Multi-scale simulations with complex automata: in-stent restenosis and suspension flow

Lorenz, E.

Publication date
2010

Chapter 2

Complex Automata for multi-scale modeling[1]

2.1 Multi-scale Modeling

2.1.1 Introduction

Cellular Automata (CA) are generally acknowledged to be a powerful way to describe and model natural phenomena [1, 8, 9]. There are even tempting claims that nature itself is one big (quantum) information processing system, e.g. [10], and that CA may actually be nature's way to do this processing [11, 3, 5]. We will not embark on this philosophical road, but ask ourselves a more mundane question. Can we use CA to model the inherently multi-scale processes in nature *and* use these models for efficient simulations on digital computers? The ever increasing availability of experimental data on every scale, from 'atom to material' or from 'gene to health', in combination with the likewise ever increasing computational power [12, 13], facilitate the modeling and simulation of natural phenomena taking into account all the required spatial and temporal scales (see e.g. [14]). Multi-scale modeling and simulation, as a paradigm in Computational Science, is becoming more and more important, as witnessed by e.g. dedicated special issues [15] and thematic journals [16, 17]. Consider for example the field of physiology. The sequence from the genome, proteome, metabolome, physiome to health comprises multi-scale, multi-science systems [18, 19]. Studying biological sub-systems, their organization, and their mutual interactions, through an interplay between laboratory experiments and modeling and simulation, should lead to an understanding of biological function and to a prediction of the effects of perturbations (e.g. genetic mutations or presence of drugs) [20]. The concept 'from genes to health' is the vision of the Physiome [21] and ViroLab [22] projects, where multi-scale modeling and simulation of aspects of human physiology is the ultimate goal. Modeling such systems is a challenging problem but has the potential to improve our understanding of key interactions. The inherent complexity of biomedical systems is

[1] This chapter is based on A.G. Hoekstra, A. Caiazzo, E. Lorenz, J.-L. Falcone and B. Chopard, *Complex Automata: multi-scale Modeling with coupled Cellular Automata* in J. Krok, P.M.A. Sloot, A.G. Hoekstra (Eds): *Modeling of Complex Systems using Cellular Automata*, Springer, 2010.

now beginning to be appreciated fully; they are multi-scale, multi-science systems, covering a range of phenomena from molecular and cellular biology, via physics and medicine, to engineering and crossing many orders of magnitude with regard to temporal and spatial scales [23]. Despite the widely acknowledged need for multi-scale modeling and simulation, there is a scarcity of underpinning literature on methodology and generic description of the process. There are many excellent papers that present multi-scale models, but few methodological papers on multi-scale modeling (such as [24, 7]) have appeared. When using Cellular Automata to model a natural process, the lattice spacing and time step have a clear meaning in relation to the corresponding physical space and time of the process. We denote by $A(\Delta x, \Delta t, L, T)$ the spatio-temporal domain of a CA, whose spatial domain is made of cells of size Δx and it spans a region of size L, while the quantity Δt is the time step and T is the end of the simulated time interval. Therefore, processes with time scales between Δt and T can be represented and spatial scales ranging from Δx to L can be resolved. When executing such CA on a digital computer we note that the execution time T_{ex} scales as

$$T_{\text{ex}} \sim \frac{T}{\Delta t}\left(\frac{L}{\Delta x}\right)^{D}, \tag{2.1}$$

where D is the spatial dimension of the simulated domain. Trying to model a multi-scale system with a single CA would require to choose Δx and Δt in such a way that the smallest microscopic details and fastest dynamical response of the system are captured, yet the overall system size (L) and slowest dynamical time scale (T) need to be covered. For instance, in modeling human physiology the relevant range of spatial scales is from nanometer to meter (i.e. a factor 10^9) whereas temporal scale is from microseconds to human lifetime (i.e a factor 10^{15}). These numbers, in combination with (2.1) immediately show that one will probably never be able to simulate multi-scale systems with a single CA spanning such a wide range of scales.

The literature on using Cellular Automata to model multi-scale phenomena is relatively small, maybe with the exception of using CA to model land usage and geographical systems (e.g. [25]). Furthermore, many papers exist that use CA in multi-scale modeling, but there CA is typically coupled to other types of models (e.g. [26]). The bulk of CA multi-scale attempts are grid refinement methods, also termed multi-blocks. The idea is to adapt the local grid size to the local process scale, i.e. using a fine grid in regions where small scale processes occur and a coarse grid where larger scales are sufficient. A common approach is to couple grids of different scales with an overlap region [27].

Other ways of coupling multi-scale CA come from two theoretical frameworks. The first one is based on higher-order CA [28]. In this framework, the CA rules are not only able to change the cell state, but also the rules themselves, the neighborhood and the topology. Moreover, these models are also able to take into account hierarchical CA where higher level cells are connected to one or more lower level cells. The second one results from the work of Israeli and Goldenfeld [29] who have shown that it is possible to coarse-grain 1D nearest-neighbor CA, by defining a macroscopic CA whose behavior is similar to a microscopic CA. That is an important result because the authors have achieved the coarse-graining of CA known to be irreducible.

We developed a multi-scale, multi-science framework, coined Complex Automata (CxA), for modeling and simulation of multi-scale complex systems [30, 31]. The key idea is that a multi-scale system can be decomposed into N single-scale CA that mutually interact across the scales[2]. Decomposition is facilitated by building a Scale Separation Map (SSM) on which each single-scale system can be represented as an area according to its spatial and temporal scales. Processes having well-separated scales are easily identified as the components of the multi-scale model. We validate the CxA approach by building a large set of exemplary applications, and applying it to the challenging clinical problem of coronary artery in-stent restenosis (ISR) [32]. The CxA approach was developed within the context of the Coast project [33].

In this chapter we will review the current state of development of Complex Automata and explore the possibilities that are offered by Cellular Automata (CA) for multi-scale Modeling and Simulation.

2.2 Complex Automata

2.2.1 A definition

Formally, we shall define a CA as a tuple

$$\mathcal{C} = \{A(\Delta x, L, \Delta t, T), \mathbb{F}, \Phi, f_{\text{init}} \in \mathbb{F}, \mathbf{u}, \mathbf{O}\}. \tag{2.2}$$

A is the domain, made of spatial cells of size Δx and spanning a region of size L, while the quantity Δt is the time step and $T/\Delta t$ is the number of iterations during which the CA will be run. Therefore, processes with time scales between Δt and T can be represented and spatial scales ranging from Δx to L can be resolved. The state of the CA is described by an element of $\mathbb{F}$ (space of states) and it evolves according to the update rule $\Phi : \mathbb{F} \to \mathbb{F}$ (note that formally both $\mathbb{F}$ and Φ depend on the discretizations $(\Delta x, \Delta t)$). Additionally, we constrain the update rule to be in the form of *collision+propagation*, such that the operator Φ can be decomposed as a

$$\Phi = \mathbf{PCB}, \tag{2.3}$$

i.e. into a boundary condition, a propagation, and a collision operator, each depending, possibly, on the field $\mathbf{u}$ (see also section 2.2.3 for more details). The terminology collision-propagation is borrowed from the lattice gas automata framework (see e.g. [1]). This is equivalent to the more classical Gather-Update CA paradigm, as was formally demonstrated recently [34]. The initial condition (f_{init}) is a particular element of the space of states. At the spatial boundaries of A, additional information is needed (boundary conditions).

In definition 2.2, we introduced additional elements. The field $\mathbf{u}$ collects the external information exchanged at each iteration between the CA and its environment. The functional $\mathbf{O} : \mathbb{F} \to \mathbb{R}^d$, the *observable*, specifies the quantity we are interested in.

[2]Note that our approach is not limited to CA but also includes extensions such as lattice Boltzmann models and agent based models, because they can all be described by a generic update rule discussed in section 2.2.3.

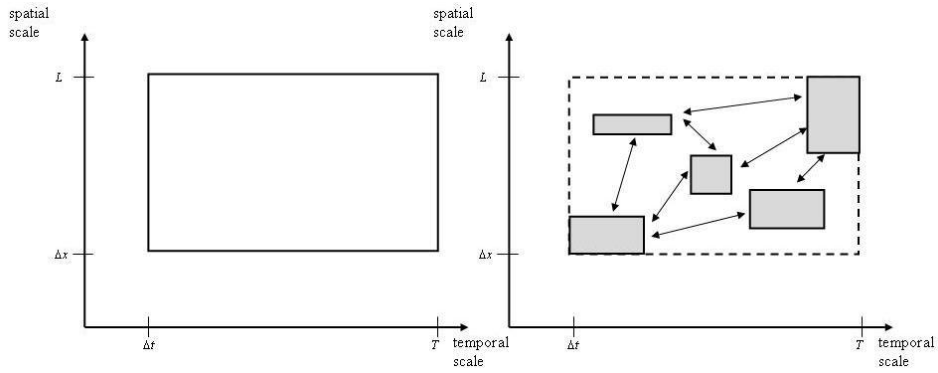


Figure 2.1: The Scale Separation Map with left a single CA and right a hypothetical CxA with 5 single scale CA modeling the same process.

