_class(ind),y) = ysite

        IF (gc) THEN

          proportion = REAL(nb_site_class(ind),8)/REAL(nb_sites,8)

          IF (proportion.gt.1.0) THEN
            clean = .true.
          ENDIF

        ENDIF

        site_clnb(xsite,ysite) = site_clnb(xsite,ysite)+1
        site_clid(xsite,ysite,site_clnb(xsite,ysite)) = ind
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

      ENDIF

    ENDDO

  ENDIF

  RETURN
  END
*-----------------------------------------------------------------------*
*************************************************************************

*************************************************************************
  SUBROUTINE calc_kreac ()
*-----------------------------------------------------------------------*
```

```
      IMPLICIT NONE
      INCLUDE 'variables.inc'

      INTEGER :: i
      REAL(KIND=8),DIMENSION(nb_gas) :: k_ads
*-----------------------------------------------------------------------*
      DO i=1,nb_gas

        k_ads(i) = sticking_gas(i)*(N_Avog/site_density)*
     &                                press(i)/sqrt(2.0*pi*weight_gas(i)*R*temperature)

      ENDDO

      DO i=1,nb_reac

        IF ((type_reac(i).eq.1).or.(type_reac(i).eq.3)) THEN

          k_reac(i) = k_ads(param_reac(i,1))

        ELSE IF (type_reac(i).eq.9) THEN

          k_reac(i) = k_ads(param_reac(i,1))*
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

        ELSE

          k_reac(i) = preexp(i)*dexp(-real(actenergy(i),8)/(R*temperature))

        ENDIF

      ENDDO

      RETURN
      END
*-----------------------------------------------------------------------*
*************************************************************************

*************************************************************************
      SUBROUTINE calc_proba ()
*-----------------------------------------------------------------------*
      IMPLICIT NONE
      INCLUDE 'variables.inc'

      INTEGER :: i, j
      INTEGER :: s1, s2, ind
      REAL(KIND=8) :: numer
*-----------------------------------------------------------------------*
      proba(:,:) = 0.0

      DO i=1,nb_proba

        s1 = proba_coord(i,1)
        s2 = proba_coord(i,2)

        IF (s2.eq.-1) THEN

          ind = class(s1,-1,-1)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

        ELSE

          numer = 0.0

          DO j=1,4

            ind = class(s2,s1,j)
```

```
          numer = numer+j*realnb_site_class(ind)

        ENDDO

        ind = class(s2,-1,-1)

        IF (realnb_site_class(ind).ne.0) THEN
           proba(s1,s2) = numer/(4.0*REAL(realnb_site_class(ind),8))
        ENDIF

      ENDIF

    ENDDO

    RETURN
    END
*-----------------------------------------------------------------------*
*************************************************************************

*************************************************************************
    SUBROUTINE calc_gamma ()
*-----------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: i,p1,p2
*-----------------------------------------------------------------------*
    gamma(:) = 0.0

    DO i=1,nb_reac

      IF (type_reac(i).eq.1) THEN

        gamma(i) = k_reac(i)*proba(0,-1)

      ELSE IF ((type_reac(i).eq.2).or.(type_reac(i).eq.8)) THEN

        p1 = param_reac(i,1)
        gamma(i) = k_reac(i)*proba(p1,-1)

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

        gamma(i) = k_reac(i)*proba(0,-1)*proba(0,0)

      ELSE IF (type_reac(i).eq.4) THEN

        p1 = param_reac(i,1)
        gamma(i) = k_reac(i)*proba(p1,-1)*proba(p1,p1)

      ELSE IF ((type_reac(i).eq.5).or.(type_reac(i).eq.10)
     &                    .or.(type_reac(i).eq.11)) THEN

        p1 = param_reac(i,1)
        p2 = param_reac(i,2)
        gamma(i) = k_reac(i)*(proba(p1,-1)*proba(p2,p1)+proba(p2,-1)*proba(p1,p2))

        IF (p1.eq.p2) gamma(i) = gamma(i)/2

      ELSE IF ((type_reac(i).eq.6).or.(type_reac(i).eq.7)) THEN

        p1 = param_reac(i,1)
        gamma(i) = k_reac(i)*(proba(p1,-1)*proba(0,p1)+proba(0,-1)*proba(p1,0))

      ELSE IF (type_reac(i).eq.9) THEN

        p1 = param_reac(i,1)
```

```
        p2 = param_reac(i,2)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

      ENDIF

    ENDDO

    RETURN
    END
*------------------------------------------------------------------------------*
*******************************************************************************

*******************************************************************************
    SUBROUTINE make_reaction_1site (xbox,ybox,rnum)
*------------------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: rnum
    INTEGER :: xbox,ybox
    INTEGER :: i,p1,p2,p3
    INTEGER :: ind,nsite,xn,yn
    INTEGER :: xx1,yy1
    REAL(KIND=8) :: ran2
    LOGICAL :: ok,outside
*------------------------------------------------------------------------------*
    p1 = param_reac(rnum,1)
    p2 = param_reac(rnum,2)
    p3 = param_reac(rnum,3)

    IF (type_reac(rnum).eq.1) THEN

      ind = class(0,-1,-1)
      ok = .false.

      DO WHILE (.not.ok)

        nsite = INT(ran2(idum)*nb_site_class(ind))+1
        xx1 = class_site(ind,nsite,x)
        yy1 = class_site(ind,nsite,y)

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

      ENDDO

      lattice(xx1,yy1) = p2
      NBGAS(p1) = NBGAS(p1)-1
      NBADS(p2) = NBADS(p2)+1
      NBADS(0) = NBADS(0)-1

    ELSE IF (type_reac(rnum).eq.2) THEN

      ind = class(p1,-1,-1)
      ok = .false.

      DO WHILE (.not.ok)

        nsite = INT(ran2(idum)*nb_site_class(ind))+1
        xx1 = class_site(ind,nsite,x)
        yy1 = class_site(ind,nsite,y)

        IF (enabled(ind,nsite).and.(.not.outside(xx1,yy1))) ok = .true.

      ENDDO

      lattice(xx1,yy1) = 0
```

```
      NBADS(p1) = NBADS(p1)-1
      NBGAS(p2) = NBGAS(p2)+1
      NBADS(0) = NBADS(0)+1

     ELSE IF (type_reac(rnum).eq.8) THEN

      ind = class(p1,-1,-1)
      ok = .false.

      DO WHILE (.not.ok)

       nsite = INT(ran2(idum)*nb_site_class(ind))+1
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
       yy1 = class_site(ind,nsite,y)

       IF (enabled(ind,nsite).and.(.not.outside(xx1,yy1))) ok = .true.

      ENDDO

      lattice(xx1,yy1) = p2
      NBADS(p1) = NBADS(p1)-1
      NBADS(p2) = NBADS(p2)+1

     ELSE IF (type_reac(rnum).eq.9) THEN

      ind = class(p2,-1,-1)
      ok = .false.

      DO WHILE (.not.ok)

       nsite = INT(ran2(idum)*nb_site_class(ind))+1
       xx1 = class_site(ind,nsite,x)
       yy1 = class_site(ind,nsite,y)

       IF (enabled(ind,nsite).and.(.not.outside(xx1,yy1))) ok = .true.

      ENDDO

      lattice(xx1,yy1) = p3
      NBGAS(p1) = NBGAS(p1)-1
      NBADS(p2) = NBADS(p2)-1
      NBADS(p3) = NBADS(p3)+1

     ELSE

      WRITE(*,*)
      WRITE(*,'(''ERROR: '',i2,'' is not 1-site chemical reaction'')')rnum
      WRITE(*,'(''Program terminated.'')')
      WRITE(*,*)
      STOP

     ENDIF

     CALL update_class_onesite(xx1,yy1)
     CALL update_class_twosite(xbox,ybox,xx1,yy1)

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
      CALL neighbour(xx1,yy1,xn,yn,i)
      CALL update_class_twosite(xbox,ybox,xn,yn)
     ENDDO

     RETURN
     END
*------------------------------------------------------------------------------*
*******************************************************************************
```

```
********************************************************************************
     SUBROUTINE make_reaction_2site (xbox,ybox,rnum)
*-------------------------------------------------------------------------------*
     IMPLICIT NONE
     INCLUDE 'variables.inc'

     INTEGER :: rnum,ind,side
     INTEGER :: i,j,k,p1,p2,p3,s1,s2
     INTEGER :: site1,site2
     INTEGER :: xbox,ybox
     INTEGER :: nbc, nsite, cl, neigh
     INTEGER,DIMENSION(2) :: suppr
     INTEGER :: xx1, yy1, xn, yn, xx2, yy2
     REAL(KIND=8) :: ran,ran2,sum
     INTEGER,DIMENSION(8) :: idclass
     INTEGER,DIMENSION(8) :: Cxym
     INTEGER :: Cxytot
     LOGICAL :: ok,outside,okk
*-------------------------------------------------------------------------------*
     idclass(:) = -1
     p1 = param_reac(rnum,1)
     p2 = param_reac(rnum,2)
     p3 = param_reac(rnum,3)

     IF (type_reac(rnum).eq.3) THEN

      s1 = 0
      s2 = 0

      nbc = 0
      Cxytot = 0

      DO i=1,4

       ind = class(s1,s2,i)

       IF (realnb_site_class(ind).gt.0) THEN
        nbc = nbc+1
        idclass(nbc) = ind
        Cxym(nbc) = i*realnb_site_class(ind)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
       ENDIF

      ENDDO

      cl = 1
      sum = 0.0
      ran = ran2(idum)*REAL(Cxytot,8)

      DO i=1,nbc-1

       sum = sum+REAL(Cxym(i),8)

       IF ((ran.gt.sum).and.(ran.lt.sum+REAL(Cxym(i+1),8))) cl = i+1

      ENDDO

      ind = idclass(cl)
      ok = .false.

      DO WHILE (.not.ok)

       nsite = INT(ran2(idum)*REAL(nb_site_class(ind),8))+1

       IF (enabled(ind,nsite)) THEN
```

```
          ok = .true.

          xx1 = class_site(ind,nsite,x)
          yy1 = class_site(ind,nsite,y)

         ENDIF

        ENDDO

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
        ok = .false.

        DO WHILE (.not.ok)

         neigh = mod(neigh,4)+1

         CALL neighbour(xx1,yy1,xn,yn,neigh)

         IF ((.not.outside(xn,yn)).and.
     &       (lattice(xn,yn).eq.site2)) ok = .true.

        ENDDO

        xx2 = xn
        yy2 = yn

        lattice(xx1,yy1) = p2
        lattice(xx2,yy2) = p2

        NBGAS(p1) = NBGAS(p1)-1
        NBADS(0) = NBADS(0)-2
        NBADS(p2) = NBADS(p2)+2

      ELSE IF (type_reac(rnum).eq.4) THEN

        s1 = p1
        s2 = p1

        nbc = 0
        Cxytot = 0

        DO i=1,4

         ind = class(s1,s2,i)

         IF (realnb_site_class(ind).gt.0) THEN
           nbc = nbc+1
           idclass(nbc) = ind
           Cxym(nbc) = i*realnb_site_class(ind)
           Cxytot = Cxytot+Cxym(nbc)
         ENDIF

        ENDDO

        cl = 1
        sum = 0.0
        ran = ran2(idum)*REAL(Cxytot,8)

        DO i=1,nbc-1

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

         IF ((ran.gt.sum).and.(ran.lt.sum+REAL(Cxym(i+1),8))) cl = i+1

        ENDDO
```

```
    ind = idclass(cl)
    ok = .false.

    DO WHILE (.not.ok)

     nsite = INT(ran2(idum)*REAL(nb_site_class(ind),8))+1

     IF (enabled(ind,nsite)) ok = .true.

      xx1 = class_site(ind,nsite,x)
      yy1 = class_site(ind,nsite,y)
      site1 = lattice(xx1,yy1)
      site2 = class_coord(ind,2)

    ENDDO

    neigh = INT(ran2(idum)*4)+1
    ok = .false.

    DO WHILE (.not.ok)

     neigh = mod(neigh,4)+1

     CALL neighbour(xx1,yy1,xn,yn,neigh)

     IF ((.not.outside(xn,yn)).and.
  &       (lattice(xn,yn).eq.site2)) ok = .true.

    ENDDO

    xx2 = xn
    yy2 = yn

    lattice(xx1,yy1) = 0
    lattice(xx2,yy2) = 0

    NBADS(p1) = NBADS(p1)-2
    NBADS(0) = NBADS(0)+2
    NBGAS(p2) = NBGAS(p2)+1

   ELSE IF (type_reac(rnum).eq.5) THEN

    s1 = p1
    s2 = p2

    nbc = 0
    Cxytot = 0

    DO i=1,4

     ind = class(s1,s2,i)

     IF (realnb_site_class(ind).gt.0) THEN
      nbc = nbc+1
      idclass(nbc) = ind
      Cxym(nbc) = i*realnb_site_class(ind)
      Cxytot = Cxytot+Cxym(nbc)
     ENDIF

     ind = class(s2,s1,i)

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
      nbc = nbc+1
      idclass(nbc) = ind
      Cxym(nbc) = i*realnb_site_class(ind)
      Cxytot = Cxytot+Cxym(nbc)
```

```
      ENDIF

    ENDDO

    cl = 1
    sum = 0.0
    ran = ran2(idum)*REAL(Cxytot,8)

    DO i=1,nbc-1

      sum = sum+REAL(Cxym(i),8)

      IF ((ran.gt.sum).and.(ran.lt.sum+REAL(Cxym(i+1),8))) cl = i+1

    ENDDO

    ind = idclass(cl)
    ok = .false.

    DO WHILE (.not.ok)

      nsite = INT(ran2(idum)*REAL(nb_site_class(ind),8))+1

      IF (enabled(ind,nsite)) THEN

        ok = .true.

        xx1 = class_site(ind,nsite,x)
        yy1 = class_site(ind,nsite,y)
        site1 = lattice(xx1,yy1)
        site2 = class_coord(ind,2)

      ENDIF

    ENDDO

    neigh = INT(ran2(idum)*4)+1
    ok = .false.

    DO WHILE (.not.ok)

      neigh = mod(neigh,4)+1

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

      IF ((.not.outside(xn,yn)).and.
  &       (lattice(xn,yn).eq.site2)) ok = .true.

