7
Conclusions and Future Work

Self-organization and emergence are important aspects of decentralized distributed systems. Through the applicability of self-organizing mechanisms, distributed systems can have decentralized control and increase in robustness. Although self-organization is an old biology concept, it is not a mature computer science concept, and the community that investigates the particularities of its usage in computer systems is still quite small, and mostly associated to the Distributed Systems area.

We believe the widespread of building self-organizing emergent systems depends on software engineering techniques and this was the focus of the research presented in this thesis. Next section presents the main contributions of the research. Then section 7.2 identifies the different future research directions which this thesis implies. We present further details about the future directions on which there are some partial research already developed.

7.1 Contributions

The research presented in this thesis represents a step toward the advances on architectural design of self-organizing emergent systems. The contributions of this thesis are fivefold:

1. Self-Organizing Software Engineering
   We applied an agent-oriented technology to build self-organizing emergent systems. This technology provides design support, engineering method, a Simulation-based Self-Organizing Architecture (SSOA) and best practices as reuse and modularity. Regarding the design support, we have presented the Coordinated Statecharts concept that motivated the UML meta-model customization in order to provide behavioral features that help add semantics to the proposed modeling abstraction. Coordinated Statecharts can be used to design the Information Flow concept presented by De Wolf. In order to illustrate the use of the modeling abstraction, we have presented a base to develop a pattern language on top of self-organizing patterns that already exist in the literature. This

pattern language contributes to the state of the art in many ways: (i) providing a guide to use the patterns, (ii) providing reusable structures and dynamics, and finally (iii) providing a way of combining basic patterns into a complex one – the Gradient Fields. As a result, we believe that the self-organizing system designer have both a design abstraction complemented with a notational support that help with her design task. Regarding SSOA, this architecture provides an asset base engineers can draw from when developing self-organizing applications. The application of the engineering guidelines and simulation-based self-organizing architecture in three complex industrial applications is a contribution. In particular, we have applied the various mechanisms for architectural design of self-organizing systems to meet the functionality and satisfy the desired guarantees of the system. The insights derived from the architectural design of these applications have considerably contributed to the development of the proposed architecture.

2. Agent-Oriented Software Engineering
   In order to use an agent-oriented technology, a suitable notational support to this area was proposed, with the UML customizations, and also an autonomic validation architectural features for validating emergent behavior of multi-agent systems. Regardless self-organizing mechanisms, agents in multi-agent systems need to be coordinated in some way, and those customizations help on this task. Also, emergent behavior is an inherent multi-agent system characteristic, and the autonomic validation method helps on the monitoring and controlling of agents misbehavior.

3. Normative Multi-Agent Systems
   We investigated the notational support, engineering method and SSOA application to a normative multi-agent system. This is currently a recent and active area of research. Research on governance of multi-agent systems has mainly focused on representing or specifying norms in a precise manner [Oren 2008, Silva 2008, Paes 2005, Esteva 2003], monitoring for the violation of these norms [Meneguzzi 2008, Minsky 2000, Paes 2007], explaining the outcomes of the monitoring process [Meneguzzi 2008], enforcing behavior in a way that the agent cannot violate the norm [Paes 2007] and ensure fault-tolerance according to norms [Gatti 2007]. However, languages and tools for specifying norms do not by themselves provide understanding of the emergent behavior in a complex domain. Somewhat related approaches are found in [Lacroix 2008] where the authors focus on norms in simulation. They proposed to model behavioral

differentiation in such simulations as a violation of the norm, and illustrate the approach through the traffic simulation. In [Bou 2007] the authors study how traffic control strategies can be improved using a case-based reasoning approach that allows an electronic institution to self-configure its regulations. And in [Doniec 2006], the authors show how the introduction of non-normative behaviors improves the microscopic traffic simulation. By allowing agents to break some of the formal rules of the road, norms are implicitly taken into account in the agents' decision model. Our original contribution consists of the iterative refinement of contracts and desired guarantees analysis that can be done with the results of a simulation-based approach and contract model .We believe there is enough re-usability in the work described, taking Section 6.2.a as a general approach and the case study as a guiding example, for others to benefit from this work.

4. Autonomic Computing
   The main contribution for this area consists of the autonomic validation method and architecture themselves, since the emergent misbehavior can be avoided with these technologies. Moreover, it advances the state of the art in this area by showing that autonomic principles can be applied in the simulation field.

5. Biological Systems Computational Modeling
   We illustrated through the stem cell computational modeling case study how the proposed technologies help on the attempt to provide computational methods to the modeling and simulation of biology systems, how their self-organizing properties can be analyzed and validated. It contributes to the usability to express the models, their processes, dynamic environment and partial view in this area.