A CxA can be viewed as a collection of interacting CA. Definition 2.2 suggests that a CxA can be represented as a graph $\mathcal{X} = (V, E)$ where V is the set of vertexes and E the set of edges with the following properties

- Each vertex is a CA $\mathcal{C}_i = \{A_i(\Delta x_i, L_i, \Delta t_i, T_i), \mathbb{F}_i, \Phi_i, f_{\text{init,i}} \in \mathbb{F}_i, \mathbf{u}_i, \mathbf{O}_i\}$
- each edge E_{ij} is a coupling procedure describing the interaction between $\mathcal{C}_i$ and $\mathcal{C}_j$. In practice, E_{ij} will define how and when information is exchanged between the two subsystems.

During the initialization phase, this problem-dependent graph is built according to the modeler's specifications.

2.2.2 The Scale Separation Map

A key idea behind CxA is that a multi-scale system can be decomposed into N single-scale Cellular Automata that mutually interact across the scales. The decomposition is achieved by building a Scale Separation Map (SSM) on which each system can be represented as an area according to its spatial and temporal scales. Processes having well separated scales are easily identified as the components of the multi-scale model.

Fig. 2.1 shows a SSM, where the horizontal axis represents the temporal scales and the vertical axis the spatial scales. On the left a CA with spatio-temporal domain $A(\Delta x, \Delta t, L, T)$ is represented on the SSM. Assuming that the process to be simulated is really multi-scale in the sense that it contains relevant sub-processes on a wide range of scales, simulations based on the finest discretizations are not really feasible (recall (2.1)), the approach we propose in CxA modeling is to try to split the original CA into a number of single-scale CA and let these CA exchange information in such a way that the dynamical behavior of the multi-scale process is mimicked as accurately as possible. This is shown schematically in the right part in Fig. 2.1. The subsystem in the lower left part operates on small spatial scales, and short time scales, the one at the upper right part operates at large scales, and the other three at intermediate scales. This could e.g. be processes operating at the micro-, meso-, and macro-scale.

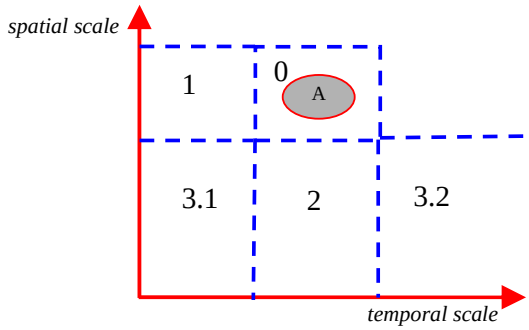


Figure 2.2: Interaction regions on the scale map.

After identifying all subsystems and placing them on the scale map, coupling between subsystems is then represented by edges on the map. For instance, a process can be coupled with another through a lumped parameter or through detailed spatially and temporally resolved signals, in which case they would typically share a boundary and synchronously exchange information. The distance between subsystems on the map indicates which model embedding method to use to simulate the overall system. In the worst case, one is forced to use the smallest scales everywhere, probably resulting in intractable simulations. On the other hand, if the subsystems are well separated and the smallest scale subsystems are in quasi-equilibrium, then they can be solved separately, although infrequent (possibly event-driven) feedback between the subsystems will still be required.

Consider two processes A and B with their own specific spatial - and temporal scale, denoted by ξ_i and τ_i respectively ($i \in \{A, B\}$). Assume that A has the largest spatial scale. In case the spatial scales are the same, A has the largest temporal scale. In other words, $(\xi_B < \xi_A)$ OR $(\xi_B = \xi_A$ AND $\tau_B < \tau_A)$. We can now place A on the scale map and then investigate the different possibilities of placing B on the map relative to A. This will lead to a classification of types of multi-scale coupling, as in Fig. 2.2.
Depending on where B is, we find the following regions:

B in region 0: A and B overlap, so we do not have a scale separation, we are dealing here with a single-scale multi-science model.

B in region 1: Here $\xi_B = \xi_A$ AND $\tau_B < \tau_A$, so we observe a separation of time scales at the same spatial scale.

B in region 2: Here $\xi_B < \xi_A$ AND $\tau_B = \tau_A$, so we observe a separation in spatial scales, like coarse and fine structures on the same temporal scale.

B in region 3: Separation in time - and spatial scales. Region 3.1 is the well-known micro $\Leftrightarrow$ macro coupling, so fast processes on a small spatial scale coupled to slow processes on a large spatial scale. This type of multi-scale model has received most attention in the literature. In region 3.2 we have the reversed situation, a slow process on small spatial scales coupled to a fast process on large spatial scales. We believe that

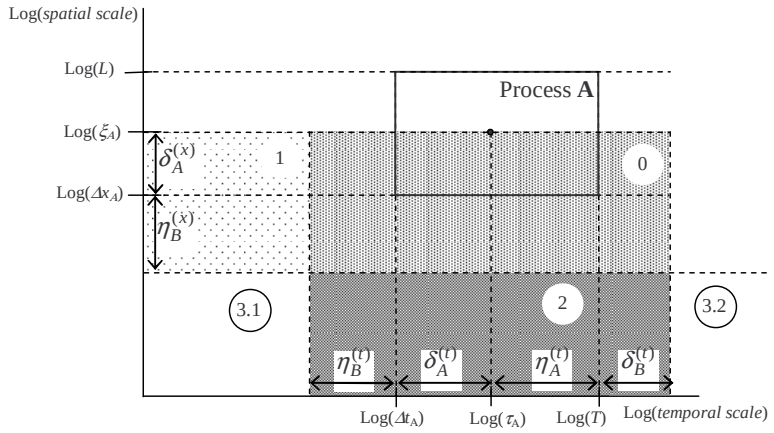


Figure 2.3: Position of a process A with parameters $\{\xi, \delta^{(x)}, \eta^{(x)}; \tau, \delta^{(t)}, \eta^{(t)}\}$ and the interaction regions on the logarithmic scale map.

this region is very relevant in for instance coupling of biological with physical processes, where the biological process is e.g. the slow response of cells to a faster physical process on a larger scale (e.g. blood flow in arteries).

Note that we do not have to consider other regions of the scale map, because then the role of A and B just reverses, and we fall back to one of the five cases identified above.

Next we address the question of the area that processes A and B occupy on the SSM. As discussed earlier, a 1D CA is characterized by a spatial discretization Δx and a system size L. We assume that Δx and L have been chosen such that the spatial scale of the process is well represented on this CA, so at least we will have $\Delta x < \xi < L$. We define $N^{(x)}$ as the number of CA cells that extend the full domain, i.e. $N^{(x)} = L/\Delta x$. Next assume that the discretization has been chosen such that the spatial scale is represented by $10^{\delta^{(x)}}$ cells (i.e. $\Delta x = \xi/10^{\delta^{(x)}}$) and the spatial extension of the CA is $10^{\eta^{(x)}}$ times the spatial scale, i.e. $L = \xi 10^{\eta^{(x)}}$, and therefore $N^{(x)} = 10^{\eta^{(x)}+\delta^{(x)}}$. Likewise for the temporal domain, i.e. a single scale CA has a time step Δt and the CA is simulated over a time span T, and we have $\Delta t < \tau < T$. The number of time steps $N^{(t)} = T/\Delta t$. The discretization has been chosen such that the temporal scale is represented by $10^{\delta^{(t)}}$ time steps (i.e. $\Delta t = \tau/10^{\delta^{(t)}}$) and that simulation time of the CA is $10^{\eta^{(t)}}$ times the temporal scale, i.e. $T = \tau 10^{\eta^{(t)}}$ and $N^{(t)} = 10^{\eta^{(t)}+\delta^{(t)}}$.

A process' position on the scale map is now fully determined by the tuple $\{\xi, \delta^{(x)}, \eta^{(x)}; \tau, \delta^{(t)}, \eta^{(t)}\}$, and is drawn in Fig. 2.3, where the axes are now on a logarithmic scale. On such logarithmic SSM the process is rectangular with an area $(\delta^{(t)} + \eta^{(t)}) \times (\delta^{(x)} + \eta^{(x)})$ asymmetrically centered around the point $(\log(\tau), \log(\xi))$. In the special case that $\delta^{(x)} = \eta^{(x)} = \delta^{(t)} = \eta^{(t)} = 1$ (a reasonable first order assumption) we see that the process is symmetrically centered around $(\log(\tau), \log(\xi))$ and that the size of the box extends 2 decades in each dimension.

In Fig. 2.3 we show the extension of Fig. 2.2, where regions $1-3$ now have well

defined positions and size. Depending on the location of process B, that is the point $(\log(\tau_B), \log(\xi_B))$ on the SSM, and with all information on the spatial and temporal extensions of processes A and B, we can unambiguously find in which region of the scale map they are located with respect to each other. The scale separation between two processes can now clearly be defined in terms of a distance on the SSM, and this can then become an important measure to determine errors that are induced by scale splitting procedures. This is further elaborated in Section 2.3.

Consider once more region 3, where there is a separation in time and length scales. In region 3.1 we find that $L_B < \Delta x_A$ and $T_B < \Delta t_A$. As said earlier, this is the classical micro $\Leftrightarrow$ macro coupling, and in our language this means the *full* spatio-temporal extend $T_B \times L_B$ of process B is smaller than one single spatio-temporal step $\Delta t_A \times \Delta x_A$ of process A. A number of modeling and simulation paradigms have been developed for this type of multi-scale systems (see e.g. [7]).