    ENDDO

    xx2 = xn
    yy2 = yn

    lattice(xx1,yy1) = 0
    lattice(xx2,yy2) = 0

    NBADS(p1) = NBADS(p1)-1
    NBADS(p2) = NBADS(p2)-1
    NBADS(0) = NBADS(0)+2

    DO j=1,p3
      k = param_reac(rnum,3+j)
      NBGAS(k) = NBGAS(k)+1
    ENDDO
```

```
    ELSE IF (type_reac(rnum).eq.6) THEN

      s1 = p1
      s2 = 0

      nbc = 0
      Cxytot = 0

      DO i=1,4

       ind = class(s1,s2,i)

       IF (realnb_site_class(ind).gt.0) THEN
         nbc = nbc+1
         idclass(nbc) = ind
         Cxym(nbc) = i*realnb_site_class(ind)
         Cxytot = Cxytot+Cxym(nbc)
       ENDIF

       ind = class(s2,s1,i)

       IF (realnb_site_class(ind).gt.0) THEN
         nbc = nbc+1
         idclass(nbc) = ind
         Cxym(nbc) = i*realnb_site_class(ind)
         Cxytot = Cxytot+Cxym(nbc)

       ENDIF

      ENDDO

      cl = 1
      sum = 0.0
      ran = ran2(idum)*REAL(Cxytot,8)

      DO i=1,nbc-1

       sum = sum+REAL(Cxym(i),8)

       IF ((ran.gt.sum).and.(ran.lt.sum+REAL(Cxym(i+1),8))) cl = i+1

      ENDDO

      ind = idclass(cl)
      ok = .false.

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

       DO WHILE (.not.ok)

        nsite = INT(ran2(idum)*REAL(nb_site_class(ind),8))+1

        IF (enabled(ind,nsite)) THEN

          ok = .true.

          xx1 = class_site(ind,nsite,x)
          yy1 = class_site(ind,nsite,y)
          site1 = lattice(xx1,yy1)
          site2 = class_coord(ind,2)

        ENDIF

       ENDDO

       neigh = INT(ran2(idum)*4)+1
```

```
      ok = .false.

      DO WHILE (.not.ok)

       neigh = mod(neigh,4)+1

       CALL neighbour(xx1,yy1,xn,yn,neigh)

       IF (.not.outside(xn,yn)) THEN

         IF (lattice(xn,yn).eq.0) ok = .true.

       ELSE

         ok = .true.

         IF (((xbox.eq.1).and.(neigh.eq.3)).or.
     &         ((xbox.eq.teeth(x)).and.(neigh.eq.4))) THEN

           ok = .false.

         ENDIF

         IF (((ybox.eq.1).and.(neigh.eq.2)).or.
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

           ok = .false.

         ENDIF

       ENDIF

      ENDDO

      xx2 = xn
      yy2 = yn

      IF (.not.outside(xx2,yy2)) THEN

       IF (ran2(idum).lt.0.5) THEN
         lattice(xx1,yy1) = p2
         lattice(xx2,yy2) = p3

         NBADS(p1) = NBADS(p1)-1
         NBADS(0) = NBADS(0)-1
         NBADS(p2) = NBADS(p2)+1
         NBADS(p3) = NBADS(p3)+1

       ELSE

         lattice(xx1,yy1) = p3
         lattice(xx2,yy2) = p2

         NBADS(p1) = NBADS(p1)-1
         NBADS(0) = NBADS(0)-1
         NBADS(p2) = NBADS(p2)+1
         NBADS(p3) = NBADS(p3)+1

       ENDIF

      ELSEIF (outside(xx2,yy2)) THEN

       IF (ran2(idum).lt.0.5) THEN

         lattice(xx1,yy1) = p2
```

```
      NBADS(p1) = NBADS(p1)-1
      NBADS(p2) = NBADS(p2)+1
      outgoing(nsim,xbox,ybox,neigh,p3)=outgoing(nsim,xbox,ybox,neigh,p3)+1

     ELSE

      lattice(xx1,yy1) = p3

      NBADS(p1) = NBADS(p1)-1
      NBADS(p3) = NBADS(p3)+1
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

     ENDIF

    ENDIF

   ELSE

    DO WHILE (.not.ok)

     nsite = INT(ran2(idum)*REAL(nb_site_class(ind),8))+1

     IF (enabled(ind,nsite)) THEN

      xx1 = class_site(ind,nsite,x)
      yy1 = class_site(ind,nsite,y)

      IF (.not.outside(xx1,yy1)) THEN

       ok = .true.

       site1 = lattice(xx1,yy1)
       site2 = class_coord(ind,2)

      ENDIF

     ENDIF

    ENDDO

    neigh = INT(ran2(idum)*4)+1
    ok = .false.

    DO WHILE (.not.ok)

     neigh = mod(neigh,4)+1

     CALL neighbour(xx1,yy1,xn,yn,neigh)

     IF ((.not.outside(xn,yn)).and.(lattice(xn,yn).eq.site2)) ok = .true.

    ENDDO

    xx2 = xn
    yy2 = yn

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

     lattice(xx1,yy1) = p2
     lattice(xx2,yy2) = p3

     NBADS(p1) = NBADS(p1)-1
     NBADS(0) = NBADS(0)-1
     NBADS(p2) = NBADS(p2)+1
     NBADS(p3) = NBADS(p3)+1
```

```
      ELSE

        lattice(xx1,yy1) = p3
        lattice(xx2,yy2) = p2

        NBADS(p1) = NBADS(p1)-1
        NBADS(0) = NBADS(0)-1
        NBADS(p2) = NBADS(p2)+1
        NBADS(p3) = NBADS(p3)+1

      ENDIF

    ENDIF

  ELSE IF (type_reac(rnum).eq.7) THEN

    s1 = p1
    s2 = 0

    nbc = 0
    Cxytot = 0

    DO i=1,4

      ind = class(s1,s2,i)

      IF (realnb_site_class(ind).gt.0) THEN
        nbc = nbc+1
        idclass(nbc) = ind
        Cxym(nbc) = i*realnb_site_class(ind)
        Cxytot = Cxytot+Cxym(nbc)
      ENDIF

      ind = class(s2,s1,i)

      IF (realnb_site_class(ind).gt.0) THEN
        nbc = nbc+1
        idclass(nbc) = ind
        Cxym(nbc) = i*realnb_site_class(ind)
        Cxytot = Cxytot+Cxym(nbc)

      ENDIF

    ENDDO

    cl = 1
    sum = 0.0
    ran = ran2(idum)*REAL(Cxytot,8)

    DO i=1,nbc-1

      sum = sum+REAL(Cxym(i),8)

      IF ((ran.gt.sum).and.(ran.lt.sum+REAL(Cxym(i+1),8))) cl = i+1

    ENDDO

    ind = idclass(cl)
    ok = .false.

    IF (class_coord(ind,1).eq.s1) THEN

      DO WHILE (.not.ok)

        nsite = INT(ran2(idum)*REAL(nb_site_class(ind),8))+1
```

```
        IF (enabled(ind,nsite)) THEN

          ok = .true.

          xx1 = class_site(ind,nsite,x)
          yy1 = class_site(ind,nsite,y)
          site1 = lattice(xx1,yy1)
          site2 = class_coord(ind,2)

        ENDIF

      ENDDO

      neigh = INT(ran2(idum)*4)+1
      ok = .false.

      DO WHILE (.not.ok)

        neigh = mod(neigh,4)+1

        CALL neighbour(xx1,yy1,xn,yn,neigh)

        IF (.not.outside(xn,yn)) THEN

          IF (lattice(xn,yn).eq.0) ok = .true.

        ELSE

          ok = .true.

          IF (((xbox.eq.1).and.(neigh.eq.3)).or.
     &         ((xbox.eq.teeth(x)).and.(neigh.eq.4))) THEN

            ok = .false.

          ENDIF

          IF (((ybox.eq.1).and.(neigh.eq.2)).or.
     &         ((ybox.eq.teeth(y)).and.(neigh.eq.1))) THEN

            ok = .false.

          ENDIF

        ENDIF

      ENDDO

      xx2 = xn
      yy2 = yn

      IF (.not.outside(xx2,yy2)) THEN

        lattice(xx1,yy1) = 0
        lattice(xx2,yy2) = p1

      ELSEIF (outside(xx2,yy2)) THEN

        lattice(xx1,yy1) = 0
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
        NBADS(p1) = NBADS(p1)-1
        NBADS(0) = NBADS(0)+1

      ENDIF

      ELSE
```

```
    DO WHILE (.not.ok)

     nsite = INT(ran2(idum)*REAL(nb_site_class(ind),8))+1

     IF (enabled(ind,nsite)) THEN

      xx1 = class_site(ind,nsite,x)
      yy1 = class_site(ind,nsite,y)

     IF (.not.outside(xx1,yy1)) THEN

       ok = .true.

       site1 = lattice(xx1,yy1)
       site2 = class_coord(ind,2)

      ENDIF

     ENDIF

    ENDDO

    neigh = INT(ran2(idum)*4)+1
    ok = .false.

    DO WHILE (.not.ok)

     neigh = mod(neigh,4)+1

     CALL neighbour(xx1,yy1,xn,yn,neigh)

     IF ((.not.outside(xn,yn)).and.(lattice(xn,yn).eq.site2)) ok = .true.

    ENDDO

    xx2 = xn
    yy2 = yn

    lattice(xx1,yy1) = p1
    lattice(xx2,yy2) = 0

   ENDIF

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

   s1 = p1
   s2 = p2

   nbc = 0
   Cxytot = 0

   DO i=1,4

    ind = class(s1,s2,i)

    IF (realnb_site_class(ind).gt.0) THEN
     nbc = nbc+1
     idclass(nbc) = ind
     Cxym(nbc) = i*realnb_site_class(ind)
     Cxytot = Cxytot+Cxym(nbc)
    ENDIF

    ind = class(s2,s1,i)

    IF (realnb_site_class(ind).gt.0) THEN
```

```
        nbc = nbc+1
        idclass(nbc) = ind
        Cxym(nbc) = i*realnb_site_class(ind)
        Cxytot = Cxytot+Cxym(nbc)

       ENDIF

      ENDDO

      cl = 1
      sum = 0.0
      ran = ran2(idum)*REAL(Cxytot,8)

      DO i=1,nbc-1

       sum = sum+REAL(Cxym(i),8)

       IF ((ran.gt.sum).and.(ran.lt.sum+REAL(Cxym(i+1),8))) cl = i+1

      ENDDO

      ind = idclass(cl)
      ok = .false.

      DO WHILE (.not.ok)

       nsite = INT(ran2(idum)*nb_site_class(ind))+1

       IF (enabled(ind,nsite)) THEN

        ok = .true.

        xx1 = class_site(ind,nsite,x)
        yy1 = class_site(ind,nsite,y)
        site1 = lattice(xx1,yy1)
        site2 = class_coord(ind,2)

       ENDIF

      ENDDO

      neigh = INT(ran2(idum)*4)+1
      ok = .false.


xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

       neigh = mod(neigh,4)+1

       CALL neighbour(xx1,yy1,xn,yn,neigh)

       IF ((.not.outside(xn,yn)).and.
     &      (lattice(xn,yn).eq.site2)) ok = .true.

      ENDDO

      xx2 = xn
      yy2 = yn

      IF (ran2(idum).lt.0.5) THEN

       lattice(xx1,yy1) = p3
       lattice(xx2,yy2) = 0

       NBADS(p1) = NBADS(p1)-1
       NBADS(p2) = NBADS(p2)-1
```

```
      NBADS(p3) = NBADS(p3)+1
      NBADS(0) = NBADS(0)+1

    ELSE

      lattice(xx1,yy1) = 0
      lattice(xx2,yy2) = p3

      NBADS(p1) = NBADS(p1)-1
      NBADS(p2) = NBADS(p2)-1
      NBADS(p3) = NBADS(p3)+1
      NBADS(0) = NBADS(0)+1

    ENDIF

  ELSE IF (type_reac(rnum).eq.11) THEN

    s1 = p1
    s2 = p2

    nbc = 0
    Cxytot = 0

    DO i=1,4

      ind = class(s1,s2,i)

      IF (realnb_site_class(ind).gt.0) THEN
        nbc = nbc+1
        idclass(nbc) = ind
        Cxym(nbc) = i*realnb_site_class(ind)
        Cxytot = Cxytot+Cxym(nbc)
      ENDIF

      ind = class(s2,s1,i)

      IF (realnb_site_class(ind).gt.0) THEN
        nbc = nbc+1
        idclass(nbc) = ind
        Cxym(nbc) = i*realnb_site_class(ind)
        Cxytot = Cxytot+Cxym(nbc)

      ENDIF

    ENDDO

    cl = 1
    sum = 0.0
    ran = ran2(idum)*REAL(Cxytot,8)

    DO i=1,nbc-1

      sum = sum+REAL(Cxym(i),8)

      IF ((ran.gt.sum).and.(ran.lt.sum+REAL(Cxym(i+1),8))) cl = i+1

    ENDDO

    ind = idclass(cl)
    ok = .false.

    DO WHILE (.not.ok)

      nsite = INT(ran2(idum)*nb_site_class(ind))+1

      IF (enabled(ind,nsite)) THEN
```

```
          ok = .true.

          xx1 = class_site(ind,nsite,x)
          yy1 = class_site(ind,nsite,y)
          site1 = lattice(xx1,yy1)
          site2 = class_coord(ind,2)

         ENDIF

        ENDDO

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
        ok = .false.

        DO WHILE (.not.ok)

         neigh = mod(neigh,4)+1

         CALL neighbour(xx1,yy1,xn,yn,neigh)

         IF ((.not.outside(xn,yn)).and.
     &       (lattice(xn,yn).eq.site2)) ok = .true.