7.2 Limitations

Given the goals proposed, it is important to understand the limitations of this thesis. We understand that to provide a notational support that customizes UML 2 is different from proposing full details of extensions, including formalizations, to UML 2. Rather, the elements that we propose in the dynamic models can be considered as a first step toward this goal.

Regarding the engineering guidelines, we have connected the notational support to the description of the proposed architecture. We have illustrated through the Unified Process how the simulation-based self-organizing architecture life-cycle can be realized. On the other hand, it was not the goal of

this thesis to provide a complete methodology or process. Instead, it means us to compare the differences between related approaches; it provides guidelines for using the notational support and architecture by showing how useful the approach is in helping a software engineer to understand how the activities are related to a well known and accepted software process and its phases.

In addition, it is important to emphasize that the proposed self-organizing architecture is simulation-based, therefore it is not supposed to be fully implemented (for instance, in a real warehouse). Instead, it works as a support to the design activity because it is usually very expensive to implement self-organizing systems. Although the core self-organizing behavior will be fully implemented at a later stage, the simulation and validation features that the architecture provides are not supposed to exist at production time since it is unfeasible.

Finally, the presented results achieved with the case studies are meant to show how the architecture could be instantiated and evaluated w.r.t real applications.

7.3 Future Directions

This work has uncovered a myriad of problems to be solved, which are listed below on next sections. Some of them are current ongoing works, and although they can be found in [Sangiorgi 2009, Motta et al. 2009, Motta et al. 2010], we describe them in more details in order to reinforce their contributions.

(a) A Model-driven Approach for the Engineering of Self-Organizing Multi-Agent Systems

There is a need for a specific editor for the notational support, since drawing the models by hand can be a really time-consuming task. Furthermore, the artifact produced in this manner only contains drawings; there is no semantic on the models, hence it is not possible to automatically validate them.

Therefore, to have a specific editor to design the models would contribute not only to faster modeling of static and dynamic models, but also would lead us to a better understanding of emergent properties of the self-organizing systems onto the proposed design. In addition, to have well defined models leads the way for code generation in the future.

The model driven approach would consist of the meta-model to structure entities, Coordination Statecharts, and an editor to support the modeling, and

a Java-based source code generation with the automatic instantiation of SSOA framework.

Sangiorgi [Sangiorgi 2009] started the development of a group of Eclipse plug-ins for the editor, mainly using GMF (Graphical Modeling Framework)[1], which allows a rapid development of graphical editors based on GEF (Graphical Editor Framework)[2] for domain models specified with EMF (Eclipse Modeling Framework)[3]. All these frameworks combined allow a user to visually develop a domain model represented in a compliant XMI.

Both the dynamic and static models have been written using EMF, extending the Eclipse UML2 framework model, just like the meta-models extend UML's model. According to [Bruck 2008], extensions to UML Tools meta-models may be done using a lightweight, a middleweight or a heavyweight approach. We had chosen to use the middleweight, since the lightweight uses profiles and stereotypes, and therefore it was impossible to redefine structure or behavior, and using the heavyweight one involves reuse by "copy and merge" instead of reuse by specializing types from the meta-model, which could aggregate more complexity to the work and lose interoperability with other UML tools.

Thus, we used a middleweight solution, creating our own state editor to support the Coordinated Statecharts modeling. The editor trees for the static and dynamic models were built in order to allow a first validation of the tool, hence the visual editor can be developed. It is possible to add specific elements to this editor (beyond the UML meta-model elements) in order to compose an appropriate model to the proposed specification.

The dynamic model editor tree is identical to a state machine editor, with the addition of the facility of transitions edition. As the transition syntax might change in the future and is very long, we showed the visible attributes in a different way (right inferior part of the Figure 3). Therefore, the visualization is flexible enough, in the future, to allow the removal or shortening of very extensive attributes.