Region 3.2 also exhibits separation of time and length scales, but now the situation is quite different. We find that, just like in region 3.1, $L_B < \Delta x_A$. So, the spatial extend of process B is smaller than the grid spacing of process A. However, now we find that $T_A < \Delta t_B$. In other words, the full time scale of process A is smaller then the time step in process B. This will result in other modeling and simulation paradigms than in region 3.1. Typically, the coupling between A and B will involve time averages of the dynamics of the fast process A.

Let us now turn our attention to the regions where there is overlap on the temporal - or spatial scales, or both (regions 0, 1, and 2, in Fig. 2.3). In all these cases we can argue that we have partial or full overlap of the scales, giving rise to different types of (multi-scale) modeling and simulation. We say that the scales fully overlap if the point $(\log(\tau_B), \log(\xi_B))$ falls within (one of) the scales spanned by process A. On the other hand, there is partial overlap if $(\log(\tau_B), \log(\xi_B))$ falls *outside* (one of) the scales spanned by process A, but the rectangular area of process B still overlaps with (one of) the scales spanned by process A. The region of partial scale overlap can also be considered as a region of gradual scale separation, a boundary region between the scale separated regions 1, 2 and 3 and region 0. Simulations of this kind of multi-scale system would typically involve CxA's with local grid refinements, or multiple time stepping approaches, or a combination of both.

2.2.3 The Sub-Model Execution Loop

A second important ingredient of the CxA formalism is the observation that each CA (i.e. vertex of the CxA) can be expressed with a common instruction flow. This gives a way to identify generic coupling templates and achieve a precise execution model (see also section 2.2.6). Using the specific collision+propagation form of the update rule, as introduced in section 2.2.1, we represent the workflow with a pseudo-code abstraction, termed the *Sub-model Execution Loop* (SEL), as shown below.

```
D := D_init              /* initialization of the domain */
f := f_init              /* initialization of state variables */
t := 0                   /* initialization of time */
    While Not EC
        t += Δt          /* increase time with one timestep t */
        D := U(D)        /* update the domain */
        f := B(f)        /* apply boundary conditions */
        f := C(f)        /* collision, update state of cells */
        f := P(f)        /* propagation, sent information to neighbors */
        O_i(f)           /* compute observables from new state */
    End
O_f(f)                   /* compute observables from final state */
```

Note that in the SEL, operators are written in bold and (state) variables as plain characters. The CA operates on a computing domain `D`, being the lattice of cells and the boundaries. Each cell in a CA has a set of state variables `f`. At the start of the SEL the domain and the state variables are initialized by the operators $\mathbf{D}_{\text{init}}$ and $\mathbf{f}_{\text{init}}$ respectively. The simulation time `t` is set to an initial value (`0` in this case). After initialization the CA enters into an iteration loop, whose termination is controlled by an end condition computed by **EC**. The end condition can simply be a fixed number of iterations, but could also be some convergence criterion depending upon the state variables. Within the main iteration loop, the time is first increased with a time step Δ`t`. Next the domain is updated by the operator **U**. If the domain is static, this operator is just the identity operator **I**. However, in many models the domain is dynamic. For instance, new cells can be created or existing cells removed (e.g. due to the movement of the boundary). In all these cases **U** will execute these domain updates. Next, the sequence **PCB**`(f)` is executed. First, the operator **B** applies the boundary conditions. This means that missing information is constructed that is needed for the actual state updates by **C** (see below) of the cells lying at the boundary of the domain `D`. For instance, if the state variables represent a concentration of some species, the boundary condition could specify a flux of those species into the domain, and from that missing information on the domain boundary cells is computed. Next the actual state change of all cells is computed by the Collision operator **C**. Finally, information is sent to neighboring cells or agents by the Propagation operator **P**. The CA is now updated for the current time step, and the simulation can proceed to the next iteration. However, before doing so an intermediate observation operator $\mathbf{O}_i$ computes observables from the state variables `f`. After termination of the main iteration loop a final observation is done of the state variables with the $\mathbf{O}_f$ operator.

2.2.4 CxA Multi-Scale Coupling

Despite the growing literature there is not a well accepted generic methodology, nor a well-defined nomenclature of multi-scale modeling. A few authors have proposed different typologies of multi-scale models. Weinan E et al. [7] have proposed 4 types of multi-scale problems and 4 general strategies. Despite the many examples given by them the relevance of their classification is not always clear, because they

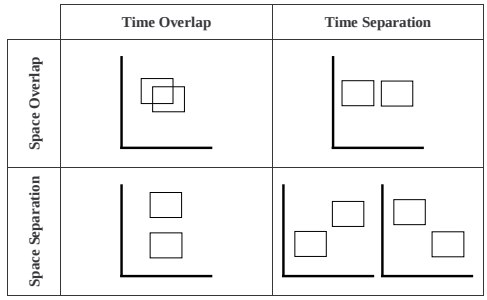


Figure 2.4: Overlap or separation on spatial and temporal scales resulting from the relative positions of sub-processes leading on a SSM.

single out, in all their examples, one specific item from their classification, and do not further discuss the relevance or completeness of the other classes. Another proposition for a multi-scale modeling methodology is that of Ingram. Working on chemical engineering simulations, Ingram et al. [24] have defined five types of macro-micro scale coupling. Ingram et al. present simulation examples for three types of coupling, showing that different strategies may be used to solve the same problem. The choice of coupling has an influence on both computational efficiency and accuracy. The fact that it is not always easy or possible to make the correspondence between the approaches by Ingram et al. and E et al. indicates that the topic of multi-scale modeling lacks consensus. This lack of consensus on terminology and methodology can be attributed to the fact that actual coupling methodologies were mixed with classifications of the computational domain and/or with the type of scale separation (temporal, spatial, or both).

In the following discussion we try to clarify the situation, in the framework of the CxA formalism. However, we believe that this is also relevant to multi-scale modeling in general. Based on the discussion on the SSM in section 2.2.2, we identified 5 different types of scale separation. We call them *Interaction Regions* on the SSM, and they are shown in Fig. 2.4. Another important parameter to distinguish multi-scale models is the Domain type. We distinguish between *single Domain* (sD) and *multi-Domain* (mD) types. In case of sD processes A and B can access the whole simulated domain and communication can occur everywhere, whereas in case of mD each process is restricted to a different physical region and communication can only occur across an interface or small overlap region.

For each combination of interaction region and domain type we can now try to identify a multi-scale coupling. We will base our approach on the SEL discussed in section 2.2.3, and show which operators from the SEL are coupled to each other. We call this *Coupling Templates*. As an example consider Weinan E's Heterogeneous Multi-scale Method [7]. On close inspection we must conclude that this is a Coupling Template for single Domain processes in interaction region 3.1. In terms of the SEL of the macroscopic process A and the microscopic process B we find as Coupling Template $\mathbf{O}_f^B \rightarrow \mathbf{C}^A$; $\mathbf{O}_i^A \rightarrow \mathbf{f}_{\text{init}}^B$ (see also Fig. 2.8). At each time step of the macroscopic process B a microscopic process A is initialized using macroscopic information. The microscopic model then runs to completion and sends final information to the collision operator of the macroscopic process.

We are currently investigating many examples of multi-scale models, their map-

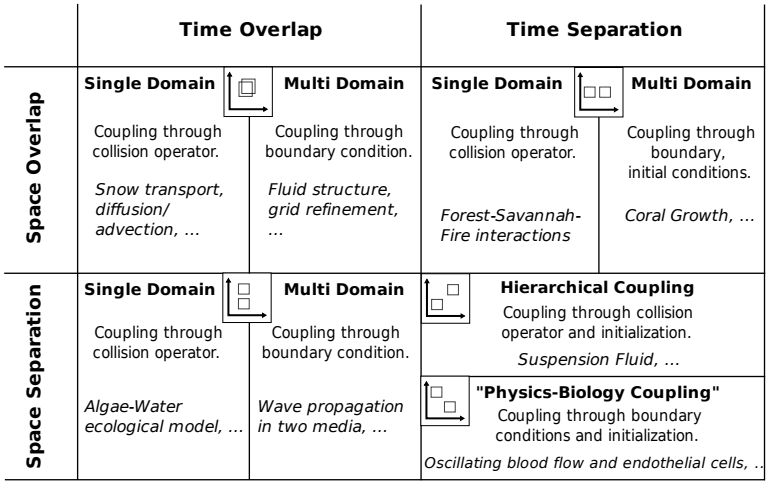


Figure 2.5: Our classification of multi-scale problems, for systems that can be reduced of two single-scale processes. This classification is based on the five interaction regions given by the SSM, and the domain type (sD or mD). For each class, the generic coupling template is indicated, in terms of the CxA operators. Examples of specific applications belonging to the given categories are indicated in italic.

ping to a CxA model, and resulting coupling templates. A detailed discussion of the results will be reported later. As a summary, some of the examples are indicated in Fig. 2.5. At this stage we can extract two observations:

- In the case of time scale overlap, the coupling will occur inside the inner iteration loop. In contrast, in the case of time scale separation, coupling is realized outside the inner loop through the initialization operators and the final observation operator.

- Single-domain models are coupled through the collision operator. Multi-domain models are coupled through the domain update or the boundary operators.

