6
Related Work

In order to build self-organizing systems, we need to understand how to model, design, and validate those systems. Before proposing new methods, architectures, or tools, we need to look at the state of the art and identify ways of improving software engineering best practices. On the other hand, while there is plenty of literature about the analysis of self-organizing and emergent mechanisms, the interest in engineering aspects grew only recently.

In this chapter we only summarize those works about self-organizing systems that are strictly related to methodological aspects proposed in this thesis. This includes modeling and designing self-organizing systems, design patterns, architectures for self-organizing systems, and validation or verification methods applied to those systems.

6.1 Modeling and Designing Self-Organizing Systems

(a) Design Support

There are four relevant references for the design support of modeling and designing self-organizing systems. They are presented in this section.

Van Parunack: A Design Guide

Van Parunak and Bruckner proposed a design guide for swarming systems engineering [Parunak 2004] consisting of ten design principles.

The four first are derived from coupled processes which are defined as interactions through information's exchange between agents or processes:

1. Use a distributed environment;
2. Use an active environment;
3. Keep agents small;
4. Map agents to Entities, not Functions.

The next three design principles are derived from autocatalysis. Parunack recalls the concept of autocatalysis from chemistry. In autocatalysis, a product of a reaction serves as a catalyst (substance that facilitates a chemical reaction without being permanently changed) for that same reaction. And an autocatalytic set is a set of reactions that are not individually autocatalytic, but whose products catalyze one another. The result is that the behaviors of the reactions in the set are correlated with one another. He then extended this concept from chemistry to any system of interacting processes, such as a multi-agent system. A set of agents has autocatalytic potential if in some regions of their joint state space, their interaction causes increase in order. In that region of state space, they are autocatalytic. The design principles are:

1. Think Flows rather than Transitions;
2. Boost and Bound, which means, design mechanisms for both positive and negative feedback loops;
3. Diversify agents to keep flows going.

And the final three last design principles are derived from functional adjustment, i.e, the desired guarantees that are useful to the system's stakeholders monitor and evaluate.

1. Generate behavioral diversity, i.e, the agents should explore the behavioral space as widely as possible;
2. Give agents access to a fitness measure
3. Provide a mechanism for selecting among alternative behaviors

Even if most of the design principles in swarming systems have demonstrated their effectiveness as an alternative model of cognition and have been applied to number of applications, we show in this thesis that some of those principles should not be taken for granted or at least in isolation. For instance, we presented a notational approach that indicates we should design thinking in flows *combined* with transitions to achieve the coordination that enables self-organization. Regarding Parunak's design principles, no associated tools exist. No modeling approach exists. No general validation or verification method is proposed.

De Wolf: The Customised Unified Process and Information Flows through UML 2.0 Activity Diagram

De Wolf [De Wolf 2007b] has defined a full iterative life-cycle methodology based on the Unified Process customized to explicitly focus on engineering macroscopic behavior of self-organizing systems.

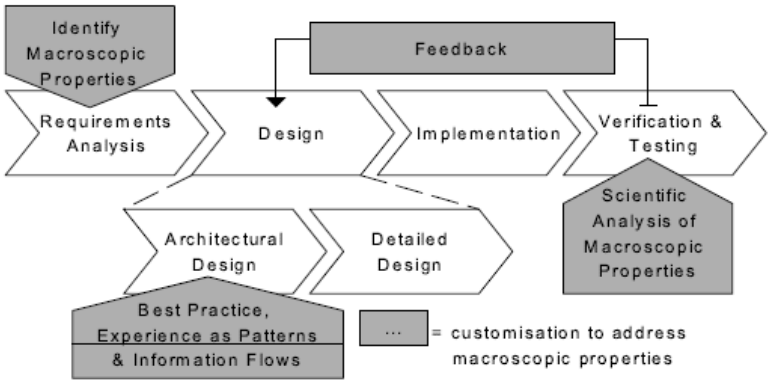

Figure 6.1: The Customised Unified Process from [De Wolf 2007]

Requirement Analysis phase The problem is structured into functional and non-functional requirements, using techniques such as use cases, feature lists and a domain model that reflects the problem domain. Macroscopic requirements (at the global level) are identified.