        ENDDO

        xx2 = xn
        yy2 = yn

        IF (ran2(idum).lt.0.5) THEN

         lattice(xx1,yy1) = param_reac(rnum,3)
         lattice(xx2,yy2) = param_reac(rnum,4)

         NBADS(p1) = NBADS(p1)-1
         NBADS(p2) = NBADS(p2)-1
         NBADS(param_reac(rnum,3)) = NBADS(param_reac(rnum,3))+1
         NBADS(param_reac(rnum,4)) = NBADS(param_reac(rnum,4))+1

        ELSE

         lattice(xx1,yy1) = param_reac(rnum,4)
         lattice(xx2,yy2) = param_reac(rnum,3)

         NBADS(p1) = NBADS(p1)-1
         NBADS(p2) = NBADS(p2)-1
         NBADS(param_reac(rnum,3)) = NBADS(param_reac(rnum,3))+1
         NBADS(param_reac(rnum,4)) = NBADS(param_reac(rnum,4))+1

        ENDIF

       ELSE

        WRITE(*,*)
        WRITE(*,'(''ERROR: '',i2,'' is not 2-site chemical reaction'')')rnum
        WRITE(*,'(''Program terminated.'')')
        WRITE(*,*)
        STOP

       ENDIF

       CALL update_class_onesite(xx1,yy1)
       CALL update_class_onesite(xx2,yy2)

       DO i=top,right
        CALL neighbour(xx1,yy1,xn,yn,i)
```

```
      CALL update_class_twosite(xbox,ybox,xn,yn)
      CALL neighbour(xx2,yy2,xn,yn,i)
      CALL update_class_twosite(xbox,ybox,xn,yn)
    ENDDO

    RETURN
    END
*-------------------------------------------------------------------------*
*****************************************************************************

*****************************************************************************
    SUBROUTINE update_class_onesite (xsite,ysite)
*-------------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: xsite,ysite
    INTEGER :: i,ind,num
    LOGICAL :: outside
*-------------------------------------------------------------------------*
    IF (.not.outside(xsite,ysite)) THEN

      DO i=1,site_clnb(xsite,ysite)
        ind = site_clid(xsite,ysite,i)
        num = site_clsite(xsite,ysite,i)

      IF (enabled(ind,num).and.(class_coord(ind,2).eq.-1)) THEN
        enabled(ind,num) = .false.
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
      ENDIF

      ENDDO

      CALL initclass_onesite(xsite,ysite)

    ENDIF

    RETURN
    END
*-------------------------------------------------------------------------*
*****************************************************************************

*****************************************************************************
    SUBROUTINE update_class_twosite (xbox,ybox,xsite,ysite)
*-------------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: xsite,ysite
    INTEGER :: xbox,ybox
    INTEGER :: i,ind,num
    LOGICAL :: outside
*-------------------------------------------------------------------------*
    IF (.not.outside(xsite,ysite)) THEN

      DO i=1,site_clnb(xsite,ysite)
        ind = site_clid(xsite,ysite,i)
        num = site_clsite(xsite,ysite,i)

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
        enabled(ind,num) = .false.
        realnb_site_class(ind) = realnb_site_class(ind)-1
      ENDIF

      ENDDO
```

```
      CALL initclass_twosite(xbox,ybox,xsite,ysite)

    ENDIF

    RETURN
    END
*-------------------------------------------------------------------------*
*****************************************************************************

*****************************************************************************
    SUBROUTINE read_parameters ()
*-------------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: i,j
    INTEGER :: nametype_ads,nametype_gas
    CHARACTER :: info*51,specname*4
*-------------------------------------------------------------------------*
    INQUIRE(FILE='parameters.dat',EXIST=existe)

    IF (.not.existe) THEN

     WRITE(*,*)
     WRITE(*,'(''*** ERROR: Parameter file "parameters.dat" was not found'')')
     WRITE(*,'(''Program terminated.'')')
     WRITE(*,*)
     STOP

    ENDIF

    OPEN(22,file='parameters.dat',status='unknown')
    REWIND(22)

    READ(22,*)
    READ(22,'(a51)') info
    READ(22,*)

    IF (info(27:29).ne.'1.0') THEN

     WRITE(*,*)
     WRITE(*,'(''*** ERROR: Version incompatibility with control file'')')
     WRITE(*,'(''Please compile the appropriate version or update the "parameters.dat" file'')')
     WRITE(*,'(''Program terminated.'')')
     WRITE(*,*)
     STOP

    ENDIF

    READ(22,*)
    READ(22,*)
    READ(22,*)
    READ(22,'(e20.10)') temperature
    READ(22,*)
    READ(22,*)
    READ(22,*)
    READ(22,'(26x,e20.10)') site_density
    READ(22,*)
    READ(22,*)
    READ(22,'(26x,i2)') nb_gas

    IF (nb_gas.ge.6) THEN

     WRITE(*,*)
     WRITE(*,*)'************************************************************************'
     WRITE(*,*)' WARNING! --> (nb_gas>5) You should change the write_rate subroutine '
```

```
  WRITE(*,*)'    as well as the nb_gas_max parameter defined in variables.inc   '
  WRITE(*,*)'**************************************************************************'
  WRITE(*,'(''Program terminated.'')')
  WRITE(*,*)
  STOP

ENDIF

READ(22,*)

DO i=1,nb_gas

  READ(22,'(6x,a4)') name_gas(i)
  READ(22,'(27x,e20.10)') weight_gas(i)
  READ(22,'(22x,e20.10)') sticking_gas(i)
  READ(22,'(23x,e20.10)') press(i)
  READ(22,'(24x,e20.10)') rate0(i)
  READ(22,*)

ENDDO

READ(22,*)
READ(22,*)
READ(22,'(27x,i2)') nb_ads

IF (nb_ads.ge.6) THEN

  WRITE(*,*)
  WRITE(*,*)'**************************************************************************'
  WRITE(*,*)' WARNING! --> (nb_ads>5) You should change the write_coverage subroutine '
  WRITE(*,*)'      as well as the nb_ads_max parameter defined in variables.inc     '
  WRITE(*,*)'**************************************************************************'
  WRITE(*,'(''Program terminated.'')')
  WRITE(*,*)
  STOP

ENDIF

READ(22,*)

DO i=1,nb_ads

  READ(22,'(6x,a4)') name_ads(i)

ENDDO

READ(22,*)
READ(22,*)
READ(22,*)

READ(22,'(20x,i2)') nb_reac

DO i=1,nb_reac

  READ(22,*)
  READ(22,'(a30)') info

  IF (info(1:23).eq.'Unimolecular adsorption') THEN

    type_reac(i) = 1
    READ(22,'(5x,a4)') specname
    param_reac(i,1) = nametype_gas(specname)
    READ(22,'(10x,a4)') specname
    param_reac(i,2) = nametype_ads(specname)

  ELSE IF (info(1:23).eq.'Unimolecular desorption') THEN
```

```
 type_reac(i) = 2
 READ(22,'(10x,a4)') specname
 param_reac(i,1) = nametype_ads(specname)
 READ(22,'(5x,a4)') specname
 param_reac(i,2) = nametype_gas(specname)
 READ(22,'(23x,e20.10)') preexp(i)
 READ(22,'(19x,e20.10)') actenergy(i)

ELSE IF (info(1:23).eq.'Dissociative adsorption') THEN

 type_reac(i) = 3
 READ(22,'(5x,a4)') specname
 param_reac(i,1) = nametype_gas(specname)
 READ(22,'(10x,a4)') specname
 param_reac(i,2) = nametype_ads(specname)

ELSE IF (info(1:22).eq.'Associative desorption') THEN

 type_reac(i) = 4
 READ(22,'(10x,a4)') specname
 param_reac(i,1) = nametype_ads(specname)
 READ(22,'(5x,a4)') specname
 param_reac(i,2) = nametype_gas(specname)
 READ(22,'(23x,e20.10)') preexp(i)
 READ(22,'(19x,e20.10)') actenergy(i)

ELSE IF (info(1:30).eq.'Bimolecular surface reaction 1') THEN

 type_reac(i) = 5
 READ(22,'(12x,a4)') specname
 param_reac(i,1) = nametype_ads(specname)
 READ(22,'(12x,a4)') specname
 param_reac(i,2) = nametype_ads(specname)
 READ(22,'(18x,i2)') param_reac(i,3)

 IF (param_reac(i,3).ge.3) THEN

  WRITE(*,*)
  WRITE(*,*)'*********************************************************************************'
  WRITE(*,*)'     ERROR! --> You need to increase the number of nb_param_max           '
  WRITE(*,*)'               Currently nb_param_max=5 at (variables.inc)           '
  WRITE(*,*)'*********************************************************************************'
  WRITE(*,'(''Program terminated.'')')
  WRITE(*,*)
  STOP

 ENDIF

 DO j=1,param_reac(i,3)
  READ(22,'(5x,a4)') specname
  param_reac(i,3+j) = nametype_gas(specname)
 ENDDO

 READ(22,'(23x,e20.10)') preexp(i)
 READ(22,'(19x,e20.10)') actenergy(i)

ELSE IF (info(1:30).eq.'Bimolecular surface reaction 2') THEN

 type_reac(i) = 10
 READ(22,'(21x,a4)') specname
 param_reac(i,1) = nametype_ads(specname)
 READ(22,'(21x,a4)') specname
 param_reac(i,2) = nametype_ads(specname)
 READ(22,'(18x,a4)') specname
 param_reac(i,3) = nametype_ads(specname)
```

```
  READ(22,'(23x,e20.10)') preexp(i)
  READ(22,'(19x,e20.10)') actenergy(i)

 ELSE IF (info(1:30).eq.'Bimolecular surface reaction 3') THEN

  type_reac(i) = 11
  READ(22,'(21x,a4)') specname
  param_reac(i,1) = nametype_ads(specname)
  READ(22,'(21x,a4)') specname
  param_reac(i,2) = nametype_ads(specname)
  READ(22,'(20x,a4)') specname
  param_reac(i,3) = nametype_ads(specname)
  READ(22,'(20x,a4)') specname
  param_reac(i,4) = nametype_ads(specname)
  READ(22,'(23x,e20.10)') preexp(i)
  READ(22,'(19x,e20.10)') actenergy(i)

 ELSE IF (info(1:26).eq.'Unimolecular decomposition') THEN

  type_reac(i) = 6
  READ(22,'(19x,a4)') specname
  param_reac(i,1) = nametype_ads(specname)
  READ(22,'(20x,a4)') specname
  param_reac(i,2) = nametype_ads(specname)
  READ(22,'(20x,a4)') specname
  param_reac(i,3) = nametype_ads(specname)
  READ(22,'(23x,e20.10)') preexp(i)
  READ(22,'(19x,e20.10)') actenergy(i)

 ELSE IF (info(1:17).eq.'Surface diffusion') THEN

  type_reac(i) = 7
  READ(22,'(10x,a4)') specname
  param_reac(i,1) = nametype_ads(specname)
  READ(22,'(23x,e20.10)') preexp(i)
  READ(22,'(19x,e20.10)') actenergy(i)

 ELSE IF (info(1:29).eq.'Unimolecular surface reaction') THEN

  type_reac(i) = 8
  READ(22,'(19x,a4)') specname
  param_reac(i,1) = nametype_ads(specname)
  READ(22,'(18x,a4)') specname
  param_reac(i,2) = nametype_ads(specname)
  READ(22,'(23x,e20.10)') preexp(i)
  READ(22,'(19x,e20.10)') actenergy(i)

 ELSE IF (info(1:19).eq.'Reactive adsorption') THEN

  type_reac(i) = 9
  READ(22,'(18x,a4)') specname
  param_reac(i,1) = nametype_gas(specname)
  READ(22,'(19x,a4)') specname
  param_reac(i,2) = nametype_ads(specname)
  READ(22,'(18x,a4)') specname
  param_reac(i,3) = nametype_ads(specname)
  READ(22,'(23x,e20.10)') preexp(i)
  READ(22,'(19x,e20.10)') actenergy(i)

 ELSE

  WRITE(*,*)
  WRITE(*,(''*** ERROR: unknown reaction name in parameter file:'')')
  WRITE(*,'(a30)') info
  WRITE(*,(''Program terminated.'')')
  WRITE(*,*)
```

```
      STOP

     ENDIF

    ENDDO

    CLOSE(22)

    RETURN
    END
*-----------------------------------------------------------------------*
*************************************************************************

*************************************************************************
    SUBROUTINE read_patches ()
*-----------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: i,nx,ny
    CHARACTER :: pr*2,info*46
*-----------------------------------------------------------------------*
    INQUIRE(FILE='patches.dat',EXIST=existe)

    IF (.not.existe) THEN

     WRITE(*,*)
     WRITE(*,'(''*** ERROR: Parameter file "patches.dat" was not found'')')
     WRITE(*,'(''Program terminated.'')')
     WRITE(*,*)
     STOP

    ENDIF

    OPEN(18,FILE='patches.dat',STATUS='unknown')
    REWIND(18)

    READ(18,*)
    READ(18,'(a46)') info
    READ(18,*)

    IF (info(22:24).ne.'1.0') THEN

     WRITE(*,*)
     WRITE(*,'(''*** ERROR: Version incompatibility with control file'')')
     WRITE(*,'(''Please compile the appropriate version or update the "patches.dat" file'')')
     WRITE(*,'(''Program terminated.'')')
     WRITE(*,*)
     STOP

    ENDIF

    READ(18,*)
    READ(18,'(27x,i10)') teeth(x)
    READ(18,'(27x,i10)') teeth(y)
    READ(18,*)
    READ(18,'(28x,i10)') lsize(x)
    READ(18,'(28x,i10)') lsize(y)
    READ(18,*)

    nb_sites = lsize(x)*lsize(y)
    name(0) = '*   '

    READ(18,*)
    READ(18,'(36x,i10)')nb_ads
    READ(18,*)
```

```
    DO i=1,nb_ads

     READ(18,'(a5,a2)')name(i),pr

     IF (pr.eq.'P0') priority(i) = 1
     IF (pr.eq.'P1') priority(i) = 10
     IF (pr.eq.'P2') priority(i) = 1000
     IF (pr.eq.'P3') priority(i) = 100000
     IF (pr.eq.'P4') priority(i) = 10000000

    ENDDO

    DO i=1,5
     READ(18,*)
    ENDDO

    theta0(:,:,:) = 0.0
    theta0(:,:,0) = 1.0

    DO ny = 1,teeth(y)
     DO nx = 1,teeth(x)

       DO i=1,nb_ads

        READ(18,*)theta0(nx,ny,i)

        theta0(nx,ny,0) = theta0(nx,ny,0)-theta0(nx,ny,i)

       ENDDO

      READ(18,*)
      READ(18,*)

     ENDDO
    ENDDO

    CLOSE(18)

    RETURN
    END SUBROUTINE
*-------------------------------------------------------------------------------*
*********************************************************************************