Based on our current set of examples, we hypothesize that for each type of multi-scale model, classified in terms of domain type and interaction region, only a very small set of coupling templates exists. If this is true, this would lead the way to a powerful CxA multi-scale modeling and simulation strategy, including a multi-scale modeling language, generic simulation software and a mathematical framework to analyze errors involved in CxA modeling. In what follows we will further elaborate on these ideas, sketching the contours of such a generic CxA based multi-scale modeling approach.

2.2.5 Multi-scale modeling strategies

A key question when dealing with a multi-scale system is how to decompose it in several coupled single-scale sub-processes. This decomposition is certainly not

unique and a good knowledge of the system may be required. Once the subprocesses are chosen, this specifies the relation between the computational domains and the interaction regions on the SSM. Then, our classification scheme indicates the expected coupling templates.

We have observed several strategies that can be used to deal with systems having a broad range of scales and to reduce their area on the scale separation map. They are briefly discussed below.

Time splitting: This approach is appropriate when two processes act at different time scales. Let us assume we have a sD problem described with a propagation operator P and a collision operator C that is the product of two operators

$$P_{\Delta t} C_{\Delta t} = P_{\Delta t} C^{(1)}_{\Delta t} C^{(2)}_{\Delta t} \tag{2.4}$$

where Δt specifies the finer scale of the process. Then, if $C^{(1)}_{\Delta t}$ acts at a longer time scale than $C^{(2)}_{\Delta t}$ we can approximate M iterations of the dynamics as

$$[P_{\Delta t} C_{\Delta t}]^M \approx P_{M\Delta t} C^{(1)}_{M\Delta t} [C^{(2)}_{\Delta t}]^M \tag{2.5}$$

We will illustrate this time-splitting strategy in detail in section 2.3.

Coarse graining: The goal of coarse graining is to express the dynamics of a given system at a larger temporal and/or spatial scale in some part of the computational domain where less accuracy is needed. After coarse graining we obtain a new process, specified by new collision and propagation operators and occupying a reduced area on the SSM. Within our formalism, a space-time coarse graining of a factor 2 can be expressed as

$$[P_{\Delta x} C_{\Delta x}]^n \approx \Gamma^{-1} [P_{2\Delta x} C_{2\Delta x}]^{n/2} \Gamma \tag{2.6}$$

where Γ is a projection operator mapping the system state, originally defined on a fine scale Δx, to a scale $2\Delta x$. The exponent n is the number of iterations needed to simulate the problem.

Amplification: This strategy can be used to reduce the larger time scale of a process. For instance, we can consider a process acting with low intensity but for a long time, in a time periodic environment, such as a growth process in a pulsatile flow.

Within our formalism, let us consider two coupled (mD) processes which are iterated $n \gg 1$ times

$$[P^{(1)} C^{(1)}]^n \qquad \text{and} \qquad [P^{(2)} C^{(2)}(k)]^n \tag{2.7}$$

where k expresses the intensity of the coupling of process 1 to process 2.

If the $C^{(1)}$ is periodic with period $m \ll n$, we can approximate the above evolution as

$$[P^{(1)} C^{(1)}]^m \qquad \text{and} \qquad [P^{(2)} C^{(2)}(k')]^m \tag{2.8}$$

with k' the new effective intensity of the coupling. For a linear coupling we would have $k' = (n/m)k$.

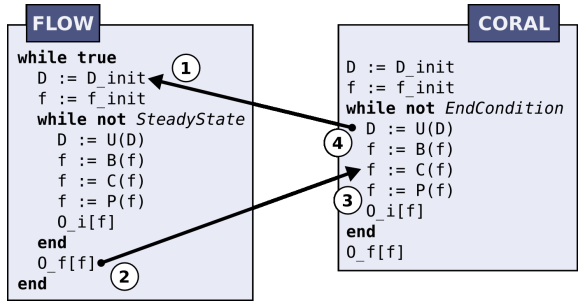


Figure 2.6: Coupling template for the so-called coral growth model. Numbers corresponds to the communication operation described in Fig. 2.7.

2.2.6 Execution Model

Coupling several sub-models, using coupling templates raises implementation issues. A typical situation is shown in Fig. 2.6 for the problem of coral growth. The growth of branching corals is modeled with the aim to understand the influence of abiotic factors (transport of nutrients by flow and diffusion) on the morphology. This is work performed under the supervision of Dr. Jaap Kaandorp, and for biological context and background we refer to [35], his recent book [36] and to [37, 38]. In short, this model works as follows: the fluid flow is transporting nutrients that are needed by the coral to grow. There is a clear time scale separation that can be exploited. Fluid flow establishes at a few seconds whereas the coral grows at a much slower pace. According to the coupling template shown in Fig. 2.6, the fluid solver is run until steady state and the resulting flow field is passed to the coral solver for calculating the growth rate. The new geometry of the coral is then used to build a new initial condition for the flow solver. The process stops when enough iterations of the coral solver have been performed.

Using this example we will explain the main concepts of our proposed execution model for CxA, which is compatible with the asynchronous channel actor-model framework [39]. A computer implementation for a CxA simulation environment, implementing this execution model, has been realized [40] and a public domain release is available[3].

CxA components

For the sake of the present discussion, CxA can be described as directed bipartite graphs whose edges represent a single direction communication channel and the vertexes are either *kernels* or *conduits*. The kernels are the main computational units of a CxA. Generally, kernels are the single-scale sub-model solvers as described above. However, when needed, they can also execute other tasks such as measurements or complex data mappings. The conduits are "smart" communication channels. Each conduit connects a pair of kernels together in an oriented fashion and, in principle, only one quantity is transported per conduit. These conduits are composed of three parts:

(1) an incoming buffer (the entrance)

[3] see http://www.complex-automata.org

(2) an outgoing buffer (the exit)

(3) (optional) one or several data *filters* between different scales (to perform interpolation, restriction, discretization, etc.)

Conduits work in a purely reactive way: when data is copied at the entrance, the conduit applies the filters and moves the resulting data into the outgoing buffer. Each conduit is connected to only two kernels, but kernels can be connected to an arbitrary number of conduits. Each component is either a full process or a thread depending on the implementation. They can reside in the same machine or be distributed across a network.

CxA Communication

In CxA, kernels communicate exclusively via conduits, using a message passing paradigm. Only two communication primitives are defined to interact with conduits:

1. send(data): this primitive sends a data vector from a kernel to a conduit entrance. It is non-blocking, since it returns as soon as the data is sent to the conduit, whether or not the destination process has read the data. This corresponds to a push communication.

2. receive(): this primitive allows a kernel to receive data from a conduit exit. This primitive is blocking, it will return only when the desired data exist in the conduit. The receiving kernel will then simply wait until the data is available before resuming its computations. This corresponds to a pull communication.

Conduits entrances and exits are supposed to have large buffers, able to store several large data structures. These buffers act as FIFO ("first in, first out") where each entry is a reference to a date-structure. So, if the sending kernel is faster than the receiving one, several data vectors will be stored in the exit buffer, waiting for a receive() call from the destination kernel. The FIFO nature of the buffer ensures that the data are always read in the correct time order. The actual communication can be either a memory copy if the kernel and conduit reside in the same processor, or a network communication if both components reside on different machines. Note that the conduit could also be used to implement a mutex coordination primitive in case of shared memory execution.

Let us consider again the example of the coral growth. The coral SEL represented in Fig. 2.6 can be rewritten as follows, to include the two communication primitives explicitly:

```
While Not EC
   D := U(D)
   DomainConduit.send( D )
   f := B(f)
   velocityMap := VelocityConduit.receive()
   f := C(f,velocityMap)
   f := P(f)
End
```

CxA Initialization and start

CxA initialization occurs in a semi-decentralized way. First, each conduit and kernel is spawned (possibly on several machines). Then a special process, termed *plumber*, is responsible for connecting each kernel with the entrances and exits of the relevant conduits. The plumber terminates as soon as this basic task is finished. The rest of the initialization process is then fully decentralized:

1. As soon as a kernel is fully connected with the required conduits, it starts its computations. If it is sending data to a yet unconnected kernel, the data will be kept in the conduit until the receiver is active and reading. On the other hand, if a conduit tries to receive data originating from an unconnected kernel, it will hang on until the sending kernel connects and transmits data.

2. For conduits the situation is even simpler. Since they are purely reactive components, nothing will happen in an unconnected conduit. Similarly, if only the conduit exit is connected, the conduit will do nothing. In contrast, if only the conduit entrance is connected, the conduit will simply process incoming data which will be accumulated in the exit buffer. Therefore, the conduit is always in a valid state (assuming it has enough internal memory).

CxA Synchronization

CxA graphs are usually cyclic. Even the basic examples with just two single-scale models (see Fig. 2.6) will display a communication cycle if both models can influence one another. Moreover CxA are multi-scale systems and kernels can thus function at different time scales, maybe in an adaptive way. These properties make a central scheduler approach impractical. However, the fact that the receive primitive is blocking and the send is non-blocking, allows a data-driven synchronization to occur naturally. Indeed, kernels will just wait until information is available before continuing their computation. An example of such synchronization is shown in Fig. 2.7 for the coral model.

The main problem with this method are possible deadlock situations. However, such issues can be easily prevented with the CxA execution model. In the coral example, deadlock is avoided by having a model (the coral) which sends before receiving. This allows the flow model to continue its computations to produce the data that will unlock the coral, etc. In contrast, the situation presented in Fig. 2.8 will produce a deadlock because both models try to receive before sending anything. This problem is easily solved by moving the observation $\mathbf{O}_i$ at the beginning to the inner loop, or adding initial send instructions before entering the submodel execution loop.

