2
Background

In this chapter, we provide an overview of the fundamental concepts involved in the present research. We zoom in the notions of self-organizing systems: definitions, concepts, mechanisms. Then we provide a detailed context in which agent-based simulation will be used both on the definition of the notation and of the architecture. Finally, section 2.3 details the problem description of the case studies used throughout the thesis.

2.1 Self-Organizing Emergent Systems

During the last years, several definitions and mechanisms have been examined in order to understand how computing can model self-organizing systems and how self-organizing systems can empower the computer science.

[Visser et al. 2004] define self-organization as the evolution of a system into an organized form in the absence of an external supervisor, where *"A system can be defined as a group of interacting agents that is functioning as a whole and distinguishable from its surroundings by its behavior"* and *"An organization is an arrangement of selected parts so as to promote a specific function"*.

[Serugendo et al. 2005] state that *"Self-organization is defined as the mechanism or the process enabling a system to change its organization without explicit external command during its execution time."* Further she defines *"Strong self-organizing systems are those systems where there is no explicit central control either internal or external"*; and *"Weak self-organizing systems are those systems where, from an internal point of view, there is re-organization maybe under an internal (central) control or planning."*

Another definition from Di Marzo Serugendo [Serugendo 2006] is: *"Self-organisation is the mechanism or the process enabling a system to change its organization without explicit external command during its execution time"*.

An exemplar self-organizing mechanism can be observed by the straight line of ants stretching from food sources to anthills. This straight line or path was made from hundreds of individual ants, each behaving energetically. If we

next focus on the micro view, which means from a modeling perspective looking at the individual elements of a system, of an individual ant's behavior, it is very easy to interpret the behavior of the individual as unfocused and chaotic. It is certainly very difficult to interpret its behavior as being purposeful when taken in isolation. It is only when we take a step back, and look at the behavior of the entire group, which we can call the macro view, that we can observe a purposeful global system behavior. The purpose of bringing food back to the ant hill (an emergent function) emerges from the cumulative view of the behaviors and interactions of the apparently undirected individual. Somehow the sum of the local interactions of each individual, each responding only to their local environmental, produces a stable, surviving system, even though individual ants get lost or die.

(a) Emergence

From the definition of self-organization there is no reference to the "emergence" concept. However, with the ant foraging example we notice that going from the micro scale to the macro scale, emergent properties might appear. These emergent properties are novel with regard to the local entities, in the case of the ants. In fact, De Wolf has defined the essence of emergence as the existence of a global behavior that is novel with respect to the constituent parts of the system. On the other hand, he explains that the essence of self-organization is an adaptable behavior that autonomously acquires and maintains an increased order for the system (i.e. statistical complexity, structure, ...). In most systems that are considered in the literature, emergence and self-organization occur together. These systems are called "self-organizing emergent" systems [De Wolf 2007].

Because of the complexity imposed by decentralization and the highly dynamic nature of the problem domain it is usually impossible to impose an initial macroscopic structure. The macroscopic behavior arises and organizes autonomously producing self-organizing emergent behavior [Ulieru 2004].

Decentralized Control A design constrain of self-organizing systems is that, while such system may get input from outside, this input should not comprise control instructions. And this is another difference if we are comparing a self-organizing system with an emergent non-self-organizing system: the latter might have control instructions. A decentralized approach is desired when we need to scale the solution with less effort, i.e, new elements can be added without changing any of the existing elements, and robustness, i.e, there is no central control that may fail.

(b) Coordination and Decentralized Control

Self-organizing systems are composed of interacting parts [De Wolf 2007]. And parallelism is not enough. Without interactions, interesting macroscopic behaviors will never arise, they arise from the interactions between the parts. This property is the essence of another property: coordination. Malone defined coordination as the management of dependencies between activities. Coordination can be achieved through the direct or indirect communication of local parts. Direct communication means each local part sends a direct message to another part. And indirect communication is achieved through information placed on the environment and the parts perceiving these information. In self-organizing systems, indirect communication helps the decentralized coordination, i.e, a coordination with decentralized control.