Design phase This phase is split into Architectural Design and Detailed Design addressing microscopic issues. Information Flow (a design abstraction) traverses the system and forms feedback loops. Locality is that limited part of the system for which the information located there is directly accessible to the entity [De Wolf 2007b]. Information flows are enabled by decentralized coordination mechanisms, defined by provided design patterns.

Implementation phase The design is realized by using a specific language. When implementing, the programmer focuses on the microscopic level of the system.

Testing and Verification phase Agent-based simulations are combined with numerical analysis algorithms for dynamical systems verification at macro-level.

Information Flow as a Design Abstraction

As a design abstraction, in particular, De Wolf proposed to use UML 2.0

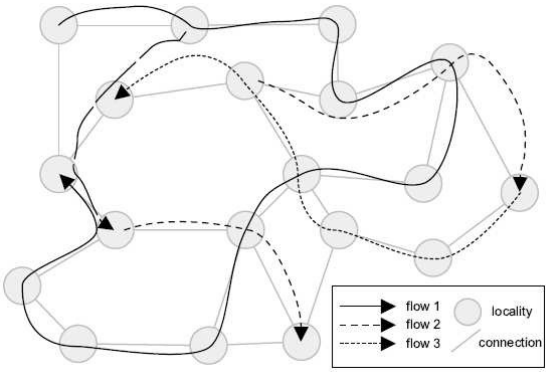

Figure 6.2: Information flows, from [De Wolf 2007]

Activity Diagram for designing Information Flows. Activity diagrams are used to determine when a certain behavior starts and what its inputs are.

Behavior models, such as activity diagrams, determine when behaviors start and what their inputs are. Activity models focus on the sequence, conditions, inputs and outputs for invoking behaviors. UML 2 activity diagrams define the routing of control and data through a graph of nodes connected by edges. Each node and edge defines when control and data values move through it. These movement rules can be combined to predict the behavior of the entire graph or system. The rules for control and data movement are only intended to predict runtime effects, that is, when behaviors will start and with what inputs.

Locality w.r.t. a system entity (e.g. an agent, network nodes, etc.) is defined as that limited part of the system for which the information located there is directly accessible to the entity, i.e. no need to communicate with other entities. A locality needs to be defined for each system entity.

UML 2 activity diagrams serve the need to indicate which actions should wait on which information flow(s) before the action can execute. The notion of a partition serves the need to represent a "locality" as something in which actions are performed and information flows can enter or leave.

There are notational difference compared with standard UML 2. Data flow edges are shown as dashed lines instead of solid lines to clearly emphasis them as being information flows. The semantics remain the same. The thick gray arrows shown later are not part of the diagram but merely an annotation to indicate the loops present in the information flows.

As this design approach is the closest to our approach, it is important to understand why we proposed a novel approach. We found two main problems with the design abstractions for this type of system. First, a very complex system would require very complex information flows at the macro scale.

De Wolf does not propose any way to modularize, compose or even reuse the models, which makes the design approach hard to understand. Also the macroscopic information flows are only based on information, a decision made upon Parunak's design principles. In this thesis we show that when combining information with states, we can achieve a design that helps on the state evaluation for validation tasks.

Gardelli: Agents and Artifacts

Luca Gardelli [Gardelli 2008] proposed a meta-model and a methodological approach for engineering self-organizing multi-agent systems. The meta-model is based on stigmergy, i.e, indirect communication where individual parts communicate with one another only by modifying their local environment. Artifacts are first-class entities representing the environment which mediates agent interaction and enables emergent coordination: as such, they encapsulate and enact the stigmergic mechanisms (diffusion, aggregation, selection, etc.) and the shared knowledge upon which emergent coordination processes are based.