In the same way, the straight edition of the transition is allowed, and the attributes are filled in the right inferior part automatically, of course, respecting the syntax. Therefore, the editor handles an XML file that contains an instance of the meta-model:

```
<?xml version="1.0" encoding="UTF8"?>
<dynamicmodel:Model xmi:version="2.0"
```

[1] http://www.eclipse.org/modeling/gmf
[2] http://www.eclipse.org/modeling/gef
[3] http://www.eclipse.org/modeling/emf

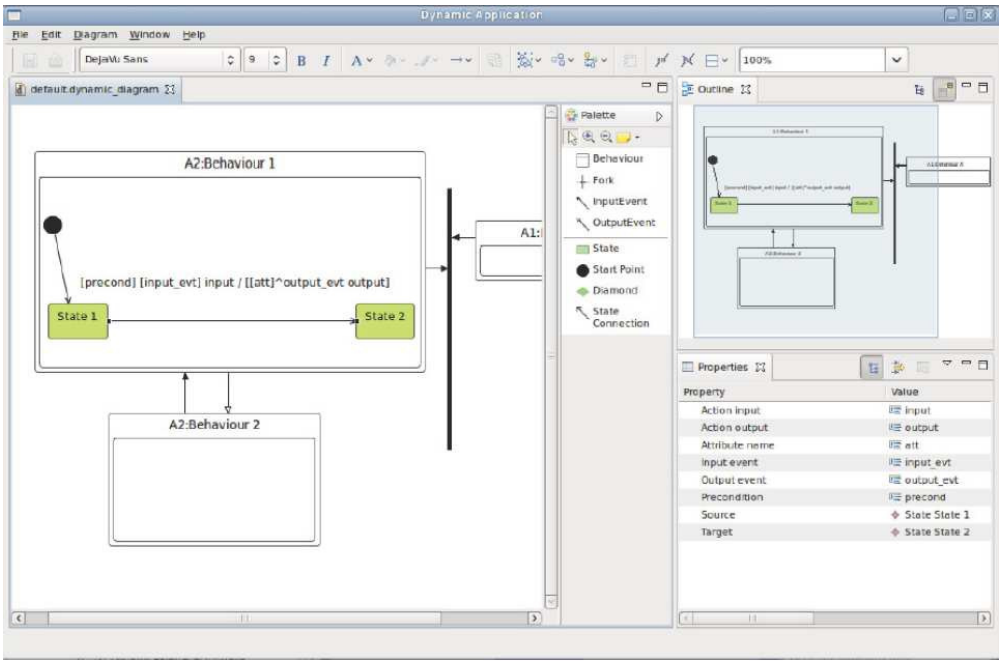

Figure 7.1: The dynamic meta-model editor

```
xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchemainstance"
xmlns:dynamicmodel="dynamicmodel">
<elements xsi:type="dynamicmodel:Behaviour" name="Behaviour 1"
agent_instance="A2">
<elements xsi:type="dynamicmodel:State" name="State 1"/>
<elements xsi:type="dynamicmodel:State" name="State 2"/>
<elements xsi:type="dynamicmodel:StartPoint"/>
<transition source="//@elements.0/@elements.0"
target="//@elements.0/@elements.1" precondition="precond"
input_event="input_evt" attribute_name="att"
output_event="output_evt" action_output="output"
action_input="input"/>
<transition source="//@elements.0/@elements.2"
target="//@elements.0/@elements.0"/>
</elements>
```

Also, an editor tree was generated for the dynamic meta-model in a way that what is seen in the editor tree is the same model in the Coordinated Statecharts (dynamic) editor.

It is expected from the editors to have an epistemic function in a way that the modeling of multi-agent systems itself leads to a better understanding and observation of coordination mechanisms in self-organizing systems. Also,

the tool could be used to model architectural patterns of self-organization optimizing the project time and effort for solutions that make use of those patterns.

Furthermore, it is desired to have automatic communication between both static and dynamic models, although we do not yet have it. Thus, the next step is to build an interface between the models that updates them in both directions, each time the models change since the programmatic interpreter of both models is already built-in. It populates objects for each element on the model, as well as updating the diagram view, whenever the XML underneath changes. Thus, an interface can provide ways of delegating such functions for the already built layer.

The next step, after the construction of update mechanisms, is the generation of code based on the built models, which is eased by the Eclipse platform, since the models are compatible with EMF framework and there are available mechanisms of reflection and model transformation.

(b) SSOA Evolution

The main directions in which the SSOA architecture can be evolved is twofold: regarding the self-organizing patterns, and the Manager component w.r.t the validation method. A catalog of self-organizing patterns can be encapsulated in SSOA architecture allowing the software engineer to easily instantiate them in specific problems.

Regarding the Manager component, a set of debug improvements can be encapsulated in a way that the Manager could provide causality relations between local actions that could not be accepted with regard to the specific macroscopic properties.