(c) Coordination and Feedback loop

The coordination in the self-organizing system produces relationships that are non-linear: a small perturbation may cause a large effect, a proportional effect, or even no effect at all. And these relationships contain feedback loops: the effects of an element's behavior are feedback in such a way that the element itself is altered. If it seeks to increase the event that caused it, it is a positive feedback loop, on the contrary, it is a negative feedback loop.

In this context, there is a general consensus in the self-organizing literature that one must think in flows rather than in transitions, because the decentralized control somehow makes the local parties to propagate information in the environment in order to another party to perceive and react to this information. Therefore, information exchange is needed. With this propose, De Wolf proposed a design abstraction called "Information Flows" [De Wolf 2007b, De Wolf 2007]:

An information flow is a stream of information from source localities toward destination localities and this stream is maintained and regularly updated to reflect changes in the system. Between sources and destinations, a flow can pass other localities where new information can be aggregated and combined into the information flow.

(d) Validation of Self-Organizing Systems

As in any software system, we need to evaluate and measure self-organizing systems. We need to verify and validate these systems. Self-organizing emergent multi-agent systems will only be acceptable in an indus-

trial application if one can give guarantees about quality of service (QoS), in particular their macroscopic behavior.

In the literature there are two ways of validating self-organizing systems designs: with qualitative and quantitative simulations. Qualitative simulations ([Gardelli 2008]) allow to check at early development stages that application designs are capable to exhibit the intended system behaviors.

Moreover, two perspectives can be studied in a self-organizing system: local or global. More perspectives can be considered if self-organization spans multiple nested hierarchical levels. In all cases measures of self-organization [Serugendo et al. 2005] mean observed organizational structure, a process that produces and maintains that structure, or a certain function or global goal that the system aims to fulfill using self-organization.

In particular, structure-based measurement focuses on assessing the system structure after it has stabilized following a series of changes in self-organization. Measures focusing on the self-organization process are related to the system's dynamics and its evolution over time. Finally, measurements focusing on the function that must be delivered by self-organization are related to how well the self-organizing system is able to fulfill its purpose. Therefore, measures in this case concern the characteristics of the problem the system is dedicated to solve and are similar to those used for performance assessment in classical systems.

Serugendo shows specific examples of typical self-organization measures such as:

- capacity to reach an organization able to fulfill the goal of the system as a whole once the system is started (success / failure / time required, convergence);
- capacity to reach a re-organization after a perturbing event (success / failure / time required);
- degree of decentralized control (central / totally decentralized / hybrid); or
- capacity to withstand perturbations: stability / adaptability.

On the other hand, how to determine those measures? How to functionally implement them? How to evaluate a required macroscopic considering that one cannot model all possible behaviors of an interaction model? We need measurements that represent these self-organizing emergent systems and exhibit trends, i.e an average system-wide behavior that are predictable. Understanding how such self-organizing components are constrained to carry out the appropriate actions at the right places and times is a key challenge.

2.2 ABS: Agent-based Simulation

A truly complex system would be completely irreducible. This means that it would be impossible to derive a model from this system (i.e. a representation simpler than reality) without losing all its relevant properties. However, in reality different levels of complexity obviously exist. If we are interested in situations which are highly structured and governed by stable laws, then it is possible, without loosing too many of the system's properties, to represent and model the system by simplification.

The reduction of complexity is an essential stage in the traditional scientific and experimental methodology. After reducing the number of variables, this approach allows systems to be studied in a controlled way, i.e. with the necessary replication of results.

Modeling is a way of solving problems that occur in the real world. It is applied when prototyping or experimenting with the real system is expensive or impossible. Modeling allows to optimize systems prior to implementation. It includes the process of mapping the problem from the real world to its model in the world of models, the process of abstraction, model analysis and optimization, and mapping the solution back to the real system.