Furthermore, the fact that communication is pairwise and that the conduits use buffers, makes race conditions impossible. Data are meant to be read by only one process, data sent in a conduit entrance will be processed only by that conduit and data moved to conduit exits will concern only a single kernel.

CxA Termination

The termination of the whole CxA is also designed to be fully decentralized: when a kernel finishes its computations (because of e.g. a preset maximum time or a

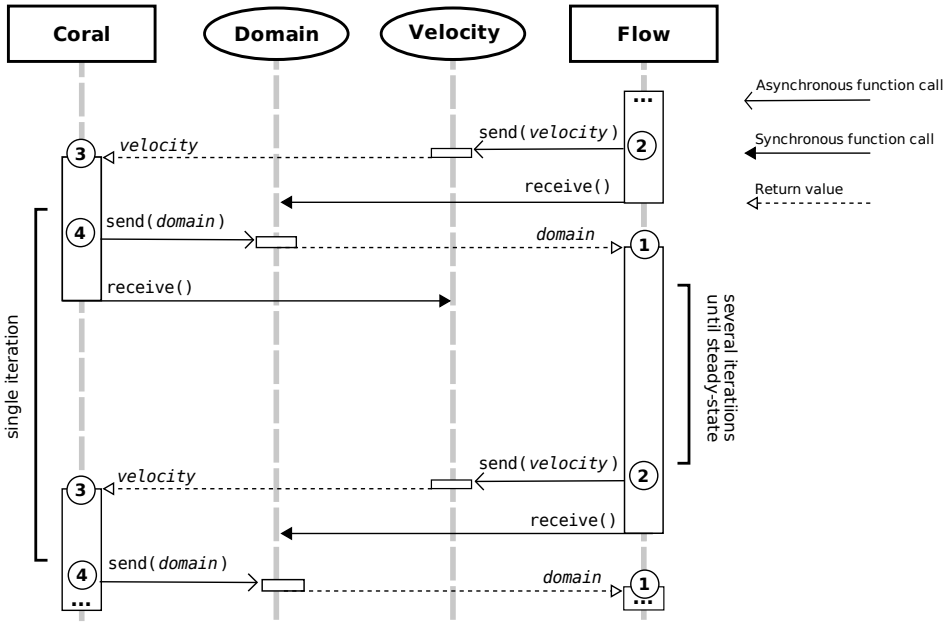


Figure 2.7: UML sequence diagram of the CxA shown in Fig. 2.6. The vertical lines represent the "life-line" of the process: the kernels are represented by rectangles and the conduits by ovals. When a process is active, the gray life line is replaced by a vertical white rectangle. The arrows represent interaction. Solid arrows with triangular heads are blocking interactions and solid arrows with thin heads represent non-blocking interactions. The return values are indicated by dashed arrows. The circled numbers correspond to Fig. 2.6.

steady state condition), it first notifies all its conduits and then it terminates itself. Similarly, when a conduit receives termination notifications from all connected kernels, it can terminate itself. While the conduit termination rule is always safe (a conduit stops when no kernel is connected anymore), the kernel termination rule needs an extra mechanism. Otherwise, a problem occurs if a kernel is waiting for information from an already terminated kernel.

For instance, in the coral example (Fig. 2.6) the flow model will hang on for the domain update, even after the coral model termination. To solve this issue a stop signal is introduced which is able to release a kernel blocked in the receive primitive. This signal is propagated by a kernel through the existing conduits, using a third primitive: stop(): this primitive sends the stop signal through a conduit. The receive primitive is then modified slightly. It works exactly as seen above but can return either the expected data or the stop signal.

Therefore a kernel waiting for data can be released by a stop signal. Kernels are then responsible to send, process and propagate stop signals. Generally a kernel receiving a stop signal should:

1. Abort the submodel execution loop.
2. Send some final data, if required.
3. Propagate the stop signal to each connected conduit entrances.
4. Notify each connected entrance and exit.
5. Terminate itself gracefully.

With this termination scheme, all kernels which need data from the rest of the CxA will thus stop. The stop signal can originate from any kernel, and this approach also works if two (or more) kernels reach a stop condition at the same time.

As an illustration we can add a stop mechanism to the example of Fig. 2.6, as follows:

1. Coral submodel

```
While Not EC
   D := U(D)
   DomainConduit.send( D )
   f := B(f)
   velocityMap :=
      VelocityConduit.receive()
   f := C(f,velocityMap)
   f := P(f)
End
DomainConduit.stop()
myStop()
```

2. Flow submodel

```
While True
   domain :=
      DomainConduit.receive()
   If domain == STOP_SIGNAL
      myStop()
   D := domain
   f := f_init
   While Not Steady_State
      [SEL]
   End
End
```

where `myStop()` is a user-defined function which terminates the kernel. But, before, if needed, it: (i) saves results, (ii) propagates the stop signal, (iii) notifies the connected conduits.

Parallelization

With the execution model described above, our framework is compatible with a distributed or grid computing approach, in which each submodel could run on a different core or, alternatively, as a different threads on the same core. The actual support for parallelization depends on the chosen implementation of our framework. For instance, the MUSCLE library[4] offers an easy but manual parallelization. On the other hand the CxA-lite library[5] only allows a multi-thread execution in which all the submodels share the same memory space.

MUSCLE Library

Within the framework of the COAST project the conceptual ideas behind the CxA approach (decomposition into single scale models, restriction to a common instruction flow and specification of finite number of coupling templates) have been used to develop the *Multi-scale Coupling Library and Environment* (MUSCLE) [40], a software environment in which a CxA can be implemented naturally.

Within the coupling library, both the kernels (i.e. the single scale models) and the conduits (i.e. the multi-scale coupling) are software agents of the underlying multi-agent platform JADE[6]. Kernels and conduits (conceptually central to the CxA modeling language) communicate using the two communication primitives of non-blocking `send` and blocking `receive`. The single scale models do not need to be aware of each other and the information on the coupling and the global setup are held by the framework. This allows the implementation of complex interfaces, where multi-scale couplings can be performed by the use of smart conduits.

Furthermore, the structure of the coupling library allows complete independence from native codes. These can be replaced with a different source, provided the interface with respect to the framework (i.e. the JAVA-wrapper agent) remains the same. In the particular example of in-stent restenosis, described in Sec. 3, three (in case of the 2D model) or four (3D model), respectively, single scale models have been implemented in different programming languages (FORTRAN90, C++, JAVA), wrapped as JAVA agents, and connected via the MUSCLE framework.

2.2.7 Formalism

The concept of a CxA as a set of coupled CA's, where the coupling is expressed in terms of input-output relations between operators of the SEL of the coupled CA's is not just a concept that allows to classify multi-scale models, as discussed in section 2.2.4, or a powerful concept to built CxA simulation software, see section 2.2.6, but it is also amenable to mathematical formalism and analysis. This section will introduce some of the formalism, which will be further used in one of the examples of section 2.3.

Recalling (2.2), the state of a CA at a certain time t is described by a $f^t \in \mathbb{F}$, denoting the numerical solution at the time step t, which evolves according to

$$\begin{aligned} f^0 &= f_{\text{init}}[\mathbf{u}_0], \text{ initial condition} \\ f^{t+\Delta t} &= \Phi[\mathbf{u}; f^t] \end{aligned} \tag{2.9}$$

[4] http://developer.berlios.de/projects/muscle
[5] http://github.com/paradigmatic/CxALite/
[6] `www.jade.tilab.com`

where $\mathbf{u}_0$ is an external field connected to the initial condition. As previously discussed, we constrain the update rule Φ to the form

$$\Phi[\mathbf{u}; f] = (\mathbf{B}[\mathbf{u}_B] \circ \mathbf{P} \circ \mathbf{C}[\mathbf{u}_C])\,[f], \tag{2.10}$$

i.e. written as a composition of three operators: *collision* $\mathbf{C}[\mathbf{u}_C]$, depending on external parameters $\mathbf{u}_C$, *propagation* $\mathbf{P}$, depending on the topology of the domain, and *boundary condition* $\mathbf{B}[\mathbf{u}_B]$, depending on external parameters denoted by $\mathbf{u}_B$.

More precisely, the space of the states $\mathbb{F}$ and the update rule Φ depend in general on the discretization parameters Δx and Δt. For simplicity, in what follows, we let the definition of CA depend also on a (small) parameter h, related to spatial and temporal discretizations (for example $\Delta x_h = h$, $\Delta t_h = \alpha h$). Accordingly, considering the CA $\mathcal{C}_h$, the evolution space and the update rule can be denoted as: $\Phi_h : \mathbb{F}_h \to \mathbb{F}_h$. Shortly, we will call f_h the numerical outcome of the CA $\mathcal{C}_h$.