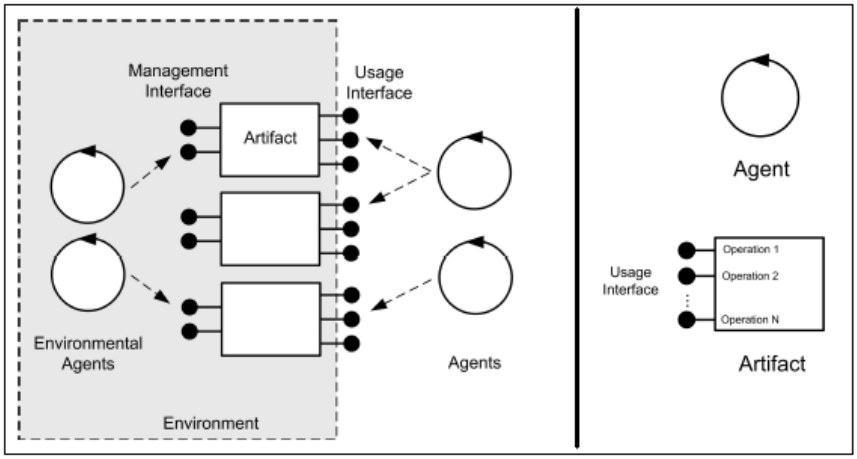

Figure 6.3: Architectural pattern featuring environmental agents as artefact administrators, from [Gardelli 2008]

Gardelli proposed a Simulation Driven Approach (SDA) situated between the analysis and the design phase, as an Early design phase. The models are analyzed using stochastic simulations (stochastic Pi-calculus [Milner et al. 1992, Priami 1995] and the Stochastic Pi-Machine (SPiM) [Phillips 2007]; stochastic simulation framework developed on top of the Maude tool [Maude 2007]) and model checking [Kwiatkowska et al. 2007] using the Probabilistic Symbolic Model Checker (PRISM) [PRISM 2007], with the goal to describe the desired

agent behavior and a set of working parameters (Simulation phase). These are calibrated through a tuning process (Tuning phase).

That said, Gardelli did not provide any dynamic visual modeling approach to the design of self-organizing systems, or in particular, information flows.

ADELFE

ADELFE[1] is an AOSE methodology that provides support for self-organization [Bernon 2004, Picard 2004], more precisely for the Adaptive MAS theory (AMAS). In the AMAS theory agents pursue their local goal while trying to keep cooperative relations with other agents embedded in the system. ADELFE extends the Rational Unified Process (RUP) and uses UML and AUML notation and a software tool to provide graphic design capabilities.

During cooperation an agent tries to anticipate problems, detect cooperation failures (Non Cooperative Situations, NCS) and repair them. An AMAS agent is autonomous and unaware of the global function of the system; it can detect NCSs and acts to return to a cooperative state. Adelfe is divided into six phases or Work Definitions (Figure 2); each phase consists of several activities (A).

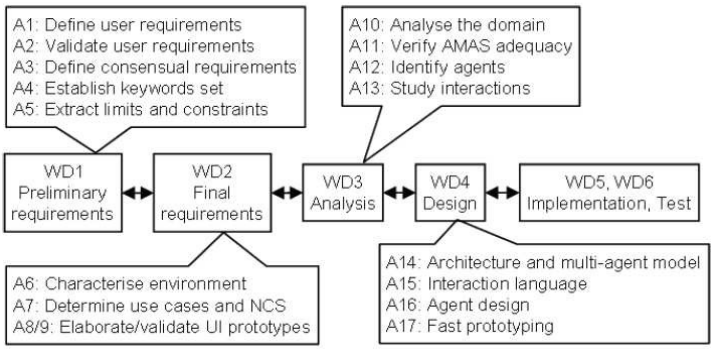

Figure 6.4: The ADELFE methodology [Picard 2004]

It is worth nothing to say that as ADELFE uses original UML and AUML notations, there is no support to design feedback loops or information flows. And although ADELFE eventually identifies the need for verification of the adopted model, there is no evidence of how to perform it pragmatically: recently, the authors evaluated the integration of simulation techniques in their methodology [Bernon 2007] to better support the design stage and the analysis of agents' behaviors. Prior to that investigation, ADELFE focused more on

[1] `http://www.irit.fr/ADELFE/`

aspects related to design-time, such as entities and responsibilities, rather than the actual system dynamics.