A simulation model may be considered as a set of rules (e.g. equations, flowcharts, state machines, cellular automata) that define how the system being modeled will change in the future, given its present state. Simulation is the process of model "execution" that takes the model through (discrete or continuous) state changes over time. In general, for complex problems where time dynamics is important, simulation modeling is a better answer.

The first use of simulation is to detect which conclusions may be drawn from complex antecedents. A target system is represented by a computer model (with all the necessary simplifications). The question is about the possible futures of such a target system:

- Will the system behavior stabilize overtime or be destabilized?
- What happens if we change something in the initial conditions?
- Can the system be optimized, regarding some parameters?

This is the core of the simulation process. When the system is carefully modeled, a simulation can indeed give some very useful results about what we can expect from the target system. Of course simplifications are needed and a model, by definition, is a scaled down representation of reality. But even then the results can apply to real situations.

Since several years more and more so-called agent-based simulation models are applied for microscopic modeling of systems whose behavior is

characterized by common actions of autonomously entities. The concept of an agent is more general than the one of an individual, object or simulation process. An agent is an interactive computer system that is situated in some environment and that is capable of autonomous action in this environment in order to meet its design [Wooldridge 1995]. In this sense, agents usually move in spatial systems like cities, which means motion or movement. Usually, there are many agents - not one or two but n. When there are many agents, they react to each other through time and their collective behavior can be unpredictable, surprising, hence novel and emergent. In this way, this style of modeling is quite consistent with the sciences of complexity [Jennings 2000].

An ABS is a simulation with many intelligent agents interacting among themselves and with the environment. In a typical ABS of social behavior, the agents are the individuals that take rational decisions based on their neighbors' decisions.

ABS looks at agent behavior at a decentralized level, at the level of the individual agent, in order to explain the dynamic behavior of the system at the macro-level. Instead of creating a simple mathematical model, the underlying model is based on a system comprised of various interacting agents.

Agents are autonomous entities: an agent is capable of acting without direct external intervention. Multi-agent systems can handle the complexity of solutions through decomposition, modeling and organizing the interrelationships between components [Jennings 2000]. Finally, the locality is an intrinsic feature of an agent: the agents' decisions are taken considering only the local environment and not the global average.

Finally, feedback loops can be achieved in ABS, since the result of agent actions stimulates other actions and eventually re-stimulate the first actions. Thus, a prerequisite for a multi-agent system to exhibit self-organization is feedback loops in which agents get feedback on their actions and stimulate or inhibit each other.

(a) Multi-Agent Simulation Toolkits

As we have implemented the architecture presented in this thesis as a framework on top of MASON [Luke 2004], we briefly describe it in this section together with related multi-agent simulation toolkits. We further explain why MASON has been chosen.

Before that, it is important to anticipate an important concept which is actually presented in the proposed architecture: the environment structure. An environment is structurally distributed if, at any point in time, no centralized entity has a complete knowledge of the state of the environment as a whole.

An environment is distributed from a processing perspective if it is designed to be executed in a distributed network [Arunachalam et al. 2008].

MASON [Luke 2004] requires the user to program everything in the model including the visualization, but it offers 2D and 3D views facilities and decouple of the model. MASON provides the user with grid, continuous space extensibility support and network (graph) structures. MASON offers no distribution capabilities in terms of the processing and structure. The agent carries information about the entire environment. For an agent to walk along the environment, the coordinates of the position must be programmed into the agent. In MASON the user must program the simulation model.

NetLogo [Wilensky 2004, Vidal 2009] is a programmable platform for simulating models related to natural and social phenomena. It has a simple programming language that is very easy to understand, and easy to use to build models. Netlogo offers 2D and 3D views of the model. It provides the user with only a grid structure of the environment. An individual cell in the grid is called a patch. Patches can be sensed or not sensed by the agents, based on the user's choice of the model. the agent is aware only of the existence of a patch and its activities on the environment. The agent does not possess any information about the resources available in the patch. NetLogo provides users with HubNet which enables the same model simulation to be controlled by multiple users and, hence, the entire simulation is distributed with respect to architecture and processing. The UI contains procedures for the specification and simulation of the model in different tabs.