To begin with, as in the left diagram in Fig. 2.1, we consider a multi-scale system represented as a single $\mathcal{C}_h$ defined as in (2.2). Building a CxA, instead of describing the system with a single f_h, we lower the dimension of the problem and the computational complexity, introducing coarser temporal and/or spatial discretizations

$$H = (h_1, \ldots, h_M) \tag{2.11}$$

and building a corresponding Complex Automaton

$$\mathrm{CxA}_H = (\mathcal{C}_{h_1}, \ldots, \mathcal{C}_{h_M}), \tag{2.12}$$

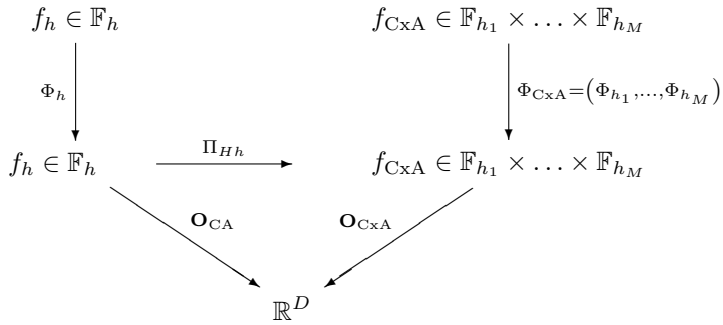
where each $\mathcal{C}_{h_m}$ is an object as in (2.2).

Formally, the definition of a CxA can be summarized in two steps. First, a projection of the space of states $\mathbb{F}$ on a product of spaces is considered

$$\Pi_{Hh} : \mathbb{F}_h \to \mathbb{F}_{h_1} \times \ldots \times \mathbb{F}_{h_M}, \tag{2.13}$$

each describing the evolution of a *single scale model* (on different discretizations). Second, a *rescaling* of the update rule is performed, according to the new discretizations h_i, on each space $\mathbb{F}_{h_i}$, for $i = 1, \ldots, M$, depending on the multi-scale technique used. Due to the form of the execution model of each CA, the rescaling can be easily expressed in terms of operations on the operators $\mathbf{P}$, $\mathbf{C}$, $\mathbf{B}$. Note that the spaces $\mathbb{F}_{h_i}$ are not necessarily disjoint, i.e. part of a single scale evolution space could be shared by several CA, in case of space overlap and single domain coupling.

Let us denote with f_{CxA} the numerical outcome of the complex automata simulation and with f_{h_m} (or f_m) the state variable of the single CA's. To be able to compare the results of the CxA versus the original multi-scale algorithm, we associate an observable $\mathbf{O}_{\mathrm{CxA}}$ to the Complex Automata, which projects the result f_H on the space of $\mathbf{O}(f_h)$. A sketch of the relevant spaces and operators is drawn below.



For the sake of simplicity, in what follows we describe the formalism restricting ourselves to the evolution of two coupled single scale models. From equations (2.9)-(2.10), we have the following general representation

$$\begin{aligned} f_1^{t_0} &= f_{\mathrm{init},1}[f_2] \\ f_1^{t+\Delta t_{h_1}} &= \left(\mathbf{B}_{h_1}[f_2] \circ \mathbf{P}_{h_1} \circ \mathbf{C}_{h_1}[f_2]\right)[f_1^t], \\ f_2^{t_0} &= f_{\mathrm{init},2}[f_1] \\ f_2^{t+\Delta t_{h_2}} &= \left(\mathbf{B}_{h_2}[f_1] \circ \mathbf{P}_{h_2} \circ \mathbf{C}_{h_2}[f_1]\right)[f_2^t], \end{aligned} \tag{2.14}$$

where two CA's are fully coupled in all the components. In detail,

- $f_{\mathrm{init},1}[f_2]$ denotes a coupling through initial conditions (i.e. the initial condition of 1 depends on the results of 2)
- $\mathbf{B}_i[f_j]$ expresses coupling through boundary conditions,
- $\mathbf{C}_i[f_j]$ expresses the coupling through collision operator.

In general, for different situations (multi-domain/single-domain, time/space separation/overlap) we can restrict the set of possible couplings to a well-specified coupling template. Consider the example of a microscopic fast process coupled to a macroscopic slow process (micro-macro coupling), as introduced earlier in section 2.2.4. The macroscopic process takes input from explicit simulations of microscopic processes at each time step and on each lattice site of the macroscopic process. The microscopic processes run to completion, assuming that they are much faster than the macroscopic process and therefore are in quasi-equilibrium on the macroscopic time scales (this approach is known in the literature as the Heterogeneous Multi-scale Method, see [7]). The macroscopic process could e.g. be a fluid flow with takes its viscosity from an underlying microscopic process (e.g. explicit suspension model).

In Fig. 2.8 we show for this example of micro-macro coupling the SSM (left) and the coupling template (right). The later is defined in [30] and shows how the operators as defined in (2.10) are coupled to each other. A close inspection of this coupling template shows indeed that, upon each iteration of the macroscopic process, the microscopic process executes a complete simulation, taking input from the macroscopic process. In turn, the output from the microscopic process is fed into the collision operator of the macroscopic process.

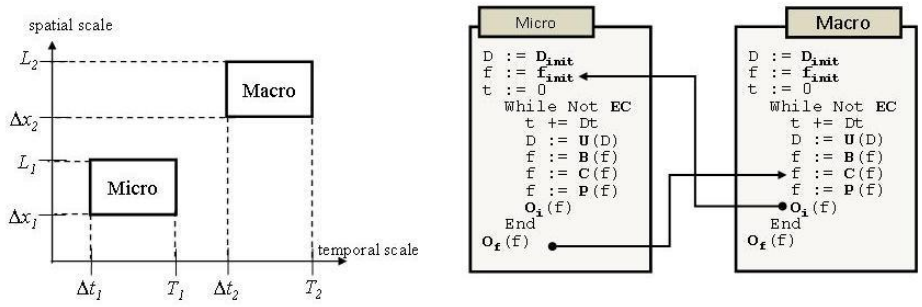


Figure 2.8: Micro-macro coupling example. Left: SSM. Right: coupling template.

We can formulate the CxA dynamics as follows (based on (2.14))

$$\begin{aligned} f_1^{t_0} &= f_{\text{init},1}[f_2] \\ f_1^{t_1+\Delta t_1} &= (\mathbf{B}_1 \circ \mathbf{P}_1 \circ \mathbf{C}_1)\,[f_1^{t_1}], \\ \\ f_2^{t_0} &= f_{\text{init},2} \\ f_2^{t_2+\Delta t_2} &= (\mathbf{B}_2 \circ \mathbf{P}_2 \circ \mathbf{C}_2[f_1])\,[f_2^{t_2}]\,, \end{aligned} \tag{2.15}$$

where 1 refers to the micro-scale and 2 to the macro-scale. The micro-scale model 1 is run until completion (i.e. until the final time T_1), then a single time step Δt_{h_2} is performed for the macro-scale model.

We can now compare an estimation of the execution time of the CxA model of Fig. 2.8 with that of using a single CA for the same system, as in the left part of Fig. 2.1. For the single CA the execution time would be $T_{CA} = k_{CA}\frac{T_2}{\Delta t_1}\left(\frac{L_2}{\Delta x_1}\right)^D$, which is (2.1) using the subscripts as introduced in Fig. 2.8. For the CxA, the execution time becomes

$$T_{\text{CxA}} = \frac{T_2}{\Delta t_2}\left(\frac{L_2}{\Delta x_2}\right)^D \left(k_2 + k_1 \frac{T_1}{\Delta t_1}\left(\frac{L_1}{\Delta x_1}\right)^D\right). \tag{2.16}$$

where k_{CA}, k_1 and k_2 are the CPU times to update one spatial cell for one time step, respectively for the full scale CA, the micro and the macro submodels.

Next one can compute a speedup, comparing the single scale CA formulation and the CxA formulation as $S = T_{\text{CA}}/T_{\text{CxA}}$. After some algebra we find

$$S = \left(k_{CA}\frac{\Delta t_2}{\Delta t_1}\left(\frac{\Delta x_2}{\Delta x_1}\right)^D\right) \Big/ \left(k_2 + k_1 \frac{T_1}{\Delta t_1}\left(\frac{L_1}{\Delta x_1}\right)^D\right). \tag{2.17}$$

Under the reasonable assumption that the execution time for a full micro scale simulation needs much more time than a single iteration of the macro scale model, i.e. when $k_1\frac{T_1}{\Delta t_1}\left(\frac{L_1}{\Delta x_1}\right)^D \gg k_2$, (2.17) reduces to $S = \frac{k_{CA}}{k_1}\frac{\Delta t_2}{T_1}\left(\frac{\Delta x_2}{L_1}\right)^D$. Note that $\frac{\Delta t_2}{T_1} > 1$ and $\frac{\Delta x_2}{L_1} > 1$, and can be interpreted as the distance on the SSM (Fig. 2.8). So, if the scale separation is large enough, the obtained speedups can be huge, principally rendering a CxA simulation feasible.

2.2.8 Scale-splitting error

The above arguments demonstrate the improvements in computational efficiency offered by the CxA formulation. On the other hand, replacing the original multi-scale model with many coupled single-scale algorithms, we face a partial loss of precision. A possible measure of this lowering in accuracy can be obtained considering the difference in the numerical results of the original $\mathcal{C}_h$ and the Complex Automaton CxA_H, which we call *scale-splitting error*

This error is measured according to the observables, i.e. the quantity of interest, formally resulting from the observable operators:

$$E^{\mathcal{C}_h \to \mathrm{CxA}} = \|\mathbf{O}(f_h) - \mathbf{O}_{\mathrm{CxA}}(f_H)\| \tag{2.18}$$

in an opportune norm. The scale-splitting error has a direct interpretation in terms of accuracy. In fact, calling $E^{\mathrm{CxA,EX}}$ the absolute error of the CxA model with respect to an exact reference solution, and $E^{\mathcal{C}_h,\mathrm{EX}}$ the error of the model itself, we have

$$\left\|E^{\mathrm{CxA,EX}}\right\| \leq \left\|E^{\mathcal{C}_h,\mathrm{EX}}\right\| + \left\|E^{\mathcal{C}_h \to \mathrm{CxA}}\right\|. \tag{2.19}$$

If we heuristically assume that the original *fine-scale* algorithm has a high accuracy, the scale splitting error is a measure of the error of the CxA model.