*********************************************************************************
    SUBROUTINE read_fluxes ()
*-------------------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: i,side
    CHARACTER :: info*50, str*6
*-------------------------------------------------------------------------------*
    INQUIRE(FILE='control.dat',EXIST=existe)

    IF (.not.existe) THEN

     WRITE(*,*)
     WRITE(*,'(''*** ERROR: Parameter file "control.dat" was not found'')')
     WRITE(*,'(''Program terminated.'')')
     WRITE(*,*)
     STOP

    ENDIF

    OPEN(24,file='control.dat',status='unknown')
```

```
REWIND(24)

READ(24,*)
READ(24,'(a50)') info
READ(24,*)

IF (info(28:30).ne.'1.0') THEN

 WRITE(*,*)
 WRITE(*,'(''*** ERROR: Version incompatibility with standard input'')')
 WRITE(*,'(''Please compile the appropriate Version of "control.dat" file'')')
 WRITE(*,'(''Program terminated.'')')
 WRITE(*,*)
 STOP

ENDIF

READ(24,*)
READ(24,*)
READ(24,*)
READ(24,*) idum
READ(24,*)
READ(24,*)
READ(24,*)
READ(24,*) nb_simul

IF (nb_simul.gt.simul_max) THEN

 WRITE(*,*)
 WRITE(*,*)'*************************************************************************'
 WRITE(*,*)'     ERROR! --> You need to increase the number of sim_max         '
 WRITE(*,*)'              Currently simul_max=10 at (variables.inc)          '
 WRITE(*,*)'*************************************************************************'
 WRITE(*,'(''Program terminated.'')')
 WRITE(*,*)
 STOP

ENDIF

READ(24,*)
READ(24,*)
READ(24,*)
READ(24,'(e20.10)') begin_time
READ(24,'(e20.10)') time_incr
READ(24,'(e20.10)') end_time
READ(24,*)
READ(24,*)
READ(24,*)
READ(24,'(e20.10)') time_int_fluxes
READ(24,*)
READ(24,*)
READ(24,*)
READ(24,'(e20.10)') time_boundary
READ(24,*)
READ(24,*)

ext_flux_gen0(:,:)=0
ext_flux_ind0(:,:,:,:)=0
ext_flux_ind(:,:,:,:,:)=0

DO side=top,right

 READ(24,*)
 READ(24,*)
 READ(24,*)
 READ(24,*) boundary(side)
```

```
      IF (boundary(side).eq.'NATURAL') THEN

       bound_fluxes = .true.
       READ(24,'(a6)') str

       IF (str.eq.'source') THEN
        source(side) = .true.
       ENDIF

       DO i=1,nb_ads

        READ(24,*) ext_flux_gen0(side,i)

       ENDDO

      ENDIF

     ENDDO

     CLOSE(24)

     time_steps = INT((end_time-begin_time)/time_int_fluxes)

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

     RETURN
     END SUBROUTINE
*-----------------------------------------------------------------------*
***********************************************************************

***********************************************************************
     SUBROUTINE read_lattice (xbox,ybox)
*-----------------------------------------------------------------------*
     IMPLICIT NONE
     INCLUDE 'variables.inc'

     INTEGER :: i,j,n
     INTEGER :: xbox,ybox,stype
     CHARACTER :: conf_file*18,ch
*-----------------------------------------------------------------------*
     conf_file(1:7) = 'config_'

     DO i=2,0,-1
      n = mod(INT(REAL(xbox)/10**i),10)
      ch = char(48+n)
      conf_file(10-i:10-i) = ch
     ENDDO

     conf_file(11:11) = '_'

     DO i=2,0,-1
      n = mod(INT(REAL(ybox)/10**i),10)
      ch = char(48+n)
      conf_file(14-i:14-i) = ch
     ENDDO

     conf_file(15:18) = '.dat'

     INQUIRE(FILE=conf_file,EXIST=existe)

     IF (.not.existe) THEN
      WRITE(*,*)
      WRITE(*,'(''*** ERROR: Initial configuration file (config.dat) not found'')')
      WRITE(*,'(''Program terminated.'')')
      WRITE(*,*)
```

```
  STOP
ENDIF

NBADS(:) = 0
nb_sites = lsize(x)*lsize(y)
NBADS(0) = nb_sites
lattice(:,:) = 0

OPEN(22,file=conf_file,status='unknown')
REWIND(22)

READ(22,'(i6)') lsize(x)
READ(22,'(i6)') lsize(y)

IF (lsize(x).gt.lmaxX) THEN
  WRITE(*,*)
  WRITE(*,'(''*** ERROR: input lattice is too large along X:'')')
  WRITE(*,'('' - lsize[X] = '',i6)') lsize(x)
  WRITE(*,'('' - lmax[X] = '',i4)') lmaxX
  WRITE(*,'(''Re-compile the program with an appropriate value for lsize(x)'')')
  WRITE(*,'('' OR change the maximum lattice size lmaxX'')')
  STOP
ENDIF

IF (lsize(y).gt.lmaxY) THEN
  WRITE(*,*)
  WRITE(*,'(''*** ERROR: input lattice is too large along Y:'')')
  WRITE(*,'('' - lsize[Y] = '',i6)') lsize(y)
  WRITE(*,'('' - lmax[Y] = '',i4)') lmaxY
  WRITE(*,'(''Re-compile the program with an appropriate value for lsize(y)'')')
  WRITE(*,'('' OR change the maximum lattice size lmaxY'')')
  STOP
ENDIF

DO i=1,lsize(x)
  DO j=1,lsize(y)
    READ(22,'(i2)') lattice(i,j)
  ENDDO
ENDDO

CLOSE(22)

DO i=1,lsize(x)
  DO j=1,lsize(y)

    stype = lattice(i,j)

    IF (stype.gt.nb_ads) THEN
      WRITE(*,*)
      WRITE(*,'(''ERROR: too many adsorbed species'')')
      WRITE(*,'(''Initial lattice is not compatible with parameter file.'')')
      WRITE(*,'(''Program terminated.'')')
      WRITE(*,*)
      STOP
    ENDIF

    IF (stype.gt.0) THEN
      NBADS(stype) = NBADS(stype)+1
      NBADS(0) = NBADS(0)-1
    ENDIF

  ENDDO
ENDDO

RETURN
END
```

```
*-----------------------------------------------------------------------*
*************************************************************************

*************************************************************************
      INTEGER FUNCTION nametype_ads(species)
*-----------------------------------------------------------------------*
      IMPLICIT NONE
      INCLUDE 'variables.inc'

      CHARACTER :: species*4
      INTEGER :: i,found
*-----------------------------------------------------------------------*
      found = 0

      DO i=1,nb_ads
        IF (name_ads(i).eq.species) found = i
      ENDDO

      IF (found.eq.0) THEN
        WRITE(*,*)
        WRITE(*,'(''Unspecified adsorbed species found in "control.dat":'',a4)') species
        WRITE(*,'(''Program terminated.'')')
        WRITE(*,*)
        STOP
      ENDIF

      nametype_ads = found

      RETURN
      END
*-----------------------------------------------------------------------*
*************************************************************************

*************************************************************************
      INTEGER FUNCTION nametype_gas(species)
*-----------------------------------------------------------------------*
      IMPLICIT NONE
      INCLUDE 'variables.inc'

      CHARACTER :: species*4
      INTEGER :: i,found
*-----------------------------------------------------------------------*
      found = 0

      DO i=1,nb_gas
        IF (name_gas(i).eq.species) found = i
      ENDDO

      IF (found.eq.0) THEN
        WRITE(*,*)
        WRITE(*,'(''Unspecified gaseous species found in "control.dat":'',a4)') species
        WRITE(*,'(''Program terminated.'')')
        WRITE(*,*)
        STOP
      ENDIF

      nametype_gas = found

      RETURN
      END
*-----------------------------------------------------------------------*
*************************************************************************

*************************************************************************
      SUBROUTINE store_class_coords(ind,site1,site2,numb)
*-----------------------------------------------------------------------*
```

```
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: ind,site1,site2,numb
*-----------------------------------------------------------------------------*
    class_coord(ind,1) = site1
    class_coord(ind,2) = site2
    class_coord(ind,3) = numb

    RETURN
    END
*-----------------------------------------------------------------------------*
*******************************************************************************

*******************************************************************************
*               Selection of random number (0,1)                    *
*******************************************************************************
    REAL FUNCTION ran2(idum)
*-----------------------------------------------------------------------------*
    IMPLICIT NONE
    INTEGER ::idum

    INTEGER,PARAMETER :: IM1=2147483563, IM2=2147483399, IMM1=IM1-1,
   &              IA1=40014, IA2=40692, IQ1=53668, IQ2=52774,
   &              IR1=12211, IR2=3791, NTAB=32, NDIV=1+IMM1/NTAB
    REAL,PARAMETER :: AM=1./IM1, EPS=1.2E-7, RNMX=1.-EPS

    INTEGER :: idum2,j,k,iv(NTAB),iy
    SAVE iv,iy,idum2
    DATA idum2/123456789/, iv/NTAB*0/, iy/0/
*-----------------------------------------------------------------------------*
    IF (idum.le.0) THEN
      idum = max(-idum,1)
      idum2 = idum
      DO j=NTAB+8,1,-1
        k = idum/IQ1
        idum = IA1*(idum-k*IQ1)-k*IR1
        IF (idum.lt.0) idum = idum+IM1
        IF (j.le.NTAB) iv(j) = idum
      ENDDO
      iy = iv(1)
    ENDIF

    k = idum/IQ1
    idum = IA1*(idum-k*IQ1)-k*IR1
    IF (idum.lt.0) idum = idum+IM1

    k = idum2/IQ2
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    IF (idum2.lt.0) idum2 = idum2+IM2

    j = 1+iy/NDIV
    iy = iv(j)-idum2
    iv(j) = idum
    IF (iy.lt.1) iy = iy+IMM1
    ran2 = min(AM*iy,RNMX)

    RETURN
    END FUNCTION
*-----------------------------------------------------------------------------*
*******************************************************************************

*******************************************************************************
    SUBROUTINE write_configuration (xbox,ybox)
*-----------------------------------------------------------------------------*
    IMPLICIT NONE
```

```
      INCLUDE 'variables.inc'

      INTEGER :: xbox,ybox
      INTEGER :: i,j,n
      CHARACTER :: conf_file*18, ch
*-----------------------------------------------------------------------------*
      conf_file(1:7) = 'config_'

      DO i=2,0,-1
        n = mod(INT(REAL(xbox)/10**i),10)
        ch = char(48+n)
        conf_file(10-i:10-i) = ch
      ENDDO

      conf_file(11:11) = '_'

      DO i=2,0,-1
        n = mod(INT(REAL(ybox)/10**i),10)
        ch = char(48+n)
        conf_file(14-i:14-i) = ch
      ENDDO

      conf_file(15:18) = '.dat'

      OPEN(21,FILE=conf_file,STATUS='unknown')
      REWIND(21)

      WRITE(21,'(i6)') lsize(x)
      WRITE(21,'(i6)') lsize(y)

      DO i=1,lsize(x)
        DO j=1,lsize(y)
          WRITE(21,'(i2)') lattice(i,j)
        ENDDO
      ENDDO

      CLOSE(21)

      RETURN
      END SUBROUTINE
*-----------------------------------------------------------------------------*
*******************************************************************************

*******************************************************************************
      SUBROUTINE write_slabs (xbox,ybox)
*-----------------------------------------------------------------------------*
      IMPLICIT NONE
      INCLUDE 'variables.inc'

      INTEGER :: xbox,ybox
      INTEGER :: i,slab
*-----------------------------------------------------------------------------*
      IF (time.lt.end_time) THEN

        slab = ceiling((time-begin_time)/time_incr)

      ELSE

        slab = INT((time-begin_time)/time_incr)

      ENDIF

      DO i=0,nb_ads

        theta(xbox,ybox,i,slab) = REAL(NBADS(i),8)
```

```
    ENDDO

    DO i=1,nb_gas

     rate(xbox,ybox,i,slab) = rate(xbox,ybox,i,slab)+REAL(NBGAS(i),8)
     NBGAS(i) = 0

    ENDDO

    RETURN
    END
*-----------------------------------------------------------------------*
*************************************************************************

*************************************************************************
    SUBROUTINE write_nbreaction ()
*-----------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: i, nx, ny
    INTEGER(KIND=8) :: nbrtot
*-----------------------------------------------------------------------*
    WRITE(*,*)
    WRITE(*,'(''-----     Number of Reaction Events     -----'')')
    WRITE(*,'(''       ----------------------------         '')')

    DO ny = 1,teeth(y)
     DO nx = 1,teeth(x)

      WRITE(*,'('' Tooth [x]:'',i4)')nx
      WRITE(*,'('' Tooth [y]:'',i4)')ny
      WRITE(*,'('' ----------          '')')

      nbrtot = 0

      DO i=1,nb_reac

       WRITE(*,'(''- Reaction of type '',i2,'': '',i13)')i,INT(nb_reaction(nx,ny,i)/nb_simul)
       nbrtot = nbrtot+INT(nb_reaction(nx,ny,i)/nb_simul)

      ENDDO

      WRITE(*,'('' ----------         '')')
      WRITE(*,'(''Total Number of Events: '',i12)')nbrtot
      WRITE(*,'(''-----------------------------------------------'')')

     ENDDO
    ENDDO

    RETURN
    END
*-----------------------------------------------------------------------*
*************************************************************************

*************************************************************************
    SUBROUTINE write_indiv_coverages ()
*-----------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: i,sl,nx,ny,n
    CHARACTER :: cover_file*17,ch
    REAL(KIND=8) :: tt
*-----------------------------------------------------------------------*
    DO ny = 1,teeth(y)
```

```
      DO nx = 1,teeth(x)

       cover_file(1:6) = 'cover_'

       DO i=2,0,-1
        n = mod(INT(REAL(nx)/10**i),10)
        ch = char(48+n)
        cover_file(9-i:9-i) = ch
       ENDDO

       cover_file(10:10) = '_'

       DO i=2,0,-1
        n = mod(INT(REAL(ny)/10**i),10)
        ch = char(48+n)
        cover_file(13-i:13-i) = ch
       ENDDO

       cover_file(14:17) = '.out'

       DO i=0,nb_ads

        theta(nx,ny,i,0) = theta0(nx,ny,i)

        DO sl=1,nb_pts

         theta(nx,ny,i,sl) = theta(nx,ny,i,sl)/REAL(nb_sites,8)