Ascape [Parker 2001, Inchiosa 2002] is a framework designed to support the development, visualization, and exploration of agent based models. Ascape is designed mainly for social science simulations. Ascape offers only 2D visualization and no 3D visualization. Ascape requires some basic programming skill from the user. In order to specify the environment and the agents, the user has to program. Ascape offers grid and graph environment structures. An individual lattice is called a cell and the agents that interact with the cell are called cell occupants. In the graph structure, the cells are the nodes and the cell occupants move from node to node. Ascape offers no distribution capabilities in terms of the processing and structure. In Ascape, the user must specify the node positions within the agents. In Ascape the user must program the environment model.

RePastS [North 2007] was initially developed to support social science applications. The latest version is RePast Simphony and can be used to simulate a variety of applications (e.g., network simulations, GIS applications). RePastS offers continuous space, grid, network, geography, and scalar field

structures. RepastS offers 2D, 3D visualizations and allows the user to choose the nature of the space in graph and grid structures. RepastS possesses only distributed processing capabilities. The environment structure is not distributed. the environment and agent can be decoupled for simpler models. The user partially specifies the environment by filling in properties in the model file.

DIVAs (Dynamic Information Visualization of Agent systems) [Mili 2006] is a platform that includes a specification and a simulation tool that run in the Eclipse IDE. So far, DIVAs has been predominantly used for social simulations. DIVA offers only a 2D visualization and there is no programming involved in specifying the model. The entire model is specified using a graphical editor. DIVA offers a graph based environment structure, which has nodes and edges. Agents move along these edges. DIVA has a distributed environment structure but has limited distributed processing capabilities. The agents are completely decoupled from the environment. The user only has to enter information in tables in the specification tool.

Since we started the definition of the architecture to suit the development of biological systems, in particular, stem cell behavior, the most important criteria for choosing a tool was the complexity of the environment and the visualization features. Moreover, the more choices for environment structures, the broader its application in the field of multi-agent simulation systems. Based on that we had to choose between RePastS and MASON. We found MASON's learning curve lower than RePastS. More details and comparisons between those tools can be found in [Arunachalam et al. 2008, Railsback et al. 2008].

2.3 Case Studies

(a) Case Study 1: The Automated Guided Vehicles

The case study of the Automated Guided Vehicles (AGV) was chosen because it has been used by the self-organizing related work [Weyns 2006, De Wolf 2007] and is a complete problem of self-organization capable of validating all the components of the architecture.

An AGV transportation system uses multiple computer-guided vehicles in warehouses. Each vehicle, or AGV, moves loads (e.g. packets, materials) in a warehouse from the pickup to the drop stations and can only perform a limited set of local actions, such as move, pick up load, and drop load. The goal is to efficiently transport incoming loads to their destination. In this dynamic problem, the warehouses can have many layouts, loads may arrive at

any moment, AGVs move constantly and fail, obstacles and congestion might appear, and there is no central control. Therefore, each vehicle's movement should result in feedback with the others and the environment. For this problem, we need to find the set of combined values for the number of pickup stations, drop stations, and AGVs; plus the rate of new income loads in a pickup station that maximizes the throughput of the dispatching and routing processes.

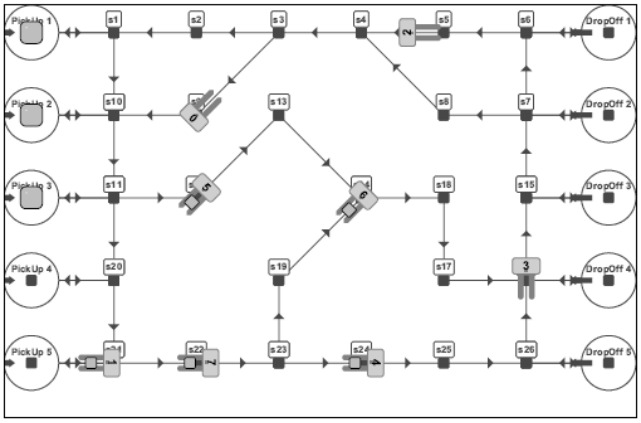

Figure 2.1: AGV case: example factory lay-out, from [De Wolf 2007]