In general, a detailed and rigorous investigation of the scale-splitting error requires a good base knowledge of the single scale CA and of the full multi-scale algorithm. Case by case, error estimates can be derived using the properties of the algorithms, the operators involved in the update rule and in the coarse-graining procedure. An example of error investigation using the formalism for a simple CxA model can be found in section 2.3.

2.3 Example: Reaction Diffusion

Let us consider a reaction-diffusion process for a concentration field $\rho = \rho(t, x)$ described by the equation

$$\begin{aligned} &\partial_t \rho = d\partial_{xx}\rho + \kappa(\rho_\lambda - \rho),\ t \in (0, T_{end}],\ x \in (0, L] \\ &\rho(0, x) = \rho_0(x) \end{aligned} \tag{2.20}$$

with periodic boundary conditions in x, ρ_0 being the initial condition and $\rho_\lambda(x)$ a given function. To consider a multi-scale model, we assume the reaction to be characterized by a typical time scale faster than the diffusion.

Numerically, problem (2.20) can be solved employing a lattice Boltzmann method (LBM) (see for example [1, 41, 42] and the references therein), discretizing the space interval with a regular grid $\mathcal{G}_h = \{0, \ldots, N_x - 1\}$ of step size $\Delta x_h = h$ and associating each node $j \in \mathcal{G}_h$ with two variables, f_1 and f_{-1} representing the density of probabilities of populations traveling with discrete velocities $c_i \in \{-1, 1\}$. The collision+propagation update has the form

$$f_i^{t_n+\Delta t}(j + c_i) = f_i^{t_n}(j) + \frac{1}{\tau}\left(\frac{\hat{\rho}^{t_n}}{2} - f_i^{t_n}(j)\right) + \Delta t_h \frac{1}{2} R(\hat{\rho}^{t_n}(j)). \tag{2.21}$$

here $R(\hat{\rho}(j)) = \kappa(\rho_\lambda(j) - \hat{\rho}(j))$, and $\hat{\rho} = \rho(f) = f_1 + f_{-1}$ is the numerical solution for the concentration field. The time step is related to the grid size according to

$$\frac{\Delta t_h}{\Delta x_h^2} = \text{const.} \quad \forall h, \tag{2.22}$$

and the parameter τ is chosen according to the diffusion constant in (2.20) (see [42, 1])

$$\tau = \frac{1}{2} + d\frac{\Delta t_h}{\Delta x_h^2}. \tag{2.23}$$

Equivalently, we can rewrite (2.21) in the form [43]

$$f_h^{t_{n+1}} = P_h(I_h + \Omega_{D_h}(\tau))(I_h + \Omega_{R_h}) f_h^{t_n} = \Phi_h f_h^{t_n}, \tag{2.24}$$

highlighting the scale h and omitting the subscript i. The update $\Phi_h = P_h(I_h + \Omega_{D_h}(\tau))(I_h + \Omega_{R_h})$, has been decomposed into a diffusion part and a reaction part.

The space of states is the set $\mathbb{F}_h = \left\{\phi : \mathcal{G}_h \to \mathbb{R}^{2N_x}\right\}$, of the real functions defined on the grid $\mathcal{G}_h$. The subscript h for the operators denotes functions acting from $\mathbb{F}_h$ to itself. In detail, I_h is simply the identity on $\mathbb{F}_h$, P_h acts on a grid function shifting the value on the grid according to $\mathbf{c}_i$

$$\left(P_h f_h\right)_i (j) = f_{i,h}(j - c_i),$$

while Ω_{D_h} and Ω_{R_h} are the operations defined in the right hand side of (2.21):

$$\left(\Omega_{D_h} f_h\right)_i = \frac{1}{\tau}(f_i^{eq}(\rho(f_h)) - f_{i,h}), \quad \left(\Omega_{R_h} f_h\right)_i (j) = h^2 \frac{1}{2} R(\rho(f_h))$$

The SSM for this example is shown in Fig. 2.9. To define the CxA, we set $\Delta t_R = \Delta t_h = h^2$ for the reaction and $\Delta t_D = Mh^2$ for the diffusion. Focusing on the case shown in Fig. 2.9b, the reaction is run up to a time T_R, then re-initialized after a diffusion time step. If $T_R = \Delta t_D$, the two processes are not completely separated. Fig. 2.9c sketches the case when reaction leads very quickly to an equilibrium state in a typical time which is even smaller than the discrete time step of the diffusion.

We focus on the case of time-coarsening, i.e. choosing

$$\Delta x_D = \Delta x_R = h, \ \Delta t_D = M \Delta t_R = Mh^2 \,. \tag{2.25}$$

Introducing reaction and diffusion operators $\mathcal{R}_s$, $\mathcal{D}_s$, where $s = R, D$ specifies the dependence of the discrete operators on the space-time discretization of reaction and (resp.) diffusion, the evolution of the system can be described with the state variable $f_H = (f_R, f_D)$, whose components are updated according to

$$\begin{array}{cc} (\mathrm{CA}_R) & (\mathrm{CA}_D) \\ f_R|_{t_0 = t_D} = f_D^{t_D}, & f_D^0 = f_D^{init}(\rho_0), \\ f_R^{t_R + \Delta t_R} = \mathcal{R}_R f_R^{t_R} & f_D^{t_D + \Delta t_D} = \mathcal{D}_D f_D^{t_D + M\Delta t_R} \,. \end{array} \tag{2.26}$$

Equation (2.26) expresses that the algorithm CA_R, which is coupled to CA_D through the initial condition (by setting at the initial time $t_0 = t_D$ (equal to a certain time of CA_D) the initial condition equal to the one obtained from CA_D,

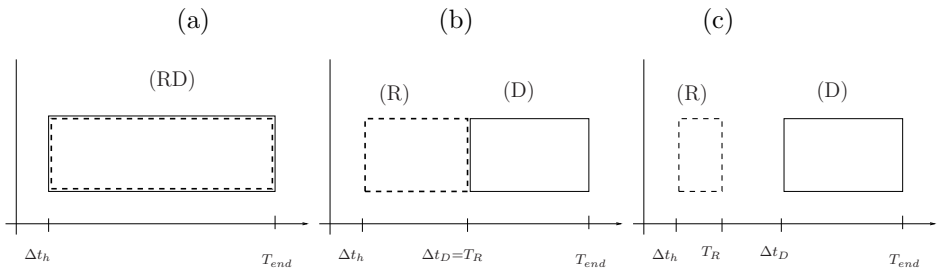


Figure 2.9: SSM for the reaction-diffusion LBM. In **(a)** reaction (dashed line) and diffusion (solid line) are considered as a single multi-scale algorithm. In **(b)** we assume to use different schemes, where the diffusion time step Δt_D is larger than the original Δt_h. **(c)** represents the situation where the two processes are time separated, with a very fast reaction yielding an equilibrium state in a time $T_R \ll \Delta t_D$.

and evolves for M steps according to an update rule depending only on the reaction process. On the right, the diffusion part CA_D is coupled to the reaction through the collision operator, since the new state of f_D is locally computed starting from the output state of CA_R. With $f_D^{init}(\rho_0)$ we denoted the original initial condition, function of the initial concentration in (2.20).

In this case, the observable is represented by the concentration ρ, obtained from the numerical solution by a simple average over the particle distributions.

According to the definition introduced in 2.2.8, we define the scale-splitting error at time iteration t_N as

$$E(t_N) = \left\| \rho\left(f_h^{t_N}\right) - \rho\left(f_{D,h}^{t_N}\right) \right\| \tag{2.27}$$

i.e. the difference between $\rho(f_h)$, the solution of the fine-grid algorithm (2.24) and the output of the CxA model (2.26) evaluated at the same physical time. Taking $f_{D,h}$ in the above formula corresponds to evaluating the error after both reaction and diffusion have been executed. Note that both f_h and $f_{D,h}$ belong to the same evolution space, since the grid spacing are equal.