(b) Design Patterns

When engineering a self-organizing emergent solution, the problem-solving power mainly resides in the interactions and coordination between agents instead of in intelligent reasoning of individual agents [ref]. Documented design practices are essential to provide developers and maintainers of complex self-organizing systems with proper design guidance and reuse. They also facilitate quality assurance processes, such as software verification and testing.

In this context, design patterns were promoted in the last decade with the object-oriented paradigm [Gamma et al. 1995]. Concerning multi-agent systems, we can find several contributions, but little attention has been payed to self-organizing systems. It is worth noting that there is plenty of literature about applications of self-organizing systems inspired by biological systems: unfortunately, despite the great insights presented in works, few of them encode successful solutions in a reusable form, and takes a lot of experience for a computer scientist to extract the actual strategy.

To our knowledge, there exists only three works in that specific context [Babaoglu 2006, Gardelli 2008, De Wolf 2007]. And, as we used some of these patterns to define a pattern language that shows how to use the notational support proposed in chapter 3, we briefly present those patterns in this section according to each author.

Ozalp Babaoglu et. al

In [Babaoglu 2006] the authors successfully establish a mapping between existing natural system and its counterpart in computer science, easing the designer task: in particular, the authors goal is to facilitate the adoption of biology-inspired ideas in distributed systems engineering.

They present the patterns by describing the following attributes: *name, context, problem, solution, example*, and finally, *design rationale*. Out of these attributes, they describe the *context* attribute separately because it is common to all patterns. The attribute *design rationale* explains where the pattern came from and why it works. In particular, it involves the discussion of the biological manifestations of the pattern, and a brief description of the insight why they function efficiently.

The design patterns are:

(i) Plain Diffusion Inspired from equalization of concentration; Receive/emit diffusive material

(ii) Replication Inspired from epidemic spreading/proliferation processes; Create new material. Eg: Virus spreading.

(iii) Stigmergy Inspired from social systems (insects, ants, birds); Global coordination from "properly designed" local interactions;

(iv) Chemotaxis Inspired from cells or other organisms that direct their movements according to the concentration gradients of one or more chemicals (signals) in the environment; To apply chemotaxis, we need to have some sort of diffusion present in the system, that generates gradients.

(v) Reaction-Diffusion Spontaneous diffusion with addition/removal of material; Used in pattern formation / developmental processes.

The authors present first the simpler ones, i.e. Plain Diffusion, Replication and Stigmergy, and then discuss the composite ones, i.e. Chemotaxis and Reaction-Diffusion.

Luca Gardelli

Gardelli discusses a pattern scheme for self-organizing MAS and evaluate a few behavioral design patterns. The pattern scheme extends the known ones described in [Lacroix 2003] and introduces novel items:

- **Feedback loop** Describes the processes or actions involved in the establishment of a feedback loop
- **Locality** Requirements in terms of spatial topology or action-perception ranges: if the environment has a notion of continuous space, perception range is specified as a float value; if the environment has a graph topology, ranges are specified as the number of hops.

The first pattern defined is the **Collective Sort**. From any arbitrary initial state, the goal of Collective Sorting is to group together similar information in the same node, while separating different kinds of information.

The other three patterns are derived from *Stigmergy*. In social insects colonies, coordination is often achieved by the use of chemical substances, usually in the form of pheromones: pheromones act as markers for specific activities, e.g. food foraging. Specifically, these substances are regulated by environmental processes called **Aggregation, Diffusion** in space and **Evaporation**

over time: each process has its own relevance, hence they were analyzed as a separate pattern.