The Problem

The dispatching, which is the process of an AGV to receive a load, and routing, which is the process of an AGV to carry a load from the Pickup Station to the nearest Drop Station to avoid congestions, requires a mechanism that enables aggregation and calculation of extra information while flowing through the warehouses. In [De Wolf 2007] the decentralized control for this problem is achieved through the use of Gradient Fields. Therefore, we have modeled the system following this rationale, although not using the same implementation solution they provided, because our solution was built on top of the architecture herein proposed.

In this way, three gradient fields that indirectly guide the automated vehicles were defined: (i) the Pickup Gradient, generated by the Pickup Stations to notify that there are loads to be dispatched. The farther the gradient is from the source station, the weaker it is. And this strength or weakness helps a vehicle to choose the nearest station; (ii) the Drop Gradient, designed with the same mechanism of the Pickup Gradient, which helps a vehicle to choose the nearest Drop Station once it has a load to be delivered; and (iii) the Vehicle Gradient, a specific repelling gradient fired by all vehicles with the goal of avoiding collisions or congestions.

(b) Case Study 2: Contracts that Govern Emergent Multi-Agent Systems

IST-CONTRACT [Meneguzzi 2008, Oren 2008] is a research project funded by the European Commission in the context of the 6th Framework Program. The aim of the project is develop frameworks, components and tools which make it possible to model, build, verify and monitor distributed electronic business systems on the basis of dynamically generated, cross-organizational contracts which underpin formal descriptions of the expected behaviors of individual services and the system as a whole. The project covers both theoretical and practical aspects and the resulting systems will make it possible to:

- specify electronic business-to-business interactions in terms of contracts,
- dynamically establish and manage contracts at runtime in a digital business environment,
- apply formal verification techniques to collections of contracts in a digital business environment, and,
- apply monitoring techniques to contract implementation in order to help provide the basis for business confidence in e-Business infrastructures.

As part of the IST-CONTRACT project, we have used of the Electronic Contracting in the Aerospace Aftermarket [Gatti 2010]. In this scenario, aircraft operators buy engines for their aircraft from engine manufacturers. However, the business relationship does not end at this point. In order for the aircraft to keep functioning, the manufacturers must service aircraft engines over time, and provide working engines so that the airline operator's aircraft can be kept flying while other engines are being repaired, this is referred to as aftercare. The engine manufacturer is paid by the hour and may face a penalty if planes are on the ground waiting for a working engine. In this business model, timely servicing becomes a key driver of long term profitability between parties.

Aircraft fly between airports based on pre-defined routes and are serviced at manufacturers' service sites, located at the airports. Servicing involves, first, replacing an engine requiring repair with one from a pool of working engines. The removed engine is then repaired using parts sourced from parts suppliers. Aftercare contracts agreed between operators, manufacturers, and part suppliers dictate the obligations and restrictions placed on each.

Aftercare contracts are complex with stipulated service levels and penalties for failure to meet them. Operators want to understand whether the aftercare contracts they agree to will lead to their business goals being met.

Specifically, they wish to know that aircraft will be in a state able to fly and that the contractual obligations will not be violated. Moreover, they would like to know whether these aims are both (1) met to a high degree in any period of time, and (2) met steadily over time.