        ENDDO

       ENDDO

       OPEN(22,FILE=cover_file,STATUS='unknown')
       REWIND(22)

       DO sl=0,nb_pts

        tt = begin_time+REAL(sl,8)*time_incr

        IF (nb_ads.eq.1) THEN
         WRITE(22,'(f13.7,2e15.6)') tt,(theta(nx,ny,i,sl),i=0,nb_ads)
        ELSEIF (nb_ads.eq.2) THEN
         WRITE(22,'(f13.7,3e15.6)') tt,(theta(nx,ny,i,sl),i=0,nb_ads)
        ELSEIF (nb_ads.eq.3) THEN
         WRITE(22,'(f13.7,4e15.6)') tt,(theta(nx,ny,i,sl),i=0,nb_ads)
        ELSEIF (nb_ads.eq.4) THEN
         WRITE(22,'(f13.7,5e15.6)') tt,(theta(nx,ny,i,sl),i=0,nb_ads)
        ELSE
         WRITE(22,'(f13.7,6e15.6)') tt,(theta(nx,ny,i,sl),i=0,nb_ads)
        ENDIF

       ENDDO

       CLOSE(22)

      ENDDO
     ENDDO

     RETURN
     END
*-----------------------------------------------------------------------*
*************************************************************************

*************************************************************************
     SUBROUTINE write_indiv_rates ()
*-----------------------------------------------------------------------*
```

```
      IMPLICIT NONE
      INCLUDE 'variables.inc'

      INTEGER :: i,sl,nx,ny,n
      CHARACTER :: rates_file*26,rates_file_more*31,ch
      REAL(KIND=8) :: tt
*-----------------------------------------------------------------------------*
      DO ny = 1,teeth(y)
       DO nx = 1,teeth(x)

        rates_file(1:15) = 'rates_mol.s^-1_'

        DO i=2,0,-1
         n = mod(INT(REAL(nx)/10**i),10)
         ch = char(48+n)
         rates_file(18-i:18-i) = ch
        ENDDO

        rates_file(19:19) = '_'

        DO i=2,0,-1
         n = mod(INT(REAL(ny)/10**i),10)
         ch = char(48+n)
         rates_file(22-i:22-i) = ch
        ENDDO

        rates_file(23:26) = '.out'

        DO i=1,nb_gas

         rate(nx,ny,i,0) = rate0(i)

         DO sl=1,nb_pts

          rate(nx,ny,i,sl) = rate(nx,ny,i,sl)/(N_AVOG*time_incr)/nb_simul

         ENDDO

        ENDDO

        OPEN(23,FILE=rates_file,STATUS='unknown')
        REWIND(23)

        DO sl=0,nb_pts

         tt = begin_time+REAL(sl,8)*time_incr

         IF (nb_gas.eq.1) THEN
          WRITE(23,'(f13.7,2e15.6)') tt,(rate(nx,ny,i,sl),i=1,nb_gas)
         ELSEIF (nb_gas.eq.2) THEN
          WRITE(23,'(f13.7,3e15.6)') tt,(rate(nx,ny,i,sl),i=1,nb_gas)
         ELSEIF (nb_gas.eq.3) THEN
          WRITE(23,'(f13.7,4e15.6)') tt,(rate(nx,ny,i,sl),i=1,nb_gas)
         ELSEIF (nb_gas.eq.4) THEN
          WRITE(23,'(f13.7,5e15.6)') tt,(rate(nx,ny,i,sl),i=1,nb_gas)
         ELSE
          WRITE(23,'(f13.7,6e15.6)') tt,(rate(nx,ny,i,sl),i=1,nb_gas)
         ENDIF

        ENDDO

        CLOSE(23)

        rates_file_more(1:20) = 'rates_mol.s^-1.m^-2_'

        DO i=2,0,-1
```

```
          n = mod(INT(REAL(nx)/10**i),10)
          ch = char(48+n)
          rates_file_more(23-i:23-i) = ch
        ENDDO

        rates_file_more(24:24) = '_'

        DO i=2,0,-1
          n = mod(INT(REAL(ny)/10**i),10)
          ch = char(48+n)
          rates_file_more(27-i:27-i) = ch
        ENDDO

        rates_file_more(28:31) = '.out'

        DO i=1,nb_gas

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

          DO sl=1,nb_pts

                rate_more(nx,ny,i,sl) = rate(nx,ny,i,sl)*site_density/REAL(nb_sites,8)

          ENDDO
        ENDDO

        OPEN(UNIT=25,FILE=rates_file_more,STATUS='unknown')
        REWIND(25)

        DO sl=0,nb_pts

          tt = begin_time + REAL(sl,8)*time_incr

          IF (nb_gas.eq.1) THEN
            WRITE(25,'(f13.7,2e15.6)') tt,(rate_more(nx,ny,i,sl),i=1,nb_gas)
          ELSEIF (nb_gas.eq.2) THEN
            WRITE(25,'(f13.7,3e15.6)') tt,(rate_more(nx,ny,i,sl),i=1,nb_gas)
          ELSEIF (nb_gas.eq.3) THEN
            WRITE(25,'(f13.7,4e15.6)') tt,(rate_more(nx,ny,i,sl),i=1,nb_gas)
          ELSEIF (nb_gas.eq.4) THEN
            WRITE(25,'(f13.7,5e15.6)') tt,(rate_more(nx,ny,i,sl),i=1,nb_gas)
          ELSE
            WRITE(25,'(f13.7,6e15.6)') tt,(rate_more(nx,ny,i,sl),i=1,nb_gas)
          ENDIF

        ENDDO

        CLOSE(25)

      ENDDO
    ENDDO

    RETURN
    END
*---------------------------------------------------------------------------------------*
****************************************************************************************
```

A.3.1.3 Variables

```
********************************************************************************
*----------------------          Parameters          ----------------------*

    INTEGER,PARAMETER :: x=1 , y=2
    INTEGER,PARAMETER :: top=1, bottom=2, left=3, right=4
    INTEGER,PARAMETER :: nb_ads_max=5, nb_gas_max=5
    INTEGER,PARAMETER :: lmaxX=100, lmaxY=100, teeth_max=5
    INTEGER,PARAMETER :: nb_site_max=lmaxX*lmaxY
    INTEGER,PARAMETER :: nb_site_max2=INT(REAL(nb_site_max)*2)
    INTEGER,PARAMETER :: nb_reac_max=11, nb_param_max=5
    INTEGER,PARAMETER :: nb_class_max=44, nbclsite_max=1000
    INTEGER,PARAMETER :: nb_pts_max=10000, simul_max=10
    REAL(KIND=8) :: pi=3.14159265359, R=8.314472, N_Avog=6.0221367e23

*----------------------        GENERAL variables       ----------------------*

    INTEGER :: time_hours,time_minutes,time_seconds,idum,nb_simul
    INTEGER :: time_steps,nb_pts
    REAL(KIND=8) :: cpu_begin,cpu_end,temperature,site_density
    REAL(KIND=8) :: begin_time,time_incr,end_time,time_int_fluxes
    REAL(KIND=8) :: time, time_end

*----------------------        SYSTEM variables        ----------------------*

    LOGICAL :: existe
    INTEGER,DIMENSION(nb_ads_max) :: priority
    INTEGER :: insert_type, nsim
    CHARACTER,DIMENSION(0:nb_ads_max) :: name*4
    LOGICAL :: finish

*----------------------        SPECIES variables       ----------------------*

    INTEGER :: nb_gas,nb_ads
    CHARACTER,DIMENSION(nb_gas_max) :: name_gas*4
    CHARACTER,DIMENSION(0:nb_ads_max) :: name_ads*4
    REAL(KIND=8),DIMENSION(nb_gas_max) :: weight_gas,sticking_gas,press
    INTEGER :: nb_sites

*----------------------        LATTICE variables       ----------------------*

    INTEGER,DIMENSION(2) :: teeth,lsize
    INTEGER :: nb_class,nb_proba
    INTEGER,DIMENSION(nb_class_max,3) :: class_coord
    INTEGER(KIND=1),DIMENSION(0:nb_ads_max,-1:nb_ads_max,-1:4) :: class
    INTEGER,DIMENSION(nb_class_max,2) :: proba_coord
    INTEGER,DIMENSION(lmaxX,lmaxY) :: lattice
    INTEGER(KIND=8),DIMENSION(0:nb_ads_max) :: NB_fin, NB
    INTEGER,DIMENSION(nb_class_max) :: nb_site_class
    INTEGER,DIMENSION(nb_class_max) :: realnb_site_class
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    INTEGER(KIND=1),DIMENSION(lmaxX,lmaxY) :: site_clnb
    INTEGER(KIND=1),DIMENSION(lmaxX,lmaxY,nbclsite_max) :: site_clid
    INTEGER,DIMENSION(lmaxX,lmaxY,nbclsite_max) :: site_clsite
    LOGICAL(KIND=1),DIMENSION(nb_class_max,nb_site_max2) :: enabled
    LOGICAL(KIND=1):: clean, gc

*----------------------       REACTIONS variables      ----------------------*

    INTEGER :: nb_reac
    INTEGER,DIMENSION(nb_reac_max) :: type_reac
    INTEGER,DIMENSION(nb_reac_max,nb_param_max) :: param_reac
    INTEGER(KIND=8),DIMENSION(teeth_max,teeth_max,nb_reac_max) :: nb_reaction
```

```
*---------------------       KINETICS variables        ----------------------*

      REAL(KIND=8),DIMENSION(nb_reac_max) :: preexp,actenergy
      REAL(KIND=8),DIMENSION(nb_reac_max) :: k_reac
      REAL(KIND=8),DIMENSION(0:nb_ads_max,-1:nb_ads_max) :: proba
      REAL(KIND=8),DIMENSION(nb_reac_max) :: gamma

*---------------------        DATAS variables        ----------------------*

      REAL(KIND=8),DIMENSION(nb_gas_max) :: rate0
      REAL(KIND=8),DIMENSION(teeth_max,teeth_max,0:nb_ads_max) :: theta0
      INTEGER,DIMENSION(0:nb_ads_max) :: NBADS
      INTEGER(KIND=8),DIMENSION(nb_gas_max) :: NBGAS
      REAL(KIND=8),DIMENSION(teeth_max,teeth_max,0:nb_ads_max,0:nb_pts_max) :: theta
      REAL(KIND=8),DIMENSION(teeth_max,teeth_max,nb_gas_max,0:nb_pts_max) :: rate
      REAL(KIND=8),DIMENSION(teeth_max,teeth_max,nb_gas_max,0:nb_pts_max) :: rate_more
      INTEGER,DIMENSION(simul_max,teeth_max,teeth_max,4,nb_ads_max) :: ingoing
      INTEGER,DIMENSION(simul_max,teeth_max,teeth_max,4,nb_ads_max) :: outgoing

*---------------------        BOUND variables        ----------------------*

      REAL(KIND=8) :: time_boundary, int_boundary
      REAL(KIND=8),DIMENSION(teeth_max,teeth_max,simul_max) :: btime_next
      REAL(KIND=8),DIMENSION(teeth_max,teeth_max,simul_max) :: time_fluxup_slots
      REAL(KIND=8),DIMENSION(teeth_max,teeth_max,simul_max) :: time_fluxupdate
      INTEGER :: tot_exch_part
      INTEGER,DIMENSION(4) :: tot_exch_side_part
      INTEGER,DIMENSION(4,nb_ads_max) :: ext_flux_gen0
      INTEGER,DIMENSION(teeth_max,teeth_max,4,nb_ads_max) :: ext_flux_ind0
      INTEGER,DIMENSION(teeth_max,teeth_max,4,nb_ads_max,simul_max) :: ext_flux_ind
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
      CHARACTER,DIMENSION(4) :: boundary*8
      LOGICAL,DIMENSION(4) :: source
      LOGICAL :: int_fluxes
      LOGICAL,DIMENSION(4) :: int_side_fluxes
      LOGICAL :: bound_fluxes

*---------------------           COMMON             ----------------------*

      COMMON / GENERAL / cpu_begin,cpu_end,time_hours,time_minutes,time_seconds,
     &            nb_simul,temperature,site_density,time,time_end,begin_time,
     &            end_time,time_incr,nb_pts,time_steps,time_int_fluxes

      COMMON / SYSTEM / insert_type,nsim,priority,idum,name,finish

      COMMON / SPECIES / nb_ads,nb_gas,nb_sites,name_gas,weight_gas,sticking_gas,
     &            press,name_ads

      COMMON / REACTIONS / nb_reaction,nb_reac,type_reac,param_reac

      COMMON / KINETICS / preexp,actenergy,k_reac,proba,gamma

      COMMON / LATTICE / lsize,teeth,lattice,NB_fin,NB,class,nb_proba,nb_class,
     &            class_coord,class_site,nb_site_class,realnb_site_class,
     &            proba_coord,site_clid,site_clsite,enabled,clean,gc,site_clnb

      COMMON / DATAS / theta0,NBADS,rate0,NBGAS,rate,rate_more,theta,outgoing,ingoing

      COMMON / BOUND / time_boundary, int_boundary,btime_next,time_fluxup_slots,
     &           time_fluxupdate,source,boundary,bound_fluxes,ext_flux_ind0,
     &           ext_flux_ind,ext_flux_gen0,nb_left,nb_to_add,
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

*---------------------------------------------------------------------------*
*******************************************************************************
```

A.3.1.4 Input files

```
*****************************************************
***          GAP-TOOTH.EXE version 1.0 - Fluxes input file          ***
*****************************************************

Random seed:
------------
-100000

Number of simulations:
----------------------
10

Initial time, time increment, end time:
---------------------------------------
0.0E+00
1.0E-04
2.0E-02

Time interval for internal flux updates:
----------------------------------------
1.0E-06

Time interval for boundary flux updates:
----------------------------------------
1.0E-04

Boundary conditions, in/outgoing particles:
-------------------------------------------
Top:
-------
NATURAL
source
0
--------------------
Bottom:
-------
NATURAL
source
0
--------------------
Left:
-------
NATURAL
source
50
--------------------
Right:
-------
NATURAL
source
0
-------------------------------------------
*****************************************************
```

```
************************************************
***         GAP-TOOTH.EXE version 1.0 - Parameters file      ***
************************************************