Although the catalog of patterns provides a structured schema, at the solution attribute level it only describes the phenomena. Gardelli does not propose a way to implement the dynamics in reusable software components as this thesis illustrated in a preliminary work.

Tom De Wolf

In [De Wolf 2007a], De Wolf provides a thorough description of two complex patterns about decentralized coordination mechanisms, namely, Gradient Fields and Market-based Control. Concerning the pattern schema, the authors rely on an existing pattern schema non-specific for self-organizing systems: it was defined from [Meszaros 1996] and, for instance, the Gang of Four patterns book [Gamma et al. 1995] uses a similar format: *Pattern Name/Also Known As, Context/Applicability, Problem/Intent, Forces, Solution, Related Mechanisms/Patterns, Examples/Known Uses*.

In particular, the *Solution* is a description of how the problem is solved, and is divided in *Inspiration, Conceptual Description, Parameter Tuning*, mechanisms for *Infrastructure*, and *Characteristics*. The *Conceptual Descriptions* are explained with the illustration of UML class diagram.

Gradient Fields Spatial, contextual, and coordination information is automatically and instantaneously propagated by the environment as computational fields. Agents simply follow the "waveform" of these fields to achieve the coordination task, i.e. no explicit exploration.

Market-based Control A virtual market where resource users sell and buy resource usage with virtual currency. The price evolves according to the market dynamics and indicates a high (high price) or low (low price) demand. This information is used by agents to decide on using the resource or not. Economic market theory states that the prices converge to a stable equilibrium.

The patterns are described at the conceptual level, hence there is no identification of actual software components although UML entities and class relationships are presented. Concerning the pattern granularity, the patterns described in [De Wolf 2007a] are quite coarse and it is possible to identify owner patterns, organizing the patterns in a hierarchy.

6.2 Architectures and Middlewares for Self-Organizing Systems

The need for architectures and middleware to build self-organizing systems is absolute. In the context of self-organization, Parunak [Parunak 1997] does not provide an architecture or middleware, but he describes several optimization algorithms for which an environment is needed. Omicini proposed TuCson [Omicini 2001], a middleware that allows the coordination of parallel processes (agents) through tuplespaces which can be seen as early (and ongoing) work to provide an environment wherein agents can interact. Mamei et al. [Mamei 2004] provide a middleware environment, called TOTA, which allows agents to coordinate their movements in a mobile network. Both TOTA and TuCson do not provide architectural simulation, multi-environment and validation features. And, finally, Weyns [Weyns 2006] proposed the first reference architecture for situated multi-agent systems that includes a set of self-organizing features and considers the environment as a first-class abstraction.

(a) TOTA: Tuples On The Air

TOTA [Mamei 2004] enables developers and programmers to easily configure any specific type of stigmergic coordination, whether pheromone-based or field-based, and to have their deployment and execution properly supported.

TOTA relies on a simple Java API for injecting tuple-based information in a network, have it propagate and/or evaporate accordingly to application-specific policies, and have it locally sensed by application agents. TOTA tuples are not associated with a specific node (or with a specific data space) of the network.TOTA tuples form a sort of spatially distributed data structure able to express properties of the network environment that can be used to acquire contextual information about the environment itself, and to support the mechanisms via which stigmergic interactions can take place. To support this idea, TOTA considers the presence of a peer-to-peer network of nodes, each node running a local version of the TOTA middleware.

(b) TuCson

TuCson [Omicini 2001] supports agent coordination by means of first class runtime coordination abstractions, the so called coordination artifacts. Coordination artifacts are organization resources shared and (concurrently) accessed/exploited by agents. TuCson is based on the Tuple space model [Lacroix 2003], and adds two more abstractions: Tuple Centers, and agents

access tuple centers through an agent coordination context (ACC).

Tuple centers specialize and extend the basic tuple space model by (1) using logic tuples: tuples are first order logic terms (such as Prolog terms); (2) while the behavior of a tuple space in response to communication events is fixed, the behavior of a tuple center can be tailored to the application needs by defining a set of specification tuples expressed in the ReSpecT language, which define how a tuple center should react to incoming / outgoing communication events.