An Exemplar Scenario

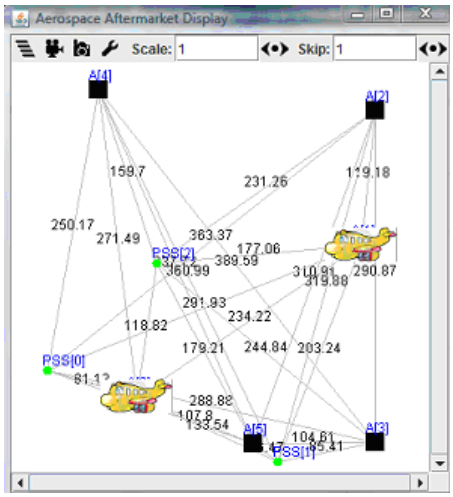

Figure 2.2: The Aerospace Scenario

Figure 2.2 illustrates the problem setting: there are airports on which the operator's airlines have employees (squares). There are parts manufacturing sites (circles) which are not located at the same site as the airports. Hence, there is a need (and costs) to shipping parts. The engine manufacturers are at some airports, not all. And the edges are the routes and contain the distance information between airports and part manufacturing sites.

In this scenario, an aircraft can be flying in any route from one airport to another airport. After flying a certain cumulative distance or after a given period of time, the aircraft will need to be serviced. Servicing an aircraft means replacing its engine or a part, whichever needs repair, with a working engine/part. Once an engine in need of repair is removed from the aircraft, the engine manufacturer will repair that engine, returning to the pool of available working engines.

Exemplar Norms

Users of the simulation, and we focus here on the aircraft operator, would like to understand the consequences of the contracts they draw up with others in the system. These contracts influence how the agents are expected to behave, and the costs of not doing so. Overall, the aircraft operator would like to

know whether the contracts are effective in two regards. First, they wish to keep aircraft flying (and so make a profit). Second, they wish to avoid costs, including those due to violations of the contract clauses, as this would suggest that the contract is unrealistic.

Consider that there are four classes of contract clause, each clause expressed as a single norm, to be analyzed in the context of the aerospace aftermarket problem description:

N1. Permissions for the airline operator to use particular sites for servicing. This means that the plane can only be serviced at a site with which it has an associated permission. There is a cost for having permission to use each site. The cost is applied for the overall use. The operator must be able to decide to add or remove a permission to use a site to optimize its costs. The operator may decide to violate this norm if the aircraft is not able to fly to the next site because it requires repair.

N2. Permissions for engine manufacturers to use particular part suppliers. This means an engine manufacturer can only receive parts from a part supplier with which there is permission. Having more permissions with part suppliers could increase the frequency of part failure (the reason for not providing permission to use a supplier is that some suppliers supply poorly maintained parts). The decision to add or remove a permission to use a part supplier should be taken considering cost optimizations. The engine manufacturers may decide to violate this norm when the next two norms are violated.

N3. Prohibitions on the part suppliers to supply parts already used by a list of different operators. Airlines sometimes do not trust each other to properly maintain parts, and so will have restrictions on who has used a part in the past. A cost will be applied if this norm is violated. The part supplier may decide to violate this norm, when there is no available part to dispatch.

N4. Obligation for the engine manufacturer to deliver serviceable engine in a specified time. This is a critical norm for keeping the aircraft flying. If the engine manufacturer broke this norm, the operator can apply successive penalties until the engine is delivered. The engine manufacturer may have to violate this norm when there is a high demand for engines. Each of N1 to N4 above specify a class of norms, e.g. N1 is a class of obligations parametrized by the operator and the site. We want to determine which combination of norms have a positive or negative impact on the system with regards to the criteria above.

Norms Violation

Each norm has a clause that defines the penalties if the norm is violated. An important issue is: when does the norm's assignee decide to violate the norm? In the aerospace case study, the parties share the same decision making process of violating a norm, as follows.

Regarding norm N1, the Airline Operator may decide to violate if she is waiting for more than a specific time to use the site. This time is not related to the time specified in norm N4 for the engine manufacturer to deliver the serviceable engine.

If an engine manufacturer requests parts and the supplier does not ship the part in sufficient time and no other allowed supplier is available, the manufacturer may violate norm N2. This in turn may increase the time the engine manufacturer takes to deliver the repaired engine, violating norm N4.