TEMPERATURE [K]:
----------
0.0E+00

SURFACE SPECIFICATIONS:
----------
Site density [sites/m^2]: 0.0E+00

GASEOUS CHEMICAL SPECIES: 1
----------
name: XXX
molecular weight [kg/mol]: 0.0E-00
sticking coefficient: 0.0E+00
Partial Pressure [Pa]: 0.0E+00
Initial rate [mol/sec]: 0.0E+00
----------

ADSORBED CHEMICAL SPECIES: 1
----------
name: *X
----------

CHEMICAL REACTIONS: 1
 1) ----------
Surface diffusion
Adsorbed: *X
Preexponential factor: 1.0E+06
Activation energy: 0.000E+00
----------
************************************************
```

```
*****************************************************
***          PATCHES version 1.0: Tooth CREATION input file          ***
*****************************************************

Number of Teeth along [X]: 11
Number of Teeth along [Y]: 1
---------------------------------
Each Tooth Lattice Size[X]: 100
Each Tooth Lattice Size[Y]: 100
---------------------------------

Number of adsorbed chemical species: 1
---------------------------------------
*X  P0
---------------------------------------

Initial surface coverages, Box(x,y):
---------------------------------------
Box(1,1)
5.000E-04
--------
Box(2,1)
5.000E-04
--------
Box(3,1)
5.000E-04
--------
Box(4,1)
5.000E-04
--------
Box(5,1)
5.000E-04
--------
Box(6,1)
5.000E-04
--------
Box(7,1)
5.000E-04
--------
Box(8,1)
5.000E-04
--------
Box(9,1)
5.000E-04
--------
Box(10,1)
5.000E-04
--------
Box(11,1)
5.000E-04
---------------------------------------
*****************************************************
```

A.3.2 Gap-tooth interpolation techniques

A.3.2.1 Linear (1st order) interpolation subroutine

```
**************************************************************************
    SUBROUTINE calc_int_fluxes ()
*-------------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'
    INTEGER :: i, k, nx, ny
*-------------------------------------------------------------------------*
    DO ny = 1,teeth(y)
     DO nx = 1,teeth(x)
      DO k=1,nb_ads

       IF (nx.eq.1) THEN

              ingoing(:,nx,ny,3,k) = ingoing(nb_simul,nx,ny,3,k) + INT((1-gap_interp(x))*REAL(outgoing(nb_simul,nx,ny,4,k),8))
              ingoing(:,nx,ny,4,k) = ingoing(nb_simul,nx,ny,4,k) + CEILING(gap_interp(x)*REAL(outgoing(nb_simul,nx+1,ny,3,k),8))

       ELSE IF ((nx.gt.1).and.(nx.lt.teeth(x))) THEN

              ingoing(:,nx,ny,3,k) = ingoing(nb_simul,nx,ny,3,k) + CEILING(gap_interp(x)*REAL(outgoing(nb_simul,nx-1,ny,4,k),8))
  &                       + INT((1-gap_interp(x))*REAL(outgoing(nb_simul,nx,ny,4,k),8))
              ingoing(:,nx,ny,4,k) = ingoing(nb_simul,nx,ny,4,k) + CEILING(gap_interp(x)*REAL(outgoing(nb_simul,nx+1,ny,3,k),8))
  &                       + INT((1-gap_interp(x))*REAL(outgoing(nb_simul,nx,ny,3,k),8))

       ELSE

              ingoing(:,nx,ny,3,k) = ingoing(nb_simul,nx,ny,3,k) + CEILING(gap_interp(x)*REAL(outgoing(nb_simul,nx-1,ny,4,k),8))
              ingoing(:,nx,ny,4,k) = ingoing(nb_simul,nx,ny,4,k) + INT((1-gap_interp(x))*REAL(outgoing(nb_simul,nx,ny,3,k),8))

       ENDIF

       IF (ny.eq.1) THEN

              ingoing(:,nx,ny,1,k) = ingoing(nb_simul,nx,ny,1,k) + CEILING(gap_interp(y)*REAL(outgoing(nb_simul,nx,ny+1,2,k),8))
              ingoing(:,nx,ny,2,k) = ingoing(nb_simul,nx,ny,2,k) + INT((1-gap_interp(y))*REAL(outgoing(nb_simul,nx,ny,1,k),8))

       ELSE IF ((ny.gt.1).and.(ny.lt.teeth(y))) THEN

              ingoing(:,nx,ny,1,k) = ingoing(nb_simul,nx,ny,1,k) + CEILING(gap_interp(y)*REAL(outgoing(nb_simul,nx,ny+1,2,k),8))
  &                       + INT((1-gap_interp(y))*REAL(outgoing(nb_simul,nx,ny,2,k),8))
              ingoing(:,nx,ny,2,k) = ingoing(nb_simul,nx,ny,2,k) + CEILING(gap_interp(y)*REAL(outgoing(nb_simul,nx,ny-1,1,k),8))
  &                                   + INT((1-gap_interp(y))*REAL(outgoing(nb_simul,nx,ny,1,k),8))

       ELSE

              ingoing(:,nx,ny,1,k) = ingoing(nb_simul,nx,ny,1,k) + INT((1-gap_interp(y))*REAL(outgoing(nb_simul,nx,ny,2,k),8))
              ingoing(:,nx,ny,2,k) = ingoing(nb_simul,nx,ny,2,k) + CEILING(gap_interp(y)*REAL(outgoing(nb_simul,nx,ny-1,1,k),8))

       ENDIF

      ENDDO
     ENDDO
    ENDDO

    RETURN
    END
*-------------------------------------------------------------------------*
**************************************************************************
```

A.3.2.2 Quadratic (2nd order) interpolation subroutine

```
**************************************************************************
    SUBROUTINE calc_int_fluxes ()
*-------------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'
    INTEGER :: i, k, nx, ny
*-------------------------------------------------------------------------*
    DO ny = 1,teeth(y)
        DO nx = 1,teeth(x)
            DO k=1,nb_ads

                IF (nx.eq.1) THEN

                 ingoing(:,nx,ny,3,k) = ingoing(nb_simul,nx,ny,3,k) + INT((1-gap_interp(x))*REAL(outgoing(nb_simul,nx,ny,4,k),8))
   &                    - INT((gap_interp(x)*(1-gap_interp(x)))/2*REAL(outgoing(nb_simul,nx+1,ny,4,k),8))

                 ingoing(:,nx,ny,4,k) = ingoing(nb_simul,nx,ny,4,k)
   &                    + CEILING((gap_interp(x)*(1+gap_interp(x)))/2*REAL(outgoing(nb_simul,nx+1,ny,3,k),8))

                ELSE IF (nx.eq.2) THEN

                 ingoing(:,nx,ny,3,k) = ingoing(nb_simul,nx,ny,3,k)
   &                    + CEILING(gap_interp(x)*REAL(outgoing(nb_simul,nx-1,ny,4,k),8))
   &                    + INT((1-gap_interp(x)**2)*REAL(outgoing(nb_simul,nx,ny,4,k),8))
   &                    - INT((gap_interp(x)*(1-gap_interp(x)))/2*REAL(outgoing(nb_simul,nx+1,ny,4,k),8))

                 ingoing(:,nx,ny,4,k) = ingoing(nb_simul,nx,ny,4,k)
   &                    + CEILING((gap_interp(x)*(1+gap_interp(x)))/2*REAL(outgoing(nb_simul,nx+1,ny,3,k),8))
   &                    + INT((1-gap_interp(x)**2)*REAL(outgoing(nb_simul,nx,ny,3,k),8))

                ELSE IF (nx.eq.teeth(x)-1) THEN

                 ingoing(:,nx,ny,3,k) = ingoing(nb_simul,nx,ny,3,k)
   &                    + CEILING((gap_interp(x)*(1+gap_interp(x)))/2*REAL(outgoing(nb_simul,nx-1,ny,4,k),8))
   &                    + INT((1-gap_interp(x)**2)*REAL(outgoing(nb_simul,nx,ny,4,k),8))

                 ingoing(:,nx,ny,4,k) = ingoing(nb_simul,nx,ny,4,k)
   &                    + CEILING(gap_interp(x)*REAL(outgoing(nb_simul,nx+1,ny,3,k),8))
   &                    + INT((1-gap_interp(x)**2)*REAL(outgoing(nb_simul,nx,ny,3,k),8))
   &                    - INT((gap_interp(x)*(1-gap_interp(x)))/2*REAL(outgoing(nb_simul,nx-1,ny,3,k),8))

                ELSE IF (nx.eq.teeth(x)) THEN

                 ingoing(:,nx,ny,3,k) = ingoing(nb_simul,nx,ny,3,k)
   &                    + CEILING((gap_interp(x)*(1+gap_interp(x)))/2*REAL(outgoing(nb_simul,nx-1,ny,4,k),8))

                 ingoing(:,nx,ny,4,k) = ingoing(nb_simul,nx,ny,4,k) + INT((1-gap_interp(x))*REAL(outgoing(nb_simul,nx,ny,3,k),8))
   &                    - INT((gap_interp(x)*(1-gap_interp(x)))/2*REAL(outgoing(nb_simul,nx-1,ny,3,k),8))

                ELSE

                 ingoing(:,nx,ny,3,k) = ingoing(nb_simul,nx,ny,3,k)
   &                    + CEILING((gap_interp(x)*(1+gap_interp(x)))/2*REAL(outgoing(nb_simul,nx-1,ny,4,k),8))
   &                    + INT((1-gap_interp(x)**2)*REAL(outgoing(nb_simul,nx,ny,4,k),8))
   &                    - INT((gap_interp(x)*(1-gap_interp(x)))/2*REAL(outgoing(nb_simul,nx+1,ny,4,k),8))

                 ingoing(:,nx,ny,4,k) = ingoing(nb_simul,nx,ny,4,k)
   &                    + CEILING((gap_interp(x)*(1+gap_interp(x)))/2*REAL(outgoing(nb_simul,nx+1,ny,3,k),8))
   &                    + INT((1-gap_interp(x)**2)*REAL(outgoing(nb_simul,nx,ny,3,k),8))
   &                    - INT((gap_interp(x)*(1-gap_interp(x)))/2*REAL(outgoing(nb_simul,nx-1,ny,3,k),8))

                ENDIF
```

```
            IF (ny.eq.1) THEN

             ingoing(:,nx,ny,1,k) = ingoing(nb_simul,nx,ny,1,k)
     &              + CEILING((gap_interp(y)*(1+gap_interp(y)))/2*REAL(outgoing(nb_simul,nx,ny+1,2,k),8))

             ingoing(:,nx,ny,2,k) = ingoing(nb_simul,nx,ny,2,k) + INT((1-gap_interp(y))*REAL(outgoing(nb_simul,nx,ny,1,k),8))
     &              - INT((gap_interp(y)*(1-gap_interp(y)))/2*REAL(outgoing(nb_simul,nx,ny+1,1,k),8))

            ELSE IF (ny.eq.2) THEN

             ingoing(:,nx,ny,1,k) = ingoing(nb_simul,nx,ny,1,k)
     &              + CEILING((gap_interp(y)*(1+gap_interp(y)))/2*REAL(outgoing(nb_simul,nx,ny+1,2,k),8))
     &              + INT((1-gap_interp(y)**2)*REAL(outgoing(nb_simul,nx,ny,2,k),8))

             ingoing(:,nx,ny,2,k) = ingoing(nb_simul,nx,ny,2,k) + CEILING(gap_interp(y)*REAL(outgoing(nb_simul,nx,ny-1,1,k),8))
     &              + INT((1-gap_interp(y)**2)*REAL(outgoing(nb_simul,nx,ny,1,k),8))
     &              - INT((gap_interp(y)*(1-gap_interp(y)))/2*REAL(outgoing(nb_simul,nx,ny+1,1,k),8))

            ELSE IF (ny.eq.teeth(y)-1) THEN

             ingoing(:,nx,ny,1,k) = ingoing(nb_simul,nx,ny,1,k)
     &              + CEILING(gap_interp(y)*REAL(outgoing(nb_simul,nx,ny+1,2,k),8))
     &              + INT((1-gap_interp(y)**2)*REAL(outgoing(nb_simul,nx,ny,2,k),8))
     &              - INT((gap_interp(y)*(1-gap_interp(y)))/2*REAL(outgoing(nb_simul,nx,ny-1,2,k),8))

             ingoing(:,nx,ny,2,k) = ingoing(nb_simul,nx,ny,2,k)
     &              + CEILING((gap_interp(y)*(1+gap_interp(y)))/2*REAL(outgoing(nb_simul,nx,ny-1,1,k),8))
     &              + INT((1-gap_interp(y)**2)*REAL(outgoing(nb_simul,nx,ny,1,k),8))

            ELSE IF (ny.eq.teeth(y)) THEN

             ingoing(:,nx,ny,1,k) = ingoing(nb_simul,nx,ny,1,k) + INT((1-gap_interp(y))*REAL(outgoing(nb_simul,nx,ny,2,k),8))
     &              - INT((gap_interp(y)*(1-gap_interp(y)))/2*REAL(outgoing(nb_simul,nx,ny-1,2,k),8))

             ingoing(:,nx,ny,2,k) = ingoing(nb_simul,nx,ny,2,k)
     &              + CEILING((gap_interp(y)*(1+gap_interp(y)))/2*REAL(outgoing(nb_simul,nx,ny-1,1,k),8))

            ELSE

             ingoing(:,nx,ny,1,k) = ingoing(nb_simul,nx,ny,1,k)
     &              + CEILING((gap_interp(y)*(1+gap_interp(y)))/2*REAL(outgoing(nb_simul,nx,ny+1,2,k),8))
     &              + INT((1-gap_interp(y)**2)*REAL(outgoing(nb_simul,nx,ny,2,k),8))
     &              - INT((gap_interp(y)*(1-gap_interp(y)))/2*REAL(outgoing(nb_simul,nx,ny-1,2,k),8))

             ingoing(:,nx,ny,2,k) = ingoing(nb_simul,nx,ny,2,k)
     &              + CEILING((gap_interp(y)*(1+gap_interp(y)))/2*REAL(outgoing(nb_simul,nx,ny-1,1,k),8))
     &              + INT((1-gap_interp(y)**2)*REAL(outgoing(nb_simul,nx,ny,1,k),8))
     &              - INT((gap_interp(y)*(1-gap_interp(y)))/2*REAL(outgoing(nb_simul,nx,ny+1,1,k),8))

            ENDIF

          ENDDO
        ENDDO
      ENDDO

      RETURN
      END
*-----------------------------------------------------------------------*
*************************************************************************
```

A.3.3 Ingoing species distribution techniques

A.3.3.1 Zone (1-5) distribution around the boundaries

```
*****************************************************************************
      SUBROUTINE make_int_fluxes (nx,ny)
*-----------------------------------------------------------------------*
      IMPLICIT NONE
      INCLUDE 'variables.inc'

      INTEGER :: i,j,k,side,rsitex,rsitey,xn,yn,nx,ny
      INTEGER,DIMENSION(0:nb_ads_max) :: int_nbB
      REAL(KIND=8) :: ran2
      LOGICAL :: ok
*-----------------------------------------------------------------------*
      DO side=top,right

       int_nbB(:) = 0

       IF ((side.eq.top).and.(int_side_fluxes(side))) THEN

        DO j=lsize(y)-4,lsize(y)
         DO i=1,lsize(x)
          int_nbB(lattice(i,j)) = int_nbB(lattice(i,j))+1
         ENDDO
        ENDDO

        DO i=1,nb_ads

         IF (ingoing(nsim,nx,ny,side,i).ne.0) THEN

          IF (int_nbB(0).gt.0) THEN

           ok = .false.