Also, the entropy concept can be introduced. From [Guerin 2004, Parunak 2001, Balch 2000], (spatial) entropy is suitable to reflect the spatial distribution of entities between different states and it is defined as:

$$E = \frac{-\sum_{i=1}^{N}(p_i \times \log p_i)}{\log N} \tag{1}$$

Where p_i is the probability that state i occurs and $\sum_{i=1}^{N} p_i = 1$. Dividing by $\log N$ normalizes E to be between 0 and 1. Entropy is high (close to 1) when the considered states have an equal probability to occur, and low (close to 0) when only a few of the states have a high probability to occur. De Wolf applied this measure to the distribution of AGVs: the different states for the entropy measure are defined as the desired situations for the AGVs. And considering the AGVs are already distributed between the desired situations.

The probability for such an AGV to be in one specific desired situation at one moment in time is used as the probability in the entropy equation.

Therefore, if the SSOA provides specific interfaces for instantiating this operator, the Manager could be able to evaluate the system w.r.t the entropy and re-initialize the simulation adaptively. In addition, the scientific numerical analysis algorithms used by De Wolf (and other related ones) could also be encapsulated in the Manager in a way that they could be easily executed to different application domains.

(c) Transparent Distribution and Performance

Another future direction is to provide a transparent distributed parallel simulation middleware. In 2D or 3D situated environments, regardless of the visualization process, one requirement is the space management. In a sequential simulation this does not incur in a problem because each request is treated in the order of arrival. However, in a distributed parallel environment requests may arrive at any time, so it is necessary to provide a solution for concurrency.

To achieve this goal, we have being investigating [Motta et al. 2009, Motta et al. 2010] on how to provide the capabilities for the proposed architecture to become a distributed and parallel solution, in the sense that it could work on a cluster environment in order to achieve better processing times or larger problem sizes. The goal for this new architecture is to provide the user with: the solution in less time, or; a solution to a larger instance of the problem in the same time frame.

However, it is important to pay special attention to the representation of the space. Regarding this issue, there were two choices that would affect the cluster evolution:

i) if we represent the space in a distributed manner, each computing node would be responsible for an octant of the space. This could improve performance when communication between two agents at the same octant takes place, but the growth would be eight machines each time, one for each octant.

ii) the other approach would be to have the space virtually represented in a centralized way, at first on a single machine, this way we can add more processing nodes easily and one at a time. Although this solution is limited by the network bandwidth and the central node throughput, it was the approach that was selected and is described in the next section.

Toward the solution

The solution selected and already started the development [Motta et al. 2009, Motta et al. 2010] is based on a cluster environment that uses processing nodes with the same hardware and software configuration and fully dedicated to the solution processing. This is better suited when trying to achieve improved parallel performance for a great number of agents that need to share or compete for some resource. Over a cluster, we have a software architecture that provides network transparency to the user.

We use an architecture based on classical solutions of parallelism to provide the user a better alternative for solving problems faster or larger problems. The solution rely on a master node that control the processing nodes and its goal is to deliver parallelism over distribution being cost effective since it allows for the growth of processing nodes, limited only by the master node processing power.

Since the architecture implementation is Java based, the solution for distribution is RMI, which is provided with the standard Java Development Kit. This is the same approach used for the WebHLA by DoD [Dongarra 2003] and is the best solution while we still have other aspects to optimize; only when all of the alternatives have been explored should we try to change the standard libraries for improved specific solutions.

For concurrency we chose to model the Sequential Object Monitors [Caromel 2004], which are special active objects that can control a given resource by providing a request queue to the clients and that use a processing thread that consumes the queue in the arriving order. This allows clients to work asynchronously with a nonblocking request call and at the same time provides a means to control requests, so that when a request gets served it is guaranteed to receive a unique space location, for instance.

The proposed solution creates an overlay network of brokers interconnected through a communication bus that is provided by the central node, which is responsible for step control and other services.

The goal is to provide a virtual environment that works exactly as the local version, at least for the application programmer. The framework's infrastructure is responsible for connecting remote nodes that should have their broker elements initialized at startup. The network bandwidth may become a bottleneck for this solution; however, a hierarchy of central nodes may be provided if necessary. A key concept here is that we use distribution as a mean to parallelism, since it is more cost effective to have more machines than a single multiprocessor. However, we have some communication and coordination cost between nodes.

Space Management as a Service

Many types of simulation will need not only 2D or 3D space management, but also many other types of supporting services, such as diffusion algorithms, force calculations and unique Id generation, the latter already being used and provided on our framework. Seeing these as services becomes much easier not only to provide them to the application through the framework, but also to evolve them in an orthogonal fashion.

Using a Service Locator [Alur 2003] Design Pattern the application code simply has to make a call to find the service needed. Implementing the correct interface, framework developers may deploy as many services as needed and may provide new abstraction levels to the application programmers.