Finally, norm N2 might be violated when the part supplier does not have new parts to supply but has used parts on the disallowed list.

The Problem

Given variations on these norms, we are interested in not only the absolute utility provided, i.e. the amount of time aircraft spent flying and costs of violations, but also the trends, i.e. the steady with which aircraft continue to be flying and the steadily of fulfilling the norms without penalty.

Let na be the number of airports. Let no be the number of different types of airline operators. Let ne be the number of engine manufacturers. Finally, let np be the number of part suppliers. The types of contract possible to specify can be calculated from:

- N1: there would be na distinct clauses, one for each airport;
- N2: there would be $ne \times np$ distinct clauses, one for each pair of engine manufacturer and part supplier;
- N3: there would be $no \times no$ distinct clauses, one for each pair of airline operators; and
- N4: there would be $ne \times t$ distinct clauses, where t is the specified time for the desired minimum repair time.

Hence the total number of possible distinct contracts is

$$NbrDistinctContracts = na \times (ne \times np) \times (no \times no) \times (ne \times t) \quad (1)$$

This means an infeasible amount of time to simulate all possibilities.

The aim is to find the best contract, plus to achieve it fast which means in less trials and with less effort, i.e., as automatically as possible. In this scenario, suppose we started the simulation with all norms active, a lot of permissions, etc. Or the opposite: we started with no norm. Our solution provides a support technique and tool that will work by itself trying to find the best set of norms according to the goals. In the first case comparing the trends with the removal of a norm, or in the second case with the addition of a norm. If a norm, for instance, contributes very negative to the trend, the simulation will rollback until the point where it added the norm, and will try a different one. At the end, we will have the best contract.

(c) Case Study 3: Computational Modeling of Stem Cell Behavior

Stem cells play a prominent role in biology and life sciences. Their importance is growing more and more, not only in basic research fields such as cell or developmental biology, but also in medicine and clinical research. The main reason underlying this broad interest in stem cells is their capacity to reconstitute functional tissues after disturbance or injury. They are able to produce a huge number of differentiated, functional cells and, at the same time, they maintain or even re-establish their own population.

A stem cell is a primitive cell that can either self-renew (reproduce itself) or give rise to more specialized cell types. The new perspective on stem cell systems as networks of different cell types and their interactions implies that stemness should not be treated as an explicit cellular property, but as the result of a dynamic self-organization process. The micro-environment interactions and their specific effects on proliferation and differentiation have to be embedded in the concept.

Nowadays, stem cells are cultivated in the lab in order to differentiate into a specific mature cell. Today it is hard to predict the stem cell behavior under some substances. Moreover, the entire infrastructure necessary to maintain a stem cell culture is very expensive and many stem cells are wasted if the injected substance does not lead the culture to the desired mature cell. Thus, stem cell simulation is a powerful tool for reducing such costs and accelerating the stem cell therapy process. We have been developing an agent-based solution to model and simulate the stem cell processes and the internal cell life-cycle [Gatti 2007a, Gatti 2008, Faustino et al. 2008].

An agent-based cell computational model is a distributed autonomous entity composed of reactive and pro-active agents. It can perceive and signal the environment. At the agent-based stem cell computational model, there are

three kinds of cells (see figure 2.3): multi-potent cells are cells with a high power of differentiation that can give rise to several other cell types; neuron progenitor cells are cells able to self-differentiate into a neuron; and non neuron progenitor cells are cells able to self-differentiate into any kind of cell but neuron's types.

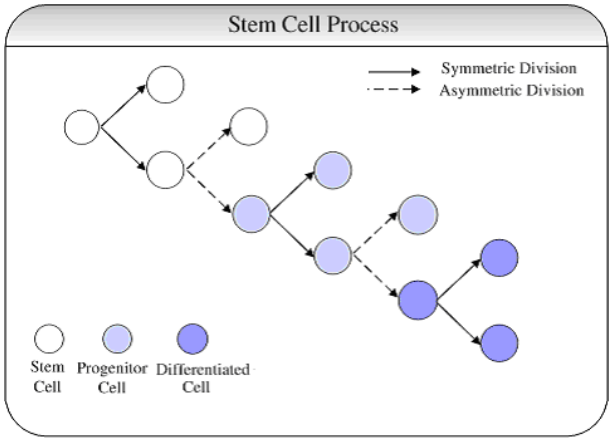

Figure 2.3: The Stem Cell Division and Differentiation Process

In order to ensure self-renewal and differentiation, the stem cells undergo two types of cell division: symmetric division: giving rise to two identical daughter cells, both endowed with stem cell properties; and asymmetric division: produces only one stem cell and a progenitor or precursor cell with limited self-renewal potential until the mature cell generation with no differentiation potentiality. In order to simplify the model we considered only the more actives components during the cell life-cycle and the mitosis division (which is the stem cell division during the self renew process). By active components we mean components which contribute and influence more directly the cellular differentiation during the cycle. Cells, proteins, DNA and complexes such CDKs-Cyclins are agents.

The cells can be a proliferative cell or non-proliferative cell. The stem, progenitor and precursor cells are proliferative, while the neuron cells are non-proliferative. CDK, Reitinoic Acid, Cyclin and LIF are proteins. The cells can die or start a cell cycle phase. Each cell cycle phase may only be started if some events occur, this is the cell checkpoint mechanism. For instance, during the prometaphase phase the cell checkpoint checks if all the kinetochore microtubules were attached to each chromosome, and if the nuclear membrane was dissolved in order to go to the metaphase phase. The DNA which is an adaptive agent interacts with the proteins during the molecular pathways regulation.

Spatial Self-Organization

The real life spatial self-organization of stem cells is explained as follows. The stem cell divides itself into two news cells, due to the mitoses division process. Both cells assume new positions: one assumes the parent position and the other an adjacency position. Before the division process, the cell grows up, pushing its neighbors away and occupying the required space for the new cell. In order to model the 3D spatial self-organization of the stem cells, we must find an empty space among its neighbors that minimizes the cell effort. The cell effort is minimized by pushing the smallest amount of neighbor cells.

The Problem

There are two macro problems for this case study. First, we have to observe the simulation of the self-organizing stem-cell represented by agents and compare/ validate according to the real behavior. And second, we need to regulate the emerging behavior. By regulation we mean the establishment of optimum differentiation or proliferation rates, for instance, through the addition and removal of some specific factors in the niche.

For instance, the main specific challenges are: how to identify local rules in a way that the perturbation induces a cancer? Moreover, how to identify a cancer in the emergent behavior? In particular, our stem cell optimal growth ratio must satisfy the following rules:

A) To minimize the cellular death It is empirically known by the stem cell researches that if we add a particular factor in the culture the cellular death decreases. However, it increases the number of undesirable differentiated cells.

B) To maximize the differentiated cell number.
The colony ideal size and characteristics change the differentiation process. For instance, suppose a colony size to be X. Then the factor has no effect. However, if the colony size grows to 2X, then the factor has.

C) To maximize the number of a specific kind of mature cell.
As we are working with a neurogenesis lab, they want all the stem cells to become neurons. Furthermore, it is known that cells with fewer chromosomes differentiate to neuron cells.

D) To minimize a tumor probability.
On the other hand, cells with more chromosomes might generate a cancer if they do not die. So it is necessary, for instance, to evaluate this trade off according to rule C.

2.4 Chapter Remarks

In this chapter we presented the main fundamental concepts of self-organization, the agent-based simulation technology concepts as a background for the main contributions of this thesis presented in next chapters. As a result the reader is able to understand the link between self-organizing systems and multi-agent systems which, if connected, represents the self-organizing emergent multi-agent systems. We briefly described the main multi-agent simulation toolkits and explained why MASON was the chosen as the one to implement the proposed architecture described in chapter 4. Finally, in order to better understand the motivation of the proposal of an architecture, we presented the detailed description of the three case studies used on this thesis that covers different domains.