           DO WHILE (.not.ok)

            rsitex = INT(ran2(idum)*REAL(lsize(x),8))+1
            rsitey = INT(ran2(idum)*REAL(5,8))+(lsize(y)-4)

            IF (lattice(rsitex,rsitey).eq.0) THEN

             lattice(rsitex,rsitey) = i
             NBADS(i) = NBADS(i)+1
             NBADS(0) = NBADS(0)-1
             int_nbB(0) = int_nbB(0)-1
             ingoing(nsim,nx,ny,side,i) = ingoing(nsim,nx,ny,side,i)-1

             IF ((int_nbB(0).eq.0).or.
     &                 (ingoing(nsim,nx,ny,side,i).eq.0)) ok = .true.

             CALL update_class_onesite(rsitex,rsitey)
             CALL update_class_twosite(nx,ny,rsitex,rsitey)

             DO k=top,right
              CALL neighbour(rsitex,rsitey,xn,yn,k)
              CALL update_class_twosite(nx,ny,xn,yn)
             ENDDO

            ENDIF
```

```
          ENDDO

         ENDIF

        ENDIF

       ENDDO

      ELSE IF ((side.eq.bottom).and.(int_side_fluxes(side))) THEN

       DO j=1,5
        DO i=1,lsize(x)
         int_nbB(lattice(i,j)) = int_nbB(lattice(i,j))+1
        ENDDO
       ENDDO

       DO i=1,nb_ads

        IF (ingoing(nsim,nx,ny,side,i).ne.0) THEN

         IF (int_nbB(0).gt.0) THEN

          ok = .false.

          DO WHILE (.not.ok)

           rsitex = INT(ran2(idum)*REAL(lsize(x),8))+1
           rsitey = INT(ran2(idum)*REAL(5,8))+1

           IF (lattice(rsitex,rsitey).eq.0) THEN

            lattice(rsitex,rsitey) = i
            NBADS(i) = NBADS(i)+1
            NBADS(0) = NBADS(0)-1
            int_nbB(0) = int_nbB(0)-1
            ingoing(nsim,nx,ny,side,i) = ingoing(nsim,nx,ny,side,i)-1

            IF ((int_nbB(0).eq.0).or.
     &                  (ingoing(nsim,nx,ny,side,i).eq.0)) ok = .true.

            CALL update_class_onesite(rsitex,rsitey)
            CALL update_class_twosite(nx,ny,rsitex,rsitey)

            DO k=top,right
             CALL neighbour(rsitex,rsitey,xn,yn,k)
             CALL update_class_twosite(nx,ny,xn,yn)
            ENDDO

           ENDIF

          ENDDO

         ENDIF

        ENDIF

       ENDDO

      ELSE IF ((side.eq.left).and.(int_side_fluxes(side))) THEN

       DO j=1,lsize(y)
        DO i=1,5
         int_nbB(lattice(i,j)) = int_nbB(lattice(i,j))+1
        ENDDO
       ENDDO
```

```
      DO i=1,nb_ads

       IF (ingoing(nsim,nx,ny,side,i).ne.0) THEN

        IF (int_nbB(0).gt.0) THEN

         ok = .false.

         DO WHILE (.not.ok)

          rsitex = INT(ran2(idum)*REAL(5,8))+1
          rsitey = INT(ran2(idum)*REAL(lsize(y),8))+1

          IF (lattice(rsitex,rsitey).eq.0) THEN

           lattice(rsitex,rsitey) = i
           NBADS(i) = NBADS(i)+1
           NBADS(0) = NBADS(0)-1
           int_nbB(0) = int_nbB(0)-1
            ingoing(nsim,nx,ny,side,i) = ingoing(nsim,nx,ny,side,i)-1

           IF ((int_nbB(0).eq.0).or.
&                    (ingoing(nsim,nx,ny,side,i).eq.0)) ok = .true.

           CALL update_class_onesite(rsitex,rsitey)
           CALL update_class_twosite(nx,ny,rsitex,rsitey)

           DO k=top,right
            CALL neighbour(rsitex,rsitey,xn,yn,k)
            CALL update_class_twosite(nx,ny,xn,yn)
           ENDDO

          ENDIF

         ENDDO

        ENDIF

       ENDIF

      ENDDO

     ELSE IF ((side.eq.right).and.(int_side_fluxes(side))) THEN

      DO j=1,lsize(y)
       DO i= lsize(x)-4,lsize(x)
        int_nbB(lattice(i,j)) = int_nbB(lattice(i,j))+1
       ENDDO
      ENDDO

      DO i=1,nb_ads

       IF (ingoing(nsim,nx,ny,side,i).ne.0) THEN

        IF (int_nbB(0).gt.0) THEN

         ok = .false.

         DO WHILE (.not.ok)

          rsitex = INT(ran2(idum)*REAL(5,8))+(lsize(x)-5)
          rsitey = INT(ran2(idum)*REAL(lsize(y),8))+1

          IF (lattice(rsitex,rsitey).eq.0) THEN

           lattice(rsitex,rsitey) = i
```

```
            NBADS(i) = NBADS(i)+1
            NBADS(0) = NBADS(0)-1
            int_nbB(0) = int_nbB(0)-1
            ingoing(nsim,nx,ny,side,i) = ingoing(nsim,nx,ny,side,i)-1

            IF ((int_nbB(0).eq.0).or.
     &                (ingoing(nsim,nx,ny,side,i).eq.0)) ok = .true.

            CALL update_class_onesite(rsitex,rsitey)
            CALL update_class_twosite(nx,ny,rsitex,rsitey)

            DO k=top,right
              CALL neighbour(rsitex,rsitey,xn,yn,k)
              CALL update_class_twosite(nx,ny,xn,yn)
            ENDDO

           ENDIF

          ENDDO

         ENDIF

        ENDIF

       ENDDO

      ENDIF

    ENDDO

    RETURN
    END
*-----------------------------------------------------------------------*
*************************************************************************
```

A.3.3.2 Zone (1-10) distribution around the boundaries

```
*************************************************************************
    SUBROUTINE make_int_fluxes (nx,ny)
*-----------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: i,j,k,side,rsitex,rsitey,xn,yn,nx,ny
    INTEGER,DIMENSION(0:nb_ads_max) :: int_nbB
    REAL(KIND=8) :: ran2
    LOGICAL :: ok
*-----------------------------------------------------------------------*
    DO side=top,right

      int_nbB(:) = 0

      IF ((side.eq.top).and.(int_side_fluxes(side))) THEN

        DO j=lsize(y)-9,lsize(y)
          DO i=1,lsize(x)
            int_nbB(lattice(i,j)) = int_nbB(lattice(i,j))+1
          ENDDO
        ENDDO

        DO i=1,nb_ads
```

```
      IF (ingoing(nsim,nx,ny,side,i).ne.0) THEN

       IF (int_nbB(0).gt.0) THEN

        ok = .false.

        DO WHILE (.not.ok)

         rsitex = INT(ran2(idum)*REAL(lsize(x),8))+1
         rsitey = INT(ran2(idum)*REAL(10,8))+(lsize(y)-9)

         IF (lattice(rsitex,rsitey).eq.0) THEN

          lattice(rsitex,rsitey) = i
          NBADS(i) = NBADS(i)+1
          NBADS(0) = NBADS(0)-1
          int_nbB(0) = int_nbB(0)-1
          ingoing(nsim,nx,ny,side,i) = ingoing(nsim,nx,ny,side,i)-1

          IF ((int_nbB(0).eq.0).or.
&                  (ingoing(nsim,nx,ny,side,i).eq.0)) ok = .true.

          CALL update_class_onesite(rsitex,rsitey)
          CALL update_class_twosite(nx,ny,rsitex,rsitey)

          DO k=top,right
           CALL neighbour(rsitex,rsitey,xn,yn,k)
           CALL update_class_twosite(nx,ny,xn,yn)
          ENDDO

         ENDIF

        ENDDO

       ENDIF

      ENDIF

     ENDDO

    ELSE IF ((side.eq.bottom).and.(int_side_fluxes(side))) THEN

     DO j=1,10
      DO i=1,lsize(x)
       int_nbB(lattice(i,j)) = int_nbB(lattice(i,j))+1
      ENDDO
     ENDDO

     DO i=1,nb_ads

      IF (ingoing(nsim,nx,ny,side,i).ne.0) THEN

       IF (int_nbB(0).gt.0) THEN

        ok = .false.

        DO WHILE (.not.ok)

         rsitex = INT(ran2(idum)*REAL(lsize(x),8))+1
         rsitey = INT(ran2(idum)*REAL(10,8))+1

         IF (lattice(rsitex,rsitey).eq.0) THEN

          lattice(rsitex,rsitey) = i
          NBADS(i) = NBADS(i)+1
          NBADS(0) = NBADS(0)-1
```

```
            int_nbB(0) = int_nbB(0)-1
            ingoing(nsim,nx,ny,side,i) = ingoing(nsim,nx,ny,side,i)-1

            IF ((int_nbB(0).eq.0).or.
&                   (ingoing(nsim,nx,ny,side,i).eq.0)) ok = .true.

            CALL update_class_onesite(rsitex,rsitey)
            CALL update_class_twosite(nx,ny,rsitex,rsitey)

            DO k=top,right
              CALL neighbour(rsitex,rsitey,xn,yn,k)
              CALL update_class_twosite(nx,ny,xn,yn)
            ENDDO

          ENDIF

        ENDDO

      ENDIF

    ENDIF

  ENDDO

 ELSE IF ((side.eq.left).and.(int_side_fluxes(side))) THEN

  DO j=1,lsize(y)
    DO i=1,10
      int_nbB(lattice(i,j)) = int_nbB(lattice(i,j))+1
    ENDDO
  ENDDO

  DO i=1,nb_ads

    IF (ingoing(nsim,nx,ny,side,i).ne.0) THEN

      IF (int_nbB(0).gt.0) THEN

        ok = .false.

        DO WHILE (.not.ok)

          rsitex = INT(ran2(idum)*REAL(10,8))+1
          rsitey = INT(ran2(idum)*REAL(lsize(y),8))+1

          IF (lattice(rsitex,rsitey).eq.0) THEN

            lattice(rsitex,rsitey) = i
            NBADS(i) = NBADS(i)+1
            NBADS(0) = NBADS(0)-1
            int_nbB(0) = int_nbB(0)-1
             ingoing(nsim,nx,ny,side,i) = ingoing(nsim,nx,ny,side,i)-1

            IF ((int_nbB(0).eq.0).or.
&                   (ingoing(nsim,nx,ny,side,i).eq.0)) ok = .true.

            CALL update_class_onesite(rsitex,rsitey)
            CALL update_class_twosite(nx,ny,rsitex,rsitey)

            DO k=top,right
              CALL neighbour(rsitex,rsitey,xn,yn,k)
              CALL update_class_twosite(nx,ny,xn,yn)
            ENDDO

          ENDIF
```

```
          ENDDO

         ENDIF

        ENDIF

       ENDDO

      ELSE IF ((side.eq.right).and.(int_side_fluxes(side))) THEN

       DO j=1,lsize(y)
        DO i= lsize(x)-9,lsize(x)
         int_nbB(lattice(i,j)) = int_nbB(lattice(i,j))+1
        ENDDO
       ENDDO

       DO i=1,nb_ads

        IF (ingoing(nsim,nx,ny,side,i).ne.0) THEN

         IF (int_nbB(0).gt.0) THEN

          ok = .false.

          DO WHILE (.not.ok)

           rsitex = INT(ran2(idum)*REAL(10,8))+(lsize(x)-9)
           rsitey = INT(ran2(idum)*REAL(lsize(y),8))+1

           IF (lattice(rsitex,rsitey).eq.0) THEN

            lattice(rsitex,rsitey) = i
            NBADS(i) = NBADS(i)+1
            NBADS(0) = NBADS(0)-1
            int_nbB(0) = int_nbB(0)-1
            ingoing(nsim,nx,ny,side,i) = ingoing(nsim,nx,ny,side,i)-1

            IF ((int_nbB(0).eq.0).or.
     &                  (ingoing(nsim,nx,ny,side,i).eq.0)) ok = .true.

            CALL update_class_onesite(rsitex,rsitey)
            CALL update_class_twosite(nx,ny,rsitex,rsitey)

            DO k=top,right
             CALL neighbour(rsitex,rsitey,xn,yn,k)
             CALL update_class_twosite(nx,ny,xn,yn)
            ENDDO

           ENDIF

          ENDDO

         ENDIF

        ENDIF

       ENDDO

      ENDIF

     ENDDO

     RETURN
     END
*-----------------------------------------------------------------------*
*************************************************************************
```

A.3.3.3 Random distribution

```
*************************************************************************
    SUBROUTINE make_int_fluxes (nx,ny)
*-----------------------------------------------------------------------*
    IMPLICIT NONE
    INCLUDE 'variables.inc'

    INTEGER :: i,j,k,side,rsitex,rsitey,xn,yn,nx,ny
    INTEGER,DIMENSION(0:nb_ads_max) :: int_nbB
    REAL(KIND=8) :: ran2
    LOGICAL :: ok
*-----------------------------------------------------------------------*
    DO side=top,right

     int_nbB(:) = 0

     IF ((side.eq.top).and.(int_side_fluxes(side))) THEN

      DO j=1,lsize(y)
       DO i=1,lsize(x)
        int_nbB(lattice(i,j)) = int_nbB(lattice(i,j))+1
       ENDDO
      ENDDO

      DO i=1,nb_ads

       IF (ingoing(nsim,nx,ny,side,i).ne.0) THEN

        IF (int_nbB(0).gt.0) THEN

         ok = .false.