The ACC is a runtime and statefull interface released dynamically to the agent by the infrastructure. The ACC interface is composed of the operations that an an agent can use to interact with tuple centers. So, in order to communicate and coordinate with TuCSoN, first an agent must obtain an ACC, then it can exploit the ACC by invoking its operations to interact with tuple centers.

(c) A Reference Architecture for Situated Multi-Agent Systems

Weyns proposed an architecture that promotes the environment to a first-class abstraction. The reference architecture provides a set of mechanisms for architectural design, including: environment infrastructure for perception, action, and communication; laws that constrain the activity of agents; dynamics in the environment [Weyns et al. 2005, Weyns 2005a, Weyns 2005b]; virtual environment [Weyns et al. 2005b]; selective perception [Weyns et al. 2003]; advanced action selection mechanisms with roles and situated commitments [Weyns et al. 2004a, Steegmans et al. 2004]; and protocol-based communication [Weyns et al. 2004b].

Basically, the reference architecture maps the functionalities of a situated multi-agent system onto a system decomposition, i.e. software elements and relationships among the elements. The reference architecture is presented by means of four views that describe the architecture from different perspectives. Views are presented as a number of view packets. A view packet focuses on a particular part of the reference architecture. The reference architecture provides three view packets of the module decomposition view. Modules in the module decomposition view include a description of the interfaces of the module that documents how the module is used in combination with other modules. The interface description distinguishes between provided and required interfaces. A provided interface specifies what functionality the module offers and needs to other modules. A required interface defines constraints of a module in terms of the services a module requires to provide its functionality.

The environment model consists of a set of modules with flows between the modules. The modules represent the core functionalities of the environment. The model consists of two main modules, the deployment context and the application environment. The deployment context can be the hardware and software and external resources with which the multi-agent system interacts and the application environment is the environment that has to be designed for an application, i.e. the functionality on top of the deployment context.

The application environment enables agents to access the external resources, shielding low-level details. Additionally, the application environment may provide a coordination infrastructure on top of the deployment context, which enables agents to coordinate their behavior.

The state maintenance module has a central role in the application environment. It provides functionality to the other modules to access and update the state of the application environment. The state of the application environment typically includes an abstraction of the deployment context and possibly additional state. The environment state may also include agent-specific data, such as agents' identities and positions, and tags used for coordination purposes.

Finally, the agent model consists of four modules with flows between the modules. The four modules provide the required agent functionalities for the mechanisms of adaptivity, including selective perception, action selection with roles and situated commitments, and protocol-based communication.

The focus of this architecture is to support the design from the agent-environment models to the deployment level. It was not designed to support discrete simulations, and it does not exploit different environment structures, or validation support.

6.3 Validation and Verification of Self-Organizing Systems

The next subsections briefly describes related work in formal methods that address qualitative simulations, and a scientific numerical analysis method called "Equation-based free Analysis" that was exploited as a quantitative validation method by De Wolf.

(a) Formal Methods

Following a traditional rationale, if the system is modeled formally and the required macroscopic behavior is proved analytically, desired guarantees could be proved. However, there is a general consensus that constructing

a complete formal model and correctness proof of a complex interacting computing system is infeasible [Wegner 1997]. The problem is that interaction models are so powerful that they can be considered to be incomplete, in the mathematical sense. One cannot model all possible behavior of an interaction model; and, thus, formally proving correctness of interactive models such as self-organizing emergent systems is not merely difficult but, in general, impossible [De Wolf 2007].

On the other hand, Gardelli formally modeled three self-organizing applications and analyzed the system-wide behavior. At same time, d'Inverno formally modeled the stem cell behavior as follows.