We chose a centralized representation of the space to be used by the application. That way we may have many working nodes processing the simulation, although we will end up with many requests for locations. To avoid concurrency problems we hold a queue of agents requests, which is an implementation of the Sequential Object Monitor. Furthermore, exposing this facility as a service makes it easier for agents to use it as if it was a local resource.

Infrastructure Transparency

A major design goal is to provide the communication means to application programmers in a way that it is not necessary to know where an agent resides nor even that there is distribution, besides the start up process.

That said, all communication occurs through each node's broker. If the target agent resides on the same machine, the local broker is allowed to route the message on its own. On the other hand, if it is a remote target, the local broker routes the message to the master broker through the communication bus. The central broker is then responsible for finding the target location on its agents table and delivers the message to the target agent's local broker, which in turn will deliver it to the agent.

The Simulation Engine

Once the simulation class is ready, we use the infrastructure to deploy it to the multiple remote worker nodes. To this end, each worker node's Broker must be instantiated at startup and wait for the Central Broker to start and connect to each worker node. The worker nodes must be configured at the central node that will connect to them in order to establish a star-like overlay network through which all the communication will take place.

Every time an agent or environment is created it must register at the Central Broker to receive a unique id. The Central Broker can then register this at a local table in order to keep the physical location of every entity created. The Remote Broker for the requester will also keep a local table of agents and environments that it keeps in order to enhance communication performance.

The Central Broker and the Remote Broker must register at Java's RMI register (*RMIRegistry*) service in order to establish the distributed Java environment for the application to start.

The Communication Bus

Since the proposed architecture uses many processing nodes it would be a major issue if each node could be connected to its neighbors in order to communicate. Since we can grow the cluster with nodes, this would saturate the network with packages and it would degrade the application's performance.

To overcome this problem, we provide a communication bus based on the central broker, which is capable of routing any message to any agent. Thus it is possible to find any entity with fewer steps although communication may become a bottleneck if the cluster grows beyond a certain point.

Virtual Space: Distributed Parallel

Creating a virtual space gives the programmer the feeling that she is working on a single centralized machine, and all the distribution and parallelism is controlled and handled by the middleware. It is necessary, however, to configure the working nodes to bring up the broker element and also to configure the nodes' addresses on the central broker.

Using brokers to create an overlay network allows the construction of a distributed agent environment with centralized step control. Over the distributed agent environment, agents are able to communicate the same way they do on a single machine using the event diffusion mechanisms, according to each self-organizing pattern and its strategies.

Furthermore, having the framework divided in layers allows us to evolve each part independently; it is possible to optimize each component and have the measure of its benefits available to the whole architecture.

Discussion

Currently, we have a working prototype that is able to connect to a certain set of fixed nodes. However the simulation is started independently on

each node. In order to coordinate the remote nodes the virtual space must be fully operational.

The use of a centralized step control allows us to coordinate the execution on the multiple nodes, and to this end the simulation is not started on each node through the mechanisms provided by the original self-organizing framework engine. The scheduler for each machine is disabled and each machine is represented at the master node through an adapter that implements the *Steppable* interface becoming a simple bridge to reach remote nodes.

Another interesting issue is that, on the master node, the adapters are held in a *ParallelSequence* structure; this means that they receive their steps in parallel, each adapter on its own thread. This allows for the cluster to work with all nodes at the same time. The first, naive, implementation used a different data structure, which led to a side effect of a sequential processing cluster, i.e., each node executed one at a time according to its adapter order on the master node.

The main contribution of this solution is to provide transparency for the complexities regarding distributed and parallel processing. The use of parallelism is targeted specifically at performance improvement, whether the processing of larger instances of the given problem at the same processing time or faster processing times for the given instance size.

The solution is layered, which allows independent evolution of each layer. This capability is crucial for the benchmark of each optimization contribution for the performance improvement. There are some optimizations specifically for the communication, as follows: 1) changing from the standard RMI library to a specialized communication infrastructure; and 2) using bulk communication packs between remote nodes and the master node. Since each communication will respect the step control, we can send a set of requests for 3D space locations on a single network package and this, in turn, will improve the network usage and traffic.

Other improvements may be provided as different services like different interaction algorithms. As a means of monitoring a web console can be provided to display the state of each processing node in terms of its health regarding the distribution infrastructure; for agent behavior the user should refer to the framework agent features. It is important for the application administrator to be able to check what is going on with the machines since simulations may require long processing times, depending on the problem instance size.