         DO WHILE (.not.ok)

          rsitex = INT(ran2(idum)*REAL(lsize(x),8))+1
          rsitey = INT(ran2(idum)*REAL(lsize(y),8))+1

          IF (lattice(rsitex,rsitey).eq.0) THEN

           lattice(rsitex,rsitey) = i
           NBADS(i) = NBADS(i)+1
           NBADS(0) = NBADS(0)-1
           int_nbB(0) = int_nbB(0)-1
           ingoing(nsim,nx,ny,side,i) = ingoing(nsim,nx,ny,side,i)-1

           IF ((int_nbB(0).eq.0).or.
&                    (ingoing(nsim,nx,ny,side,i).eq.0)) ok = .true.

           CALL update_class_onesite(rsitex,rsitey)
           CALL update_class_twosite(nx,ny,rsitex,rsitey)

           DO k=top,right
            CALL neighbour(rsitex,rsitey,xn,yn,k)
            CALL update_class_twosite(nx,ny,xn,yn)
           ENDDO

          ENDIF

         ENDDO

        ENDIF

       ENDIF
```

```
      ENDDO

     ELSE IF ((side.eq.bottom).and.(int_side_fluxes(side))) THEN

      DO j=1,lsize(y)
       DO i=1,lsize(x)
        int_nbB(lattice(i,j)) = int_nbB(lattice(i,j))+1
       ENDDO
      ENDDO

      DO i=1,nb_ads

       IF (ingoing(nsim,nx,ny,side,i).ne.0) THEN

        IF (int_nbB(0).gt.0) THEN

         ok = .false.

         DO WHILE (.not.ok)

          rsitex = INT(ran2(idum)*REAL(lsize(x),8))+1
          rsitey = INT(ran2(idum)*REAL(lsize(y),8))+1

          IF (lattice(rsitex,rsitey).eq.0) THEN

           lattice(rsitex,rsitey) = i
           NBADS(i) = NBADS(i)+1
           NBADS(0) = NBADS(0)-1
           int_nbB(0) = int_nbB(0)-1
           ingoing(nsim,nx,ny,side,i) = ingoing(nsim,nx,ny,side,i)-1

           IF ((int_nbB(0).eq.0).or.
&                (ingoing(nsim,nx,ny,side,i).eq.0)) ok = .true.

           CALL update_class_onesite(rsitex,rsitey)
           CALL update_class_twosite(nx,ny,rsitex,rsitey)

           DO k=top,right
            CALL neighbour(rsitex,rsitey,xn,yn,k)
            CALL update_class_twosite(nx,ny,xn,yn)
           ENDDO

          ENDIF

         ENDDO

        ENDIF

       ENDIF

      ENDDO

     ELSE IF ((side.eq.left).and.(int_side_fluxes(side))) THEN

      DO j=1,lsize(y)
       DO i=1,lsize(x)
        int_nbB(lattice(i,j)) = int_nbB(lattice(i,j))+1
       ENDDO
      ENDDO

      DO i=1,nb_ads

       IF (ingoing(nsim,nx,ny,side,i).ne.0) THEN

        IF (int_nbB(0).gt.0) THEN
```

```
          ok = .false.

          DO WHILE (.not.ok)

           rsitex = INT(ran2(idum)*REAL(lsize(x),8))+1
           rsitey = INT(ran2(idum)*REAL(lsize(y),8))+1

           IF (lattice(rsitex,rsitey).eq.0) THEN

            lattice(rsitex,rsitey) = i
            NBADS(i) = NBADS(i)+1
            NBADS(0) = NBADS(0)-1
            int_nbB(0) = int_nbB(0)-1
             ingoing(nsim,nx,ny,side,i) = ingoing(nsim,nx,ny,side,i)-1

            IF ((int_nbB(0).eq.0).or.
&                    (ingoing(nsim,nx,ny,side,i).eq.0)) ok = .true.

            CALL update_class_onesite(rsitex,rsitey)
            CALL update_class_twosite(nx,ny,rsitex,rsitey)

            DO k=top,right
             CALL neighbour(rsitex,rsitey,xn,yn,k)
             CALL update_class_twosite(nx,ny,xn,yn)
            ENDDO

           ENDIF

          ENDDO

         ENDIF

        ENDIF

       ENDDO

      ELSE IF ((side.eq.right).and.(int_side_fluxes(side))) THEN

       DO j=1,lsize(y)
        DO i=1,lsize(x)
         int_nbB(lattice(i,j)) = int_nbB(lattice(i,j))+1
        ENDDO
       ENDDO

       DO i=1,nb_ads

        IF (ingoing(nsim,nx,ny,side,i).ne.0) THEN

         IF (int_nbB(0).gt.0) THEN

          ok = .false.

          DO WHILE (.not.ok)

           rsitex = INT(ran2(idum)*REAL(lsize(x),8))+1
           rsitey = INT(ran2(idum)*REAL(lsize(y),8))+1

           IF (lattice(rsitex,rsitey).eq.0) THEN

            lattice(rsitex,rsitey) = i
            NBADS(i) = NBADS(i)+1
            NBADS(0) = NBADS(0)-1
            int_nbB(0) = int_nbB(0)-1
            ingoing(nsim,nx,ny,side,i) = ingoing(nsim,nx,ny,side,i)-1
```

```
            IF ((int_nbB(0).eq.0).or.
   &                (ingoing(nsim,nx,ny,side,i).eq.0)) ok = .true.

            CALL update_class_onesite(rsitex,rsitey)
            CALL update_class_twosite(nx,ny,rsitex,rsitey)

            DO k=top,right
             CALL neighbour(rsitex,rsitey,xn,yn,k)
             CALL update_class_twosite(nx,ny,xn,yn)
            ENDDO

           ENDIF

          ENDDO

         ENDIF

        ENDIF

       ENDDO

      ENDIF

    ENDDO

    RETURN
    END
*-------------------------------------------------------------------------------*
*********************************************************************************
```

A.3.4 The diffusion example using 2nd order interpolation

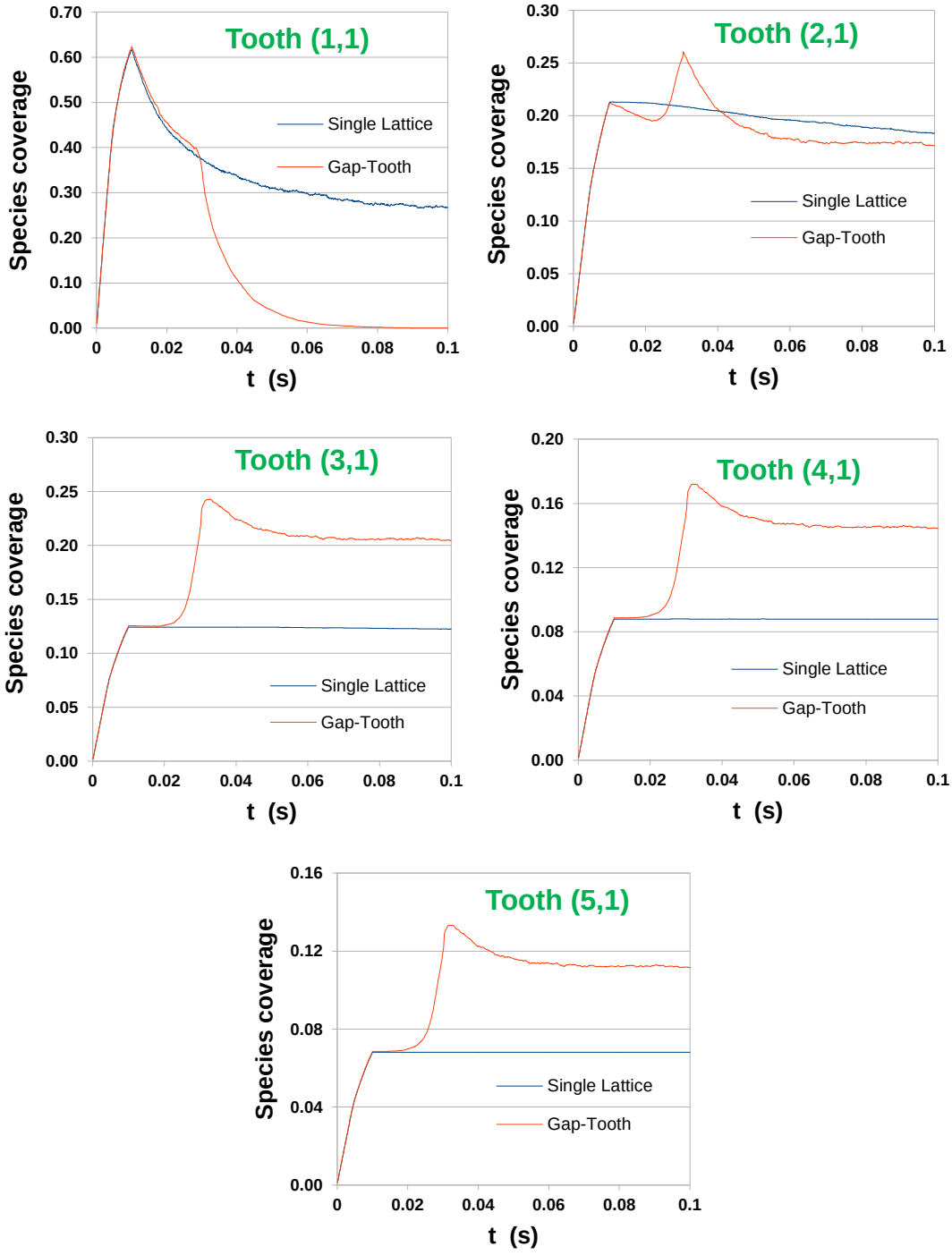


Figure A.1: Diffusing species cumulative coverage profiles comparison between a single lattice (blue lines) and a gap-tooth (red lines) framework, using quadratic (2nd order) interpolation.

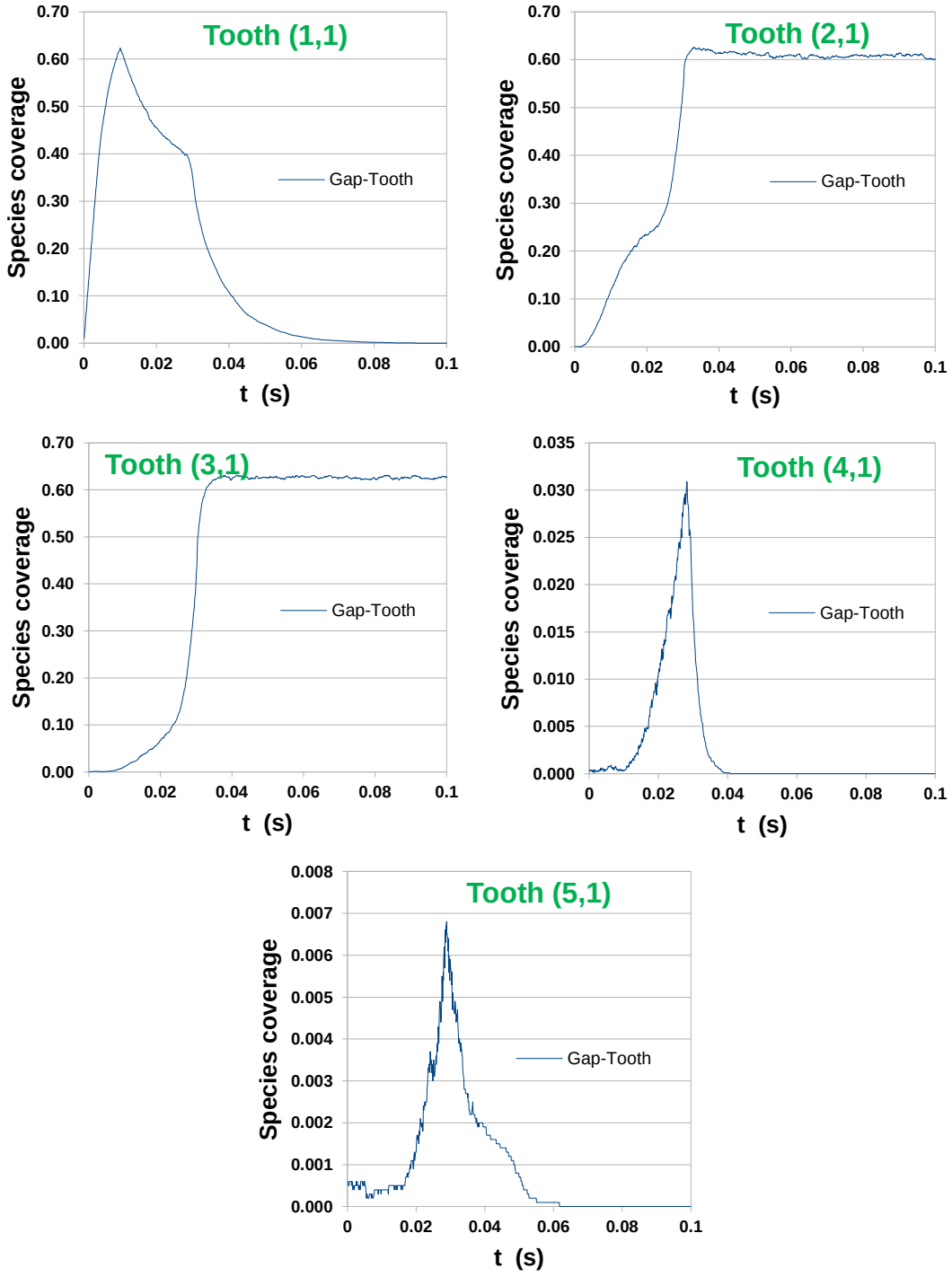


Figure A.2: Diffusing species coverage profiles in a gap-tooth framework, using quadratic (2^{nd} order) interpolation.