Gardelli

Pi-Calculus for specifying and verifying the system

Gardelli integrated a formal model approach with configuration or parameter tuning to accomplish the verification of the macroscopic behavior of self-organizing multi-agent systems and thus proving their correctness. His formal approach was founded on Pi-Calculus [Milner et al. 1992, Priami 1995] – a process calculus able to describe concurrent computations whose configuration may change during the computation.

For instance, in one of the Gardelli approaches, if an agent is behaving differently from the average, especially for critical actions, one may decide to inspect the agent further or deny access to resources. To get this done, simulation parameters are set: initial number of agents, normal vs. abnormal agent ratio, normal and abnormal agent entering rate. Anomaly detection system parameters, such as the number and rate of inspections, are also adjustable. In particular, in this approach one is able to make some assumptions about the percentage of abnormal behaving agents, the rate of agents entering / leaving the system, and the detection rate.

d'Inverno: Stem Cell problem

d'Inverno et al. [d'Inverno 2005] have applied agent-based software engineering for the stem cell modeling and simulation. They produced formal and mutually consistent specifications of the leading of some predictive models of stem cell behavior within their agent framework. They have also produced simulations of these models. In their approach, each stem cell is implemented as an atomic agent. They modeled and simulated the stem cells in a dynamic 2D grid environment with the capabilities of division and differentiation (stem cells which have reached their cycle phase and which are surrounded by stem cells become differentiated). However, the stem cell behavior modeled use prob-

abilistic determinism. They also argue that by building a formal model using a specification language from software engineering (they used the language Z [Spivey 1988]), there are techniques to ensure that the simulation correctly implements the model.

(b) De Wolf: Equation-based free Analysis

Although the use of formal tools allows us to gain a deeper insight in emergence and self-organization, there is a general belief and proof that emergent systems cannot be specified formally [Wegner 1997], [De Wolf 2007a].

In this sense, De Wolf applied the equation-free approach [De Wolf 2007], first proposed by Kevrekidis [Kevrekidis 2004], as the verification technique. In scientific computing research, there exists a whole store of numerical analysis algorithms that support the analysis of the system dynamics and which have a mathematical foundation. Typically, these are applied to formal equation-based models. The "Equation-Free Macroscopic Analysis" approach supports the empirical application of these analysis algorithms without needing a formal equation-based model. In fact, the evolutionary equations are replaced by small simulations of the system evolution, considering some input parameters. This technique results in more valuable and advanced verification results compared to normal begin-to-end simulations. These results are supported by dynamical systems theory. Additionally, begin-to-end simulations can take an enormous, and sometimes unacceptable, time to finish. According to the author, this is the main reason why the "Equation-Free Macroscopic Analysis" is based on small and fast simulation steps. The analysis algorithms can be seen to steer the simulation process by iteratively deciding on which simulations are needed and generating appropriate initial conditions, parameter values, etc.

De Wolf's validation method is complementary to the validation method proposed in this thesis. The scientific numerical analysis algorithms could be encapsulated in the Manager in a way that they could be easily executed and re-initialized to different application domains.

6.4 Chapter Remarks

In this chapter we have presented the state of the art of engineering methods, design patters, and architectures for self-organizing systems that are somehow related to the work proposed in this thesis. We saw that although the main authors that focused on modeling, designing, simulating and validating/verifying (De Wolf and Gardelli) provide a complete simulation-based methodological approach, they do not provide reusable architectures for self-

organization. On the other hand, it is worth nothing to highlight that they provide conceptual architectural design patterns. Complementary, Weyns proposed a reference architecture for situated multi-agent systems which includes a set of self-organizing features and considers the environment as a first-class abstraction. The reference architecture was organized with the mapping of the functionalities of a situated multi-agent system onto a system decomposition, i.e. software elements and relationships among the elements. Finally, none of the architectures or middleware presented were designed to support discrete simulations and they do not exploit different environment structures. Therefore, linking with the multi-agent simulation toolkits that we studied in chapter 2, we can clearly observe the need and contribution of an integrated architecture which encompasses both simulations, environment structures and dynamics, coordination components, self-organizing and validation mechanisms support as it was proposed in this thesis.