Following section 2.2.8 we now consider the scale-splitting error $E(M)$ resulting from using a diffusion time step Δt_D M times larger than the reaction time step Δt_R. The reference solution is here the solution obtained when both reaction and diffusion act at the smallest time scale, i.e. when $M = 1$. To estimate $E(M)$ we consider M reaction steps at scale h (defined by Δt_R) followed by one diffusion step at the coarser scale h' (defined by $\Delta t_D = M\Delta t_R$) and we compare the results with M reaction-diffusion steps both at the fine scale h. In terms of the reaction and diffusion operators, $E(M)$ can be expressed as

$$\begin{aligned} E(M) &= \left\| (\mathcal{D}_h \mathcal{R}_h)^M - \mathcal{D}'_h \mathcal{R}_h^M \right\| \\ &\leq \left\| (\mathcal{D}_h \mathcal{R}_h)^M - \mathcal{D}_h^M \mathcal{R}_h^M \right\| + \left\| [\mathcal{D}_h^M - \mathcal{D}'_h] \mathcal{R}_h^M \right\| \\ &= E_1(M) + E_2(M) \end{aligned} \tag{2.28}$$

Contribution E_1 can be computed from the commutator $[\mathcal{D}_h \mathcal{R}_h - \mathcal{R}_h \mathcal{D}_h]$ and E_2 follows from the time coarse-graining of the original LB model. After some calcu-

lations we obtain (see [44])

$$E(M) \leq \mathcal{O}(M^2\kappa) + \mathcal{O}(M^2D^3) \tag{2.29}$$

$$\begin{aligned} E(t_N) &= \left\| \rho \left(\left(\mathcal{D}_h\mathcal{R}_h\right)^M f_h^{t_N - M\Delta t_h} - \mathcal{D}_{D,h}\mathcal{R}_{D,h}^M f_{D,h}^{t_N-\Delta t_D} \right) \right\| \leq \\ &\leq \left\| \rho \left(\left(\mathcal{D}_h\mathcal{R}_h\right)^M - \mathcal{D}_{D,h}\mathcal{R}_{D,h}^M \right) f_{D,h}^{t_N-\Delta t_D} \right\| \\ &+ C(h,\mathcal{D},\mathcal{R}) E(t_N - \Delta t_D)\,, \end{aligned} \tag{2.30}$$

$$\begin{aligned} E(t_N) &\leq \left\| \rho \left(\left(\mathcal{D}_h\mathcal{R}_h\right)^M - \mathcal{D}_{D,h}\mathcal{R}_{D,h}^M \right) \right\| \leq \\ &\leq \left\| \rho \left(\left(\mathcal{D}_h\mathcal{R}_h\right)^M - \mathcal{D}_h^M\mathcal{R}_h^M \right) \right\| + \left\| \rho \left(\mathcal{D}_h^M\mathcal{R}_h^M - \mathcal{D}_{D,h}\mathcal{R}_{D,h}^M \right) \right\|. \end{aligned} \tag{2.31}$$

The first contribution depends on the difference $\left(\mathcal{D}_h\mathcal{R}_h\right)^M - \mathcal{D}_h^M\mathcal{R}_h^M$, which can be estimated as a function of

$$[\mathcal{D}_h, \mathcal{R}_h] = \mathcal{D}_h\mathcal{R}_h - \mathcal{R}_h\mathcal{D}_h,$$

i.e. the commutator of the operators $\mathcal{R}_h$ and $\mathcal{D}_h$. It can be shown that [44]

$$[\mathcal{D}_h, \mathcal{R}_h](f_h) = O\left(h^3\kappa\partial_x(\rho(f_h) - \rho_\lambda)\right). \tag{2.32}$$

For the second contribution, deriving from the coarsening of the diffusion part of the original lattice Boltzmann algorithm, we obtain

$$\rho\left(\mathcal{D}_h^M - \mathcal{D}_{D,h})\right) f_{D,h} = O(M^2D^3h^2)\,. \tag{2.33}$$

Furthermore, we observe that $\mathcal{R}_h^M = \mathcal{R}_{D,h}^M$.

In conclusion, a qualitative estimate of the scale splitting error in terms of M is given by

$$E^{\mathcal{C}_h \to \mathrm{CxA}} = O\left(M^2\kappa\right) + O\left(M^2D^3\right). \tag{2.34}$$

Numerical Validation

We consider the problem

$$\begin{aligned} &\partial_t\rho = d\partial_{xx}\rho - \kappa(\rho - \sin(\lambda x)),\ t \in (0, T_{end}],\ x \in (0,1] \\ &\rho(0,x) = \rho_0(x) \end{aligned} \tag{2.35}$$

with $\frac{\lambda}{2\pi} \in \mathbb{Z}$, and periodic boundary conditions in x-direction.

By selecting different values of the parameters regulating (2.35) we can tune the relevance of different time scales. Additionally, we introduce the non dimensional parameter

$$\sigma = \frac{\kappa}{\lambda^2 d}$$

to "measure" the scale separation of the simulation. In the numerical tests, we run both the original fine scale LBM and the CxA model, measuring explicitly the scale-splitting error as the difference in the resulting concentrations. Fig. 2.10

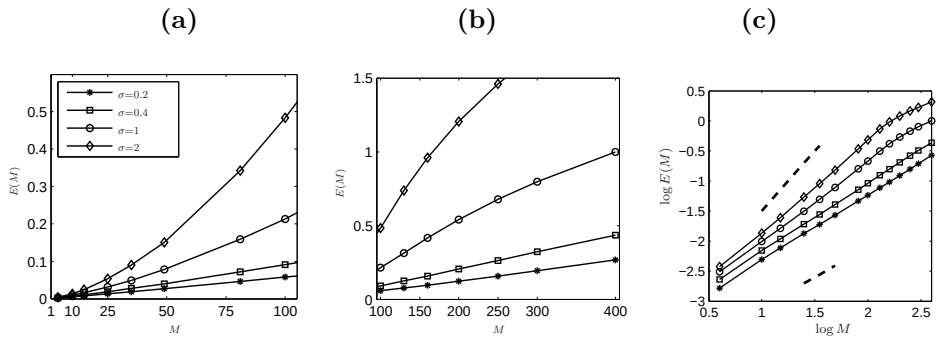


Figure 2.10: Scale-splitting error as a function of M for a time-coarsened CxA. The different curves represent different values of σ. Simulation parameters: $h = 0.02$, $\lambda = 4\pi$, $\kappa = 10$, $d \in \{0.05, 0.1, 0.25, 0.5\}$. **(a)**: $1 < M < 100$. **(b)**: $M > 100$. The size of the scale-splitting error becomes relatively large, except for the case $\sigma = 0.2$. **(c)**: Order plot (fig. (a)-(b) in double logarithmic scale) of maximum scale-splitting error versus M. The dashed lines of slope 1 (bottom) and 2 (top) indicate that $E \sim M^{\alpha}$, with $1 < \alpha < 2$. For the largest M a leveling off towards a maximum error can be observed.

shows the results of scale-splitting error for different values of M. The order plot in fig. 2.10c confirms estimate (2.34).

Results of a further test to link together scale separation and scale-splitting error are shown in Fig. 2.11a-b. Namely, for each simulation drawn in Fig. 2.10, we select the first M such that the scale splitting error lies below a certain prefixed threshold error $\bar{E}(h, H)$. These values M_{th} are plotted then as function of σ, validating the idea that better scale separation allows more efficient CxA formulations.

Detailed analysis and investigation of this example can be found in [44].

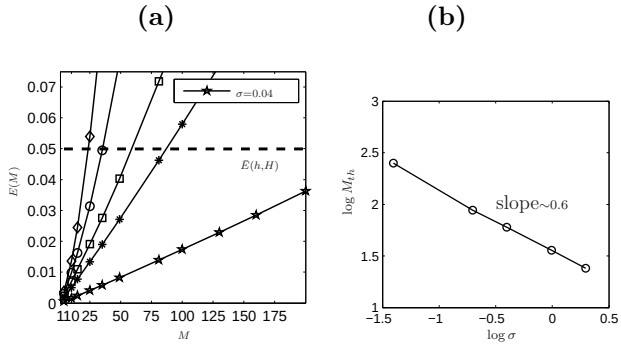


Figure 2.11: **(a)**: Zoom of the previous fig. 2.10a, including a threshold error $\bar{E}(h, H) = 0.05$ (results with $\sigma = 0.04$ are also shown). **(b)**: Values of M_{th} such that the scale-splitting error equates a threshold error $\bar{E}(h, H)$, versus the measure of scale separation $\sigma = \kappa \left(\lambda^2 D\right)^{-1}$ (in double logarithmic scale).

We can also compute the speedup resulting from the above time-splitting. Let us call a and b the CPU times of one iteration of respectively the reaction and

the diffusion processes. If we run the full system at the finer scale Δt_R for a time T, the total CPU time will be proportional to $(a + b)(T/\delta t_R)$. With the time-splitting method, the CPU time reduces to $(Ma + b)T/(M\Delta t_R)$ and the speedup is $(a + b)/(a + b/M)$. For large M, the speedup tends to $1 + b/a$. This might not be a very big gain, unless $a \ll b$. However, if we would have coarse grained the spatial scale for the diffusion processes, we would get a more interesting speedup value.

2.4 Conclusion

This chapter described a possible approach towards multi-scale modeling and simulation using Cellular Automata. The concept of Complex Automata should allow the modeling of a large range of multi-scale systems, and the related Complex Automata simulation software provides a framework to quickly develop Complex Automata simulations. The ideas behind Complex Automata have a broader significance than Cellular Automata modeling alone and, in the near future, we will explore the possibility to enlarge the CxA idea to other modeling paradigms. Moreover, we are developing a growing set of CxA models and simulations. This thesis will focus on two of them in the following chapters. Sec. 3 will present a model of a challenging multi-science problem in the field of cardiovascular diseases, namely that of in-stent restenosis. The subsequent chapter 4 will describe a multi-scale model for the simulation of macroscopic suspension flow based on microdynamics. Both of them will be described in the spirit of the concept of Complex Automata which proves very helpful in classifying and constructing single-scale models and their coupling.