Université Catholique de Louvain
Ecole Polytechnique de Louvain
Département d'ingénierie informatique

Master Thesis:

Designing robust adaptive software with feedback structures

Promotor :
Pr. Peter Van Roy (UCL, BE)
Readers :
Pr. Seif Haridi (KTH, SE)
Mrs. Ruma Paul (UCL, BE)

Varotto Nicolas

June 1, 2013

Nonlinear systems engineering is regarded not just as a difficult and confusing endeavor; it is widely viewed as dangerous to those who think about it for too long.

Wilson J. Rugh
in *Nonlinear System Theory: The Volterra/Wiener Approach* [39]

This book of mine has little need of preface, for indeed it is "all preface" from beginning to end.

D'Arcy Wentworth Thompson
in *On Growth And Form* [45]

Remerciements

Plusieurs personnes ont contribué directement ou indirectement à la réalisation de ce travail.

Tout d'abord je tiens à remercier le Professeur *Peter Van Roy*, mon promoteur, pour son attention, sa disponibilité et son écoute. Le sujet de ce mémoire étant vaste et similaire à une jungle inexplorée, il m'a laissé la liberté de mener ma barque où bon me semble, tout en me recadrant lorsque je faisais fausse route.

Merci également à mes lecteurs, Seif Haridi et Ruma Paul, pour le temps qu'ils ont consacré à la lecture de ce mémoire.

Merci à *Ruma Paul*, dont l'avis pertinent me fût fort utile suite à la lecture de mes premières ébauches.

Merci à ma famille pour leur soutien.

Enfin merci à mes amis, Ju, Jé, Ed, Bram, Gub, Val, Closset, Badoux, Nath, Adri, Max, Robi, Loïc, Seb, Antoine, Bapt, Math et tous les autres qui m'ont rappelé que de temps en temps, malgré mon mémoire, je me devais d'avoir un minimum de vie sociale.

Merci à tous!

Nico.

Contents

à mes grand-parents...

Chapter 1

Introduction

Contents

"*A building is conceived when designed, born when built, alive while standing, dead from old age or an unexpected accident.*" Mathys Levy and Mario Salvadori[27]

Based on this sentence, we can make an analogy with software development phases: design, implementation and maintenance. But what doesn't sound good is the *dead*. In fact a software is designed as he will live forever, he doesn't die from old age. But if we are interested at the historical reality, we see that software actually dies.

By *dead*, we don't speak about program failure (such as the famous Ariane 5 software bug) but the fact that the requirements and specifications are no longer met, due to the changing environment or changing expectations. Have you ever check the amount of money spent by companies to overcome their software problem just because there is a new environmental parameter to be considered? So huge for simple adjustments and reconfigurations.

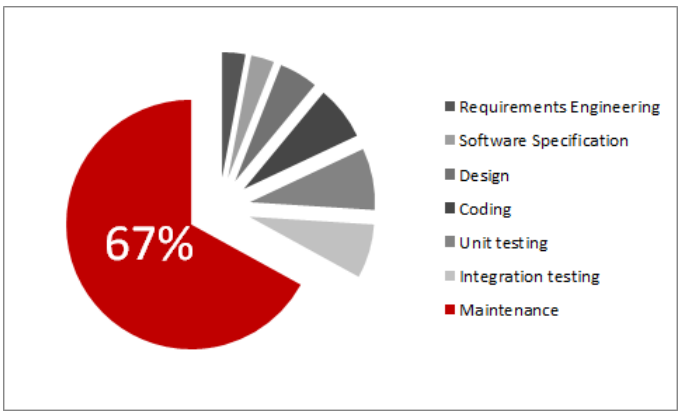


Figure 1.1: Typical distribution of total production efforts/costs[47]

Can we remove or, softly, reduce these problems by increasing the software's change tolerance? Yes, by introducing a new qualitative way of reasoning. It's the goal of my thesis : *designing robust adaptive software with feedback structures.*

1.1 Autonomic Computing and Self-management

Today, we are increasingly dependent on computer systems and high-tech, and these systems take more and more space in our day-to-day life (Intelligent Parking Assist System, Home Automation, Networked Systems, etc..). In addition, there is a runaway technology and it continues to grow, Moore's Law illustrates this situation. This problem poses a great challenge for both science and industry because this increase implies a strong growth of the complexity of modern systems. This complexity becomes the primary factor limiting the expansion of such systems. Indeed, the systems are such as that a human person could not have the required skills to manage them in a reasonable time.

It implies that more and more efforts are spent to overcome this problem. One of the most famous pioneer approaches is IBM's *autonomic computing initiative*, in 2001 [15]. The goal is the creation of selfmanagement networks to overcome the increasing complexity of Internet and other networks.

In this paper, there's a biological analogy between "autonomic computing" and "nervous system" that protects and regulates our body continuously and unconsciously: if I'm playing chess, I can focus fully on my game without thinking about the activity is going on inside my body. This example illustrates the fact that the nervous system manages our heartbeat, our body temperature,... in an inconsciously way. While the system takes care of vital tasks (low-level) our conscious mind can handle high-level tasks. The analogy with the autonomic computing is obvious: the idea is that computer systems can themselves manage routines tasks while system administrators would handle real tasks instead of troubleshooting or repair systems. All this waste

of time that we could devote to improving the system itself and moving forward towards new innovative applications. In addition, human intervention can be lengthy, incorrect, and often non-optimal.

In this paper, the notion of *self-management* was also introduced. Even today, many people find it difficult to distinguish the notions of self-management and autonomic computing. Actually self-management is part of the autonomic computing.

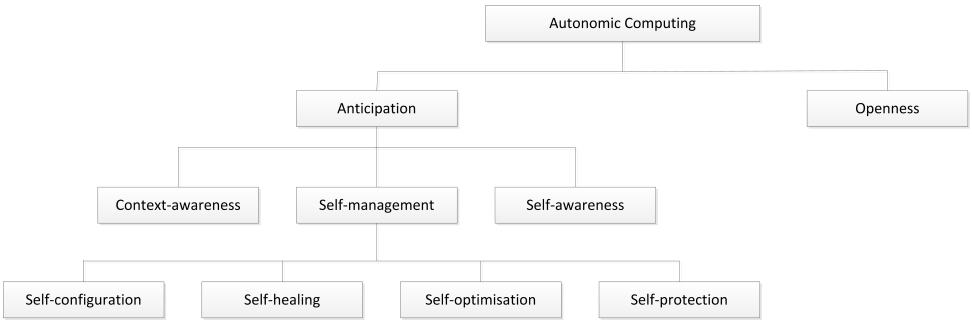


Figure 1.2: Autonomic Computing characteristics hierarchy

This work aims to provide a hierarchy according to the characteristics of the autonomic computing[32].

We can now consider that the self-management is an emerging field in computer science, a new way of thinking, a new paradigm. As stated above, this paradigm allows, according to administrator's goals and given high-level objective, computer systems and applications to manage their own operations without human intervention, which will cause reducing administrative overhead. These costs increase dramatically for distributed applications that are deployed in volatile environments such as peer-to-peer overlays which aggregate heterogeneous, poorly managed resources on top of relatively unreliable networks[14].

Extract

This passage comes from an article called *The Vision of Autonomic Computing*[21]:

> *Systems manage themselves according to an administrator's goals. New components integrate as effortlessly as a new cell establishes itself in the human body. These ideas are not science fiction, but elements of the grand challenge to create self-managing computing system. [...] As systems become more interconnected and diverse, architects are less able to anticipate and design interactions among components, leaving such issues to be dealt with at runtime. Soon systems will become too massive and complex for even the most skilled system integrators to install, configure, optimize, maintain, and merge. And there will be no way to make timely, decisive responses to the rapid stream of changing and conflicting demands. The only option remaining is automatic computing: computing systems that can manage themselves given high-level objectives from administrators.*

By staying in the analogy of the human body, the great challenge for the future would be that eventually we can integrate new components into a self-managing computing system as easily as a cell in the human body.

Thus purpose of self-management is not to replace humans entirely but rather to enable systems to adjust and adapt themselves automatically to reflect evolving policies and general behaviours defined by humans.[2] In concrete terms, the self-management initiative advocates self-configuring, self-healing, self-optimization and self-protection. We will call this by *self-**. The fact that the environment in which takes place the autonomic computing is dynamic adds a difficulty to the design of the application.

Coming back to the analogy, we can clearly see the robustness of the human body adapts and reacts depending on the environment. This is what should be autonomic systems by maintaining and adjusting their operations to a multitude of parameters: external conditions, demands, components, workloads, ... to prevent the software and hardware failures, which can either be innocent or malicious.

The approach for designing such systems is to create components that continuously monitor and manage subsystems which they are granted, this is called the control loops. From this observation, we can apply *control theory*. The disadvantage is that control theory is a quantitative approach and requires a large number of calculations and a big mathematical reasoning coupled with a mastery of optimization. This begs the question: *is there a qualitative and rigorous reasoning for the design of such*

systems?

Regarding the architectural point of view, we shall look at systems with a high level of abstraction. Techniques and design rules of such architectures, for autonomic systems but also for other software and for some no IT-oriented exemples, will be discussed in this paper.

By feeling, we realize that the autonomic systems must themselves be composed of autonomic elements. These elements can be seen as individual and independent systems, with weak interactivity between them. These elements will manage themselves their behavior, we will introduce it in the next chapter.

1.2 Feedback thinking

Before going further, we must introduce the notion of *feedback*. Actually, all complex systems around us can be seen as a set of feedback strutures. Let introduce it by a small example:

Example: thermostat

In 1948, in his book *Cybernetics: or Control and Communication in the Animal and the Machine*[30], Norbert Wiener describes a thermostat like a feedback chains in which no human element intervenes:

> *There is a setting for the desired room temperature; and if the actual temperature of the house is below this, an apparatus is actuated which opens the damper, or increases the flow of fuel oil, and brings the temperature of the house up to the desired level. If on the other hand, the temperature of the house exceeds the desired level, the dampers are turned off or the flow of fuel oil is slackened or interrupted. In this way the temperature of the house is kept approximately at a steady level. Note that the constancy of those level depends on the good design of the thermostat, and that a badly designed thermostat may send the temperature of the house into violent oscillations[...].*

According to the dictionary *feedback* is a process in which information about the past or the present influences the same phenomenon in the present or future.

This is a simple example which illustrates that the notion of feedback is present in our everyday life. In biology (eg ecosystem, organism), electronics (eg amplifiers) in population analysis (eg demography growth), in economics (eg stock market bubble), or simply in our bathroom (tank water level in the flush toilet).

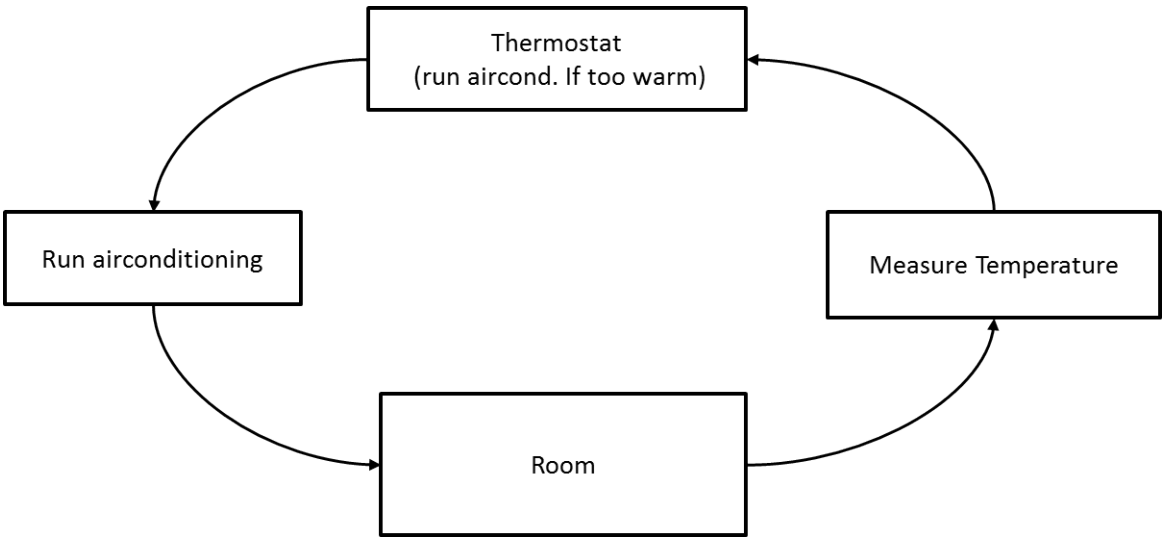


Figure 1.3: A thermostat as a feedback loop

We can control feedback to regulate certain quantities: here it is to control the temperature but there are other uses such as cruise control in an automobile or the human sensorimotor system.

Even unconsciously, we use it, as illustrated by the following example.

Example : icy road

Another Wiener example from [30]:

> *Another interesting variant of feedback systems is found in the way in which we steer a car on an icy road. Our entire conduct of driving depends on a knowledge of the slipperiness of the road surface, that is, on a knowledge of the performance charachteristics of the system car-road. If we wait to find this out by the ordinary performance of the system, we shall discover ourselves in a skid before we know it. We thus give to the steering wheel a succession of small, fast impulses, not enough to throw the car into a major skid but quite enough to report to our kinesthetic sense wheter the car is in danger of skidding, and we regulate our method of steering accordingly.*

Despite being old, this example is very relevant because feedback systems really shape our world. And as said before, we use it every day, even when we do not pay attention.

But what exactly is a feedback system? Actually we need different levels of abstraction to illustrate it.

We'll start with the basic component and illustrate it in parallel with an example on the respiratory system (developed by peter van roy [35] [38]). Afterwards we will

increase the complexity and the abstraction. This is an overview, we will go deeper and formalize this in chapter 3.

Single component Also called *concurrent component*, this is an active entity communicating with its neighbors through asynchronous messages.

Note that this entity can operate without any message, not just like a port object who works only on receipt of a message, this is more than that! We see this as an entity which has its own behavior and that, when it receives messages, processes them by causing the appropriate action or by changing its behavior. To make an analogy, it's a bit like a beehive that although it is influenced by the coming and going of foraging bees continues to produce honey and to have its internal activity.

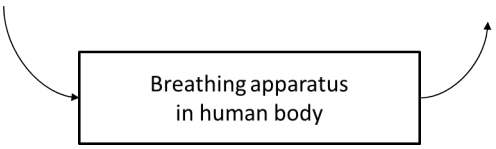


Figure 1.4: A single component

A single component can be a subsystem, as it is the case with the breathing system.

Finally note that we can have *intelligence* concentrated in core components such as *conscious control of body and breathing*.

Feedback loop A feedback loop in its general form consists of three parts, a monitor, a corrector, and an actuator, attached to a subsystem[35]. Each of these parts is an *agent*. The agents and the subsystem are concurrent components that interact by sending each other messages. A feedback loop has the characteristic of continuously maintaining one local goal.

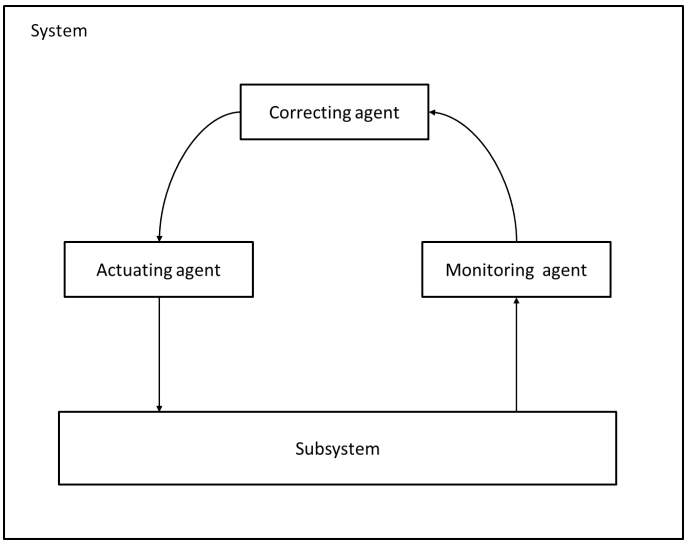


Figure 1.5: General diagram of a feedback loop

Each part can perform either a local or a global action. The corrector component contains an abstract model of the subsystem and the local goal. The feedback loop runs continuously, monitoring the subsystem (Monitoring agent) and applying corrections (Actuating agent) in order to approach these goal (Correcting agent).

We can apply this general diagram in our respiratory system example:

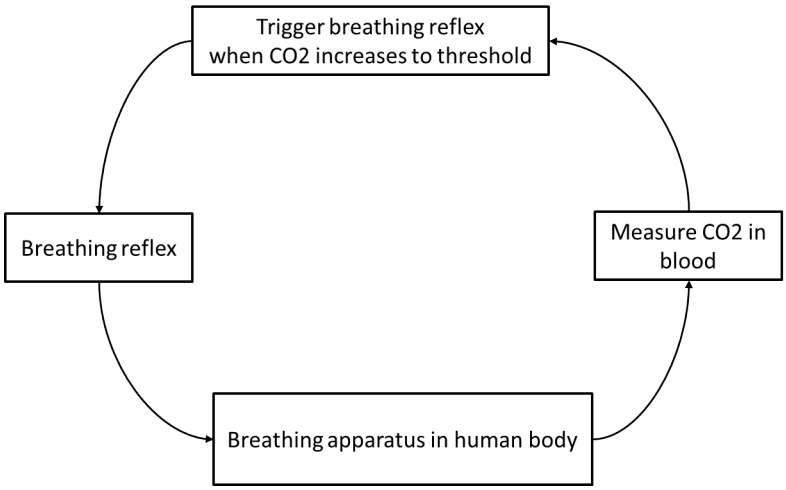


Figure 1.6: A feedback loop in the respiratory system

This is the normal scenario in humans respiratory, it's a passive process.

Feedback structure Each feedback structure consists of a set of interacting feedback loops that together maintain one desired global system property[35]. This property is maintained through the feedback loop of the goals that constitute this feedback structure. It is often organized as a hierarchy where each feedback loop may control an inner loop and be controlled by an outer loop.

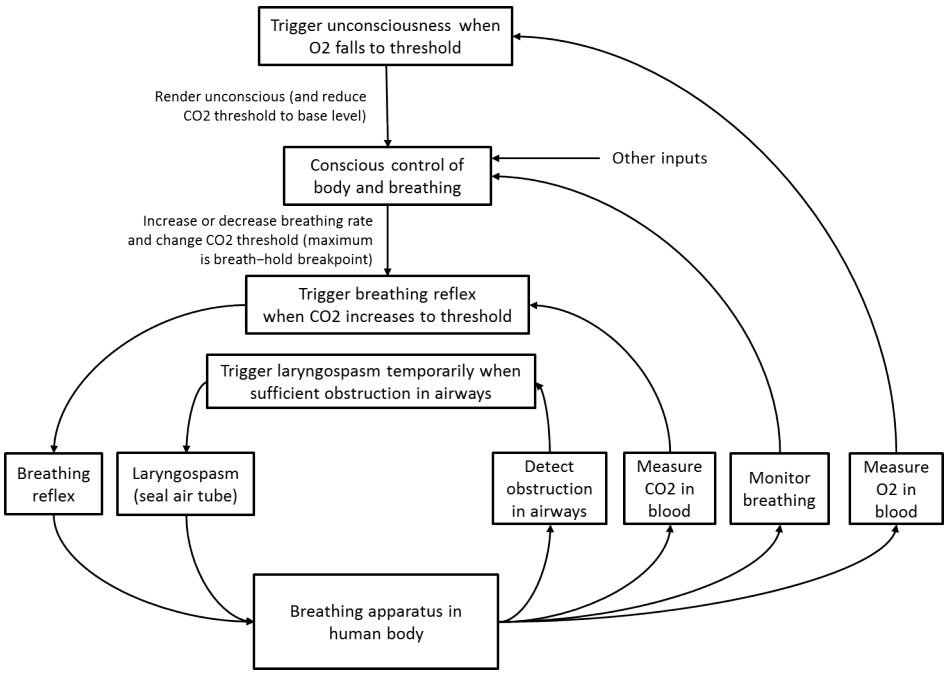


Figure 1.7: A feedback structure representation of the human respiratory system

As you can see, all feedback loops are not active at the same time: for example, in your state *normal breathing* you do not realize that you breathe. But if your environment changes, let's say you fall into the water, you realize the need to hold your breath so that the water floods into your lungs. We will see later that the feedback loops of feedback structures activate and deactivate themselves depending on the current state. This state is determined by the environment as well as internal parameters of the structure. In a feedback structure, each combination of active feedback loops is a state, we will develop it in the section on the state diagram (section 3.3).

Weakly Interacting Feedback Structures Based on the fact that a complex system can be seen as a conjunction of global properties, and the fact that we have seen above that a feedback structure maintains a property, we can infer that all complex system can be seen as *weakly interacting feedback structures* (WIFS). Each of the system property is implemented by one feedback structure.

Interaction between feedback structures is limited and well-defined (hence *weakly*). In a well-designed system, no part exists outside of a feedback structure.[35]

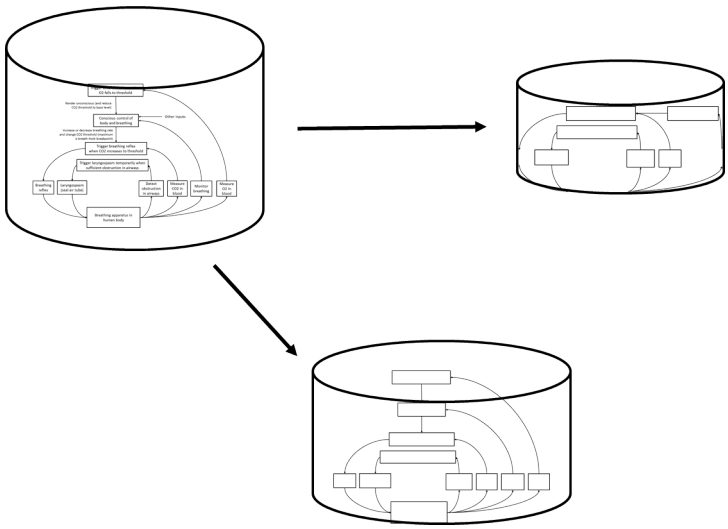


Figure 1.8: Schematic view of weakly interacting feedback structure

The whole human body can be seen as WIFS with these components: respiratory system, immune system, heart, digestion, etc.. The chapter 6 is devoted to introduce this concept with Hypothalamus example. Organizations such as human societies, as the Catholic University of Louvain, for example, can also be seen as WIFS.

We will see in chapter 3 that in weakly interacting feedback structures, the feedback structures are relatively independent, but not completely. However, they are independent enough to make their own regulation.

Now that we have an overview of what are the feedback systems, a question arises: What is the relationship between this section and the previous section on the autonomic computing and self-management?

By way of introduction, we will illustrate this with a text about the *Selfman Project*, a 3-years European research project that incubates self-managing internet applications[1].[36]

[1]More info: *www.ist-selfman.org/*

Extract : *Selfman Project*

This text is from *Self Management for Large-Scale Distributed Systems based on Structured Overlay Networks and Components - European Sixth Framework Programme Priority 2, Information Society Technologies : The Adventures of Selfman - Year Three*:

> *A self-managing application consists of a set of interacting feedback loops. Each of these loops continuously observes part of the system, calculates a correction, and then applies the correction. Each feedback loop can be designed separately using control theory or discrete systems theory. This works well for feedback loops that are independent. If the feedback loops interact, then your design must take these interactions into account. In a well-designed self-managing application, the interactions will be small and can be handled by small changes to each of the participating feedback loops. [...] It can happen that parts of the self-managing application do not fit into this 'mostly separable single feedback loops' structure. [...] In the case where the feedback loop structure consists of more than one loop intimately tied together, the global behavior must be determined by analyzing the structure as a whole and not trying to analyze each loop separately.* ***To our knowledge, no general methodology for doing this exists.*** *We have made progress on two fronts: design rules for feedback structures and patterns for common feedback structures. [...] We are preparing a comprehensive survey of feedback loop patterns*

Because there is no general methodology, the purpose of this paper is to contribute to fill this gap. **The goal is to establish a first general methodology that allows us to design systems, and especially self-management systems, based on feedback structures.**

1.3 Contributions

As previously stated, there are not yet rules to design discrete system rigorously based on a qualitative reasoning. In 1994, Steven Strogatz taught us how to reason qualitatively and rigorously in continuous system [43]. This dissertation therefore aims to develop the laws of design in the same way that Strogatz, except that the application is done in discrete systems. There are considerable prospects in this area, and just a few insights has been made.

There are already some design methodologies, but these are quantitative: *System Dynamics*, *Control Theory*, *Model Checking*. We'll give an overview of these in the

next section. All these three disciplines are complementary to the WIFS methodology. The main two goals of this thesis are:

- To present the WIFS methodology (with precise definitions and in tutorial fashion).
- To show how it can be used in complementary way to three existing quantitative methodologies. We'll apply this methodology on various concrete examples.

For an overview of the fields covered by the methodology, we are interested in the following framework (figure 1.9), derived from [43].

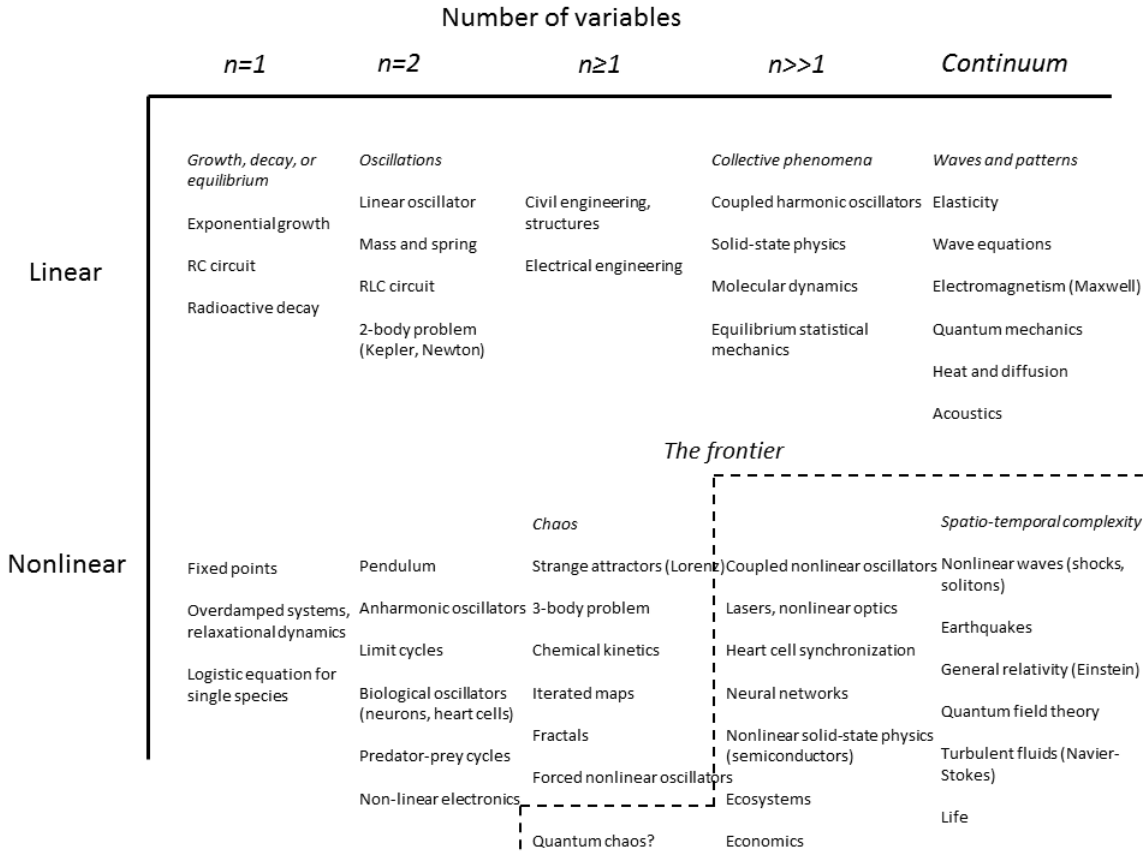


Figure 1.9: A system overview by Steven H. Strogatz

The X-axis of this framework tells us the dimension of the system, e.g. the number of variables needed to charachterize the state of the system. Note the difference between the values on X-axis: the two first columns ($n = 1$ and $n = 2$) includes systems that contains *exactly* one and two variables, respectively. The third columns ($n \geq 1$) are systems that usually contains variables greater or equal to 1, as fractals (2, 3 or more variables according the dimension) or Lorenz system (3 variables). The fourth column ($n >> 1$) includes systems that contains a huge number of variables (like neural networks [4]) despite being continuous systems.

The Y-axis differentiates linear and nonlinear systems. There is in each classification a set of system samples.

Let analyze two of them: exponential growth and pendulum. These examples are also taken from the Steven Strogatz 's book : *Nonlinear Dynamics and Chaos*.

Population growth

For this first example, let consider the exponential growth of a population of organisms. This is a first order differential equation

$$\dot{x} = rx$$

where x is the population at time t and r is the growth rate. This system is classified in the column labeled *n=1* because one piece of information -the current value of the population x- is sufficient to predict the population at any later time. The differential equation $\dot{x} = rx$ is linear in x, so finally the system is classified in the upper left-hand corner of the framework.

Swinging of a pendulum

This second example is governed by

$$\ddot{x} + \frac{g}{L}\sin x = 0$$

There is an additional parameter compared to the previous example. The state of the system is given by two variables : the current angle x and the angular velocity $\dot{x}$. Because these two variables are needed to find the state, the pendulum swinging is placed in the *n=2* column. And because this system is nonlinear, this system is placed in the lower half of the second column.

We can deduce that the simplest systems are located in the upper left-hand corner. The upper right-hand corner is the domain of classical applied mathematics and mathematical physics. There is a region circumscribed by *the frontier*. In this region this is really the jungle. The topics are only partially explored and according to Strogatz they *lie at the limits of current understanding*. These problems are nonlinear and contains a large number of parameters.

As said previously , the scope of this work is the set of all discrete systems, including nonlinears. Most of the examples covered are nonlinear because we can better appreciate the power of these methodology. Based on the figure 1.9 we can deduce that the methodology covers the four first columns.

1.4 Relationships with complementary techniques

We are driven to ask the following question : *Why not use an already existing approach?* There are two models to which the question refers: *System Dynamics* and *Control Theory*. Both have advantages and drawbacks. Firstly we will briefly present them.

1.4.1 System Dynamics

The great advantage of this modeling is that it considers the notion of feedback loops. Using this methodology, we can precisely know the evolution of a parameter through a variation of others.

According to Wikipedia, System Dynamics is *a methodology and mathematical modeling technique for framing, understanding, and discussing complex issues and problems.*

System Dynamics is part of systems theory. This is an approach to understand the behavior of complex systems over time. It considers the internal feedback loops and delay effects that affect the overall behavior of the system. It is based on models that are a formalization of assumptions about the user's system. It allows robust simulation, *if* the modelization is precise enough.

In System Dynamics, running a simulation is solving mathematical equations to obtain the value of each variable over time. The equations contain parameters that must be calibrated often on historical data. The output of a simulation for a given set of input data is called a *scenario*.

This technique allows mathematical modeling to understand and analyze complex problems. It was developed in the 50s to help business managers improve their understanding of industrial processes. Since the 90's it exists software tools to understand system dynamics with appropriate user interfaces, like STELLA[2]. They can solve problems by calculating incrementally each variable on very short time intervals.

It is interesting to see how a system reacts to an external perturbation. This stress will trigger a series of positive and negative feedback that will ultimately magnify or counteract the initial stimulus.

[2]more info: http://www.iseesystems.com/

Example: System Dynamics

This example comes from the excellent book *Dynamic Modeling of Diseases and Pests* of Bruce Hannon and Matthias Ruth[13]. This model shows the evolution of an epidemic if people lose their immunity.

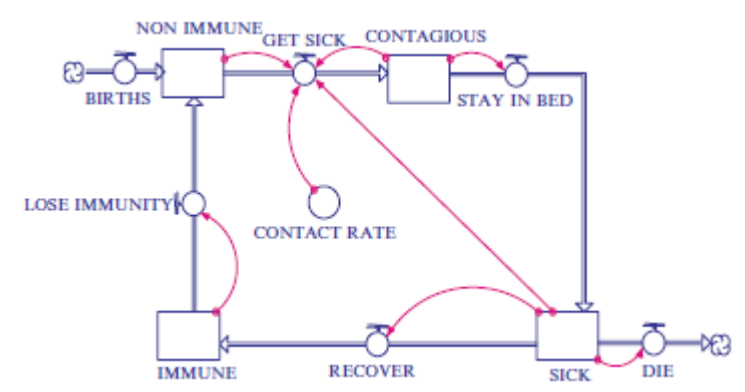


Figure 1.10: Corresponding model

After the first occurrence, loss of immunity dampens the effect of epidemics and are less severe. In the long term, the number of sick persons is constant. That number is typically larger than would be the case of permanent immunity.

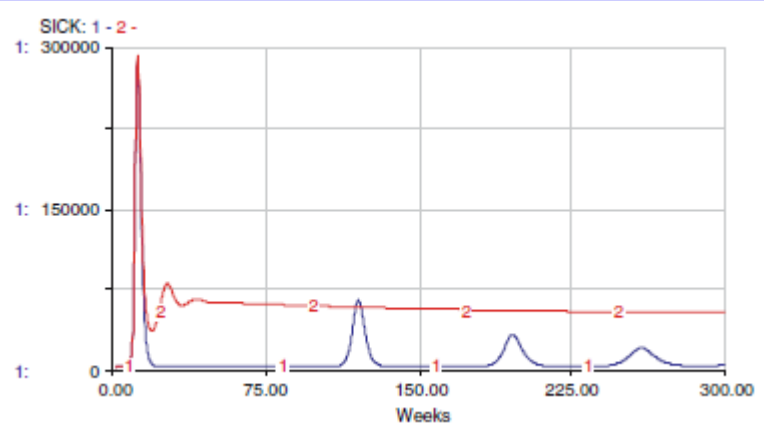


Figure 1.11: Loss of immunity. Blue line (1) represents sick population in the presence of immunity. Red line (2) represents sick population with a loss of immunity.

STELLA

This is the previous example in STELLA:

```
LOSS OF IMMUNITY
----------------
CONTAGIOUS(t)=CONTAGIOUS(t-dt)+(GET SICK-STAY IN BED) * dt
INIT CONTAGIOUS = 1 {Individuals}
INFLOWS:
GET SICK = CONTACT RATE * (CONTAGIOUS + SICK) * NON IMMUNE
{Individuals per Time Period}
OUTFLOWS:
STAY IN BED = CONTAGIOUS {Individuals per Time Period}

IMMUNE(t) = IMMUNE(t - dt) + (RECOVER - LOSE IMMUNITY) * dt
INIT IMMUNE = 0 {Individuals}
INFLOWS:
RECOVER = .9*SICK {Individuals per Time Period}
OUTFLOWS:
LOSE IMMUNITY = .1*IMMUNE

NON IMMUNE(t) = NON IMMUNE(t - dt) + (BIRTHS +
LOSE IMMUNITY - GET SICK) * dt
INIT NON IMMUNE = 1000000 {Individuals}
INFLOWS:
BIRTHS = 5000 {Individuals per Time Period}
LOSE IMMUNITY = .1*IMMUNE
OUTFLOWS:
GET SICK = CONTACT RATE * (CONTAGIOUS + SICK)*NON IMMUNE
{Individuals per Time Period}

SICK(t) = SICK(t - dt) + (STAY IN BED - RECOVER - DIE) * dt
INIT SICK = 0 {Individuals}
INFLOWS:
STAY IN BED = CONTAGIOUS {Individuals per Time Period}
OUTFLOWS:
RECOVER = .9 * SICK {Individuals per Time Period}
DIE = .1 * SICK {Individuals per Time Period}
CONTACT RATE = .000002 {1/(Number of Contagious + Sick) *
Nonimmune) per Time Period}
```

1.4.2 Control Theory

Control theory is the set of techniques to control a physical quantity (temperature, velocity, pressure, ...), without human intervention, to maintain a given value. It includes the study of the behavior of dynamic systems based on parameterized trajectories of their parameters. Control theory has a mathematical foundation with many theoretical results *if* the systems obey certain formal rules.

It's better to illustrate this theory with an example, taken from *Boris J. Lurie* and *Paul J. Enright* [26]

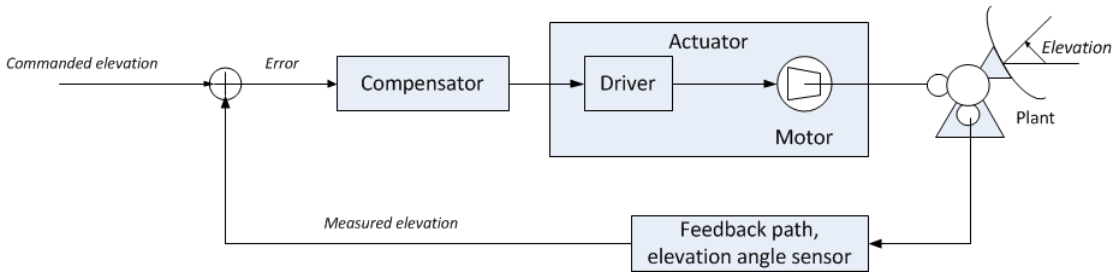


Figure 1.12: Servomechanism of a single-loop feedback system

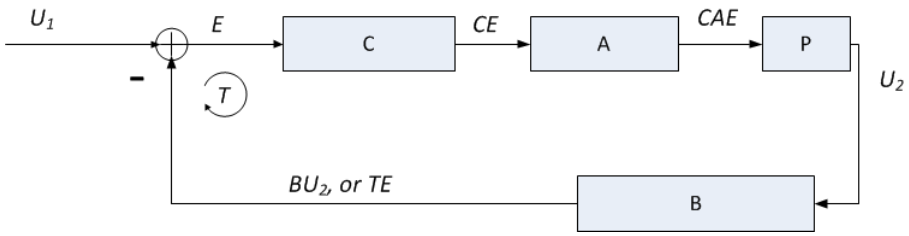


Figure 1.13: Block diagram

The first figure represents the servomechanism that regulates the elevation of an antenna. The second figure shows the block diagram for this system of control. We represent it as a cascade of elements, ie *links*. The capital letters stand for the signals' Laplace transforms and also for the transfer finctions of the linear links. This block diagram shows a single-input single-output (SISO) system. There is one input command, U_1, which is the commanded elevation angle, and just one output U_2, which is the actual elevation of the atenna. Evidently, there is one feedback loop, and so the system is also referred to as *single-loop*.

Feedback in control theory: example

The *feedback path* contains some sort of sensor for the output variable and has the transfer function B. Ideally, the measured output value BU_2 equals the commanded value U_1, and the *error* $E = U_1 - BU_2$ at the output of the summer is zero. In practice, most of the time the error is nonzero but small. The error amplified by the *compensator* C is applied to the *actuator* A, in this case a motor regulator (driver) and a motor. The motor rotates the *plant* P, the antenna itself, which is the object of the control. The compensator, actuator and plant make up the *forward path* with the transfer function CAP. If the feedback path were not present, the system would be reffered to as *open-loop*, and the output U_2 would simply equal the product $CAPU_1$. The *return signal* which goes into the summer from the feedback path is TE, where the product $T = CAPB$ is called the *loop transfer function*, or the *return ratio*.
The output of the summer is

$$E = U_1 - ET$$

so that the error

$$E = \frac{U_1}{T+1} = \frac{U_1}{F}$$

where $F = T+1$ is the *return difference* and its magnitude $|F|$ is the *feedback*. It is seen that when the feedback is large, the error is small.

1.4.3 Models comparison

We will now make a brief comparison between the two models discussed in this section and design with WIFS.

	System Dynamics	*Control Theory*	*WIFS*
Good for discrete systems	no	yes	yes
Good for continuous systems	yes	yes	no
Reasoning	quantitative	quantitative	qualitative
Loop Enabling/Disabling	no	yes	yes
Mathematical Complexity	easy	hard	easy
Local vs Global	local	local	global

Based on this table, we can infer which technique is best suited according to the system to design. Notice that as the control theory is extremely complex from a mathematical point of view, we prefer to use the WIFS when we deal with nonlinear systems.

	Discrete Systems	Continuous Systems
Linear	*Control Theory / WIFS*	*Control Theory / System Dynamics*
Nonlinear	*WIFS*	*System Dynamics*

Regarding the WIFS, the main challenge is to show that qualitative reasoning is rigorous. If we can do that, we can use the techique of WIFS in nonlinear systems, thus saving considerable time to design compared to the theory of control that involves a lot of mathematical resources. Indeed, this thesis is led to show the relationship between a *global qualitative* methodology (using WIFS) and several *local quantitative* methodologies (control theory, system dynamics, model checking).

1.4.4 What about Model Checking?

Model Checking is a family of techniques for automatic verification of dynamic systems. The aim is to check algorithmically whether a given model, the system itself or an abstraction of the system satisfies a specification, often formulated in terms of temporal logic. Model checking allows verification (making sure that the system satisfies certain safety properties) *if* the formalization is precise enough.

We'll discuss it later in the chapter 5. But the main idea is that we'll need this method to check the validity of our qualitative reasoning in our approach to design systems with feedback structures.

1.5 Approach and Organisation

The approach of this dissertation is a qualitative reasoning -but rigorous- on the design rules of systems designed with weakly interacting feedback structures.

We will analyse examples not only in Computer Sciences but also in other areas such as Biology or Business. We will focus very specifically on adaptive systems.

Here is the structure and organization of this paper:

Chapter 2 introduces the basic concepts, which means that we will see all the concepts and definitions necessary to know in to pursue in this work. There will be also a historical section devoted to cybernetics.

Chapter 3 deals with Weakly Interacting Feedback Structures, it will explain exactly what WIFS are. This will treat in depth, in technical details, what we have previously introduced in section 1.2.

Then we will face the central chapter of this book, Chapter 4. This chapter set the trial of methodology which this work treats. The finality of this chapter is to develop a set of heuristics to guide the process to the design systems with feedback structures.

Chapter 5 deals with System Validation. This is to show, demonstrate and discuss what we have advanced and found in the previous chapter.

Chapter 6 consists of a series of concrete examples where the heuristics and rules from Chapter 4 are applied. As said before, there will be not only examples dealing with computer sciences topics, but also dealing with other domains.

The final chapter, Chapter 7, is also an application of rules, but it is entirely dedicated to the human body, which is a fantastic complex system. Obviously this is not about the whole human body (if that were the case, we should devote an entire volume to it), it's just an introduction which presents the main systems in the human body.

Chapter 2

Background

Contents

This chapter presents the necessary background needed to understand the approach and methodology presented in this work. Some concepts are very useful to be redefined to have a common understanding of all these concepts. Each time we try to have some cohesion when we move from one concept to another.

Firstly we will define all the concepts around the system. In the same section, we'll also define and formalize the notion of *state machine*. After that, we will see the concept of abstraction, a key discipline to fully master the design with WIFS and conceptualizing systems. The third section is dedicated to the robustness, which is a keyword in the title of my master's thesis. The fourth section is about nonlinearity and chaos in detail and the reasons for the difficulty to design such systems. The fifth section is derived from the observation that there are often irregularities and unexpected events in large systems. And finally the last section recounts the historical development that led cybernetics to examine the design of systems and more particularly of *autonomic systems*.

2.1 Systems

A *system* can be defined as a set of elements interacting with each other according to certain principles or rules. A system can be open or closed in a given area, depending on whether or not directly interacts with its environment.

In this section we will see different types of systems but above all we will start with the notion of *state machine*.

2.1.1 State transition system

A state transition system is a model of abstract machine. It consists of a given set of states, and a set of transitions from one state to another, which can be labeled from a set of labels, same label can appear on several transitions . If this set is a singleton, the labeling may be omitted.

Note three important points:

- The set of states can be infinite or even uncountable.
- The set of all transitions can be infinite or even uncountable.
- A finite automaton has an initial state and a set of final states.

The state-transition systems can be represented as directed graphs.

State transition Formally we define a *state transition* (unlabelled) as a couple $(S, \rightarrow)$ with $\rightarrow \subset S \times S$ where S is the set of states, and $\rightarrow$ is the transition relation. If p and q are two states, $p, q \in \rightarrow$ means there is a transition from p to q and is denoted as $p \rightarrow q$

There is no assumption on S and $\rightarrow$, they may be infinite, even uncountable. However, if S is finite (and hence also $\rightarrow$), the transition system is a directed graph.

We can also give a definition of labeled transition system: at that time, we must find a set of labels Λ, and take $\rightarrow \subset S \times \Lambda \times S$. The transition system is then the triplet $(S, \Lambda, \rightarrow)$. If there is a transition labeled by $\lambda \in \Lambda$ between two states p and q, then we note $p \xrightarrow{\lambda} q$.

Finite state machine In the case where S and Λ are finite, one may refer to finite state machines (in general, we will also give a condition for accepting input word, which is often the data of two subsets of S that will be initial states, and accepting states).

Deterministic System The transition system is called *deterministic* if and only if $\rightarrow$ is a *function*. It is *non-deterministic* otherwise.

Example: State Diagram

This example comes from Wikipedia[a].
S_1 and S_2 are states and S_1 is an accepting state or a final state. Each edge is labeled with the input. This example shows an acceptor for strings over $\{0, 1\}$ that contain an even number of zeros.

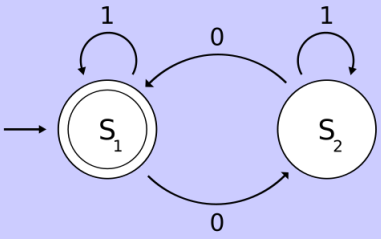


Figure 2.1: This diagram is also called *Moore machine*

[a]http://en.wikipedia.org/wiki/State_diagram

2.1.2 Complex systems and system overview

A complex system is a set consisting of many interacting entities. These entities can allow the observer to predict the system feedback, behavior and evolution changes, by *calculation*. When we want to model a system, we conceive a number of evolution rules, then the system is simulated by iterating the rules to obtain a structured result.

A system is *complex* if the final result is not directly predictable by knowing the rules. In other terms, despite a thorough knowledge of the elementary components of a system, it is now impossible to predict its behavior, other than by experience or simulation.

This limitation comes from the impossibility of putting the system into resolvable and predictive equations. What is crucial is the number of parameters, and the fact that each one can have a major influence on system behavior. To predict this behavior, it is necessary to take all of them into account, which is to run a simulation of the system studied.

A complex system is a system composed of many entities in local and simultaneously interaction. It often requires that the system has more of the following specifications (which shows that there is no widely accepted formal definition of what exactly a complex system is):

- The interaction graph is not trivial: it's not just everyone interacts with everyone (there are at least special relationships).
- The interactions, as well as most of the information, are local. There are few central organization.

- There are feedback loops. Feedback loops can be a cause of the system behavior's nonlinearity.

Most of the time the complex systems have the following characteristics in addition:

- Interactions of components together form *groups* of components strongly linked, each *group* is in interaction with others. It allows to model the complex system by levels: each component interacts locally with a limited number of components .
- The components can themselves be complex systems (different levels): a society can be seen as a system composed of interacting individuals, each individual can be seen as a system composed of interacting organs, each organ... We'll cover this in more detail in the next section.
- The system acts on its environment, we say that the system is *open*.

Finally, it would be interesting to show the following illustration, created by Hiroki Sayama[1], :

[1]Hiroki Sayama, D.Sc., Collective Dynamics of Complex Systems (CoCo) Research Group at Binghamton University, State University of New York

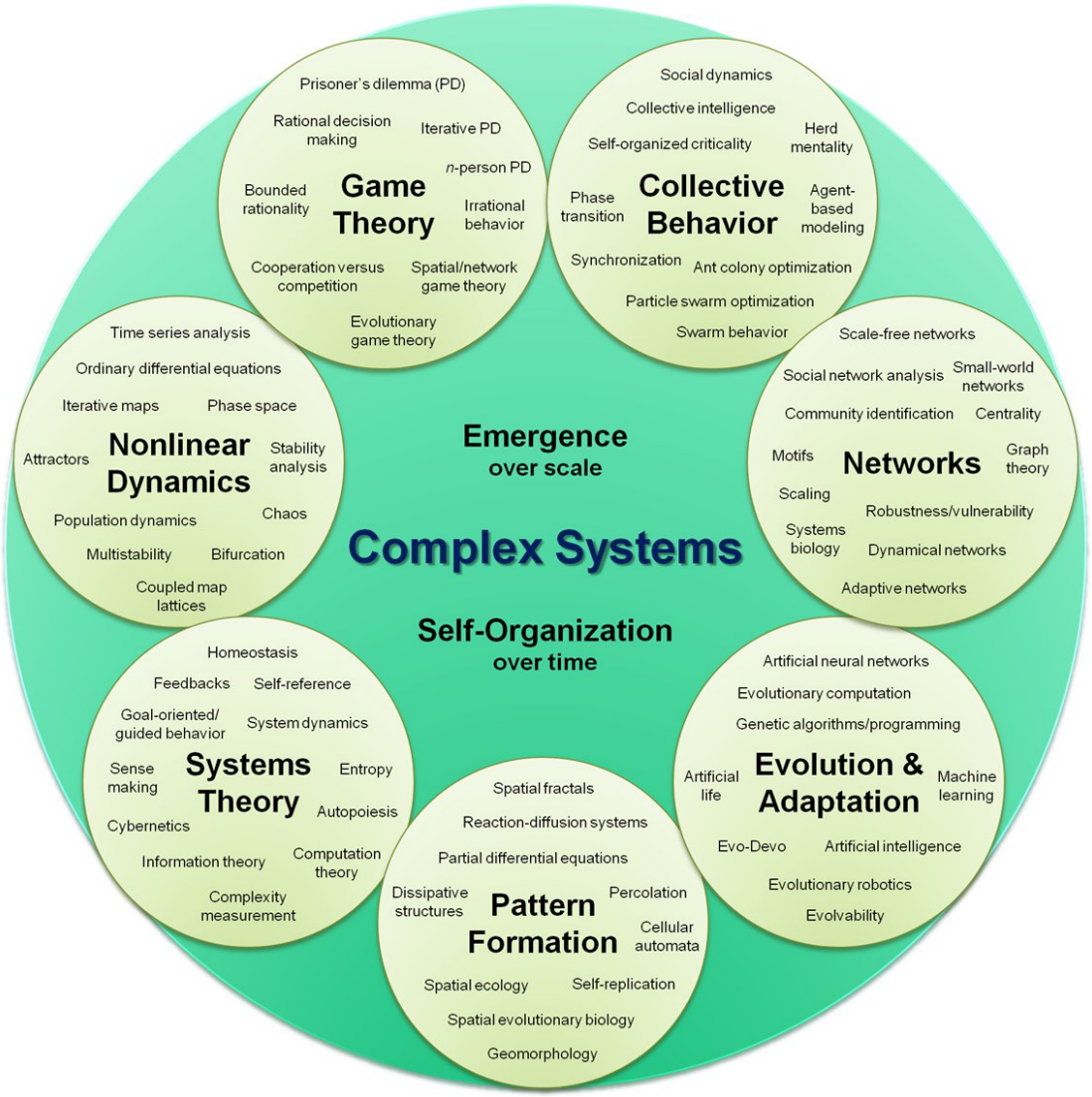


Figure 2.2: Organizational map of complex systems

This is a visual, organizational map of complex systems broken into seven sub-groups.

2.2 Abstraction and Emergence

As for computer science (eg the OSI model), the abstraction is a matter important in understanding how the system works. We will discuss two key concepts related to abstraction: *behavior* and *emerging properties*. To introduce this concept, note that the abstraction is present everywhere in systems. For example, in the respiratory system, conscious control does not need to know details of breathing reflex. Also in this example, note that we also have a layer hierarchy, as the OSI model, with *component* $\rightarrow$ *feedback loop* $\rightarrow$ *feedback structure* $\rightarrow$ *weakly interacting feedback structures*.

Behavior The behavior of a system is the part of its activity which manifests itself to an observer.

2.2.1 Emerging properties

Emerging properties refer to the appearance of new characteristics to a certain degree of complexity. We can define the emergence by two characteristics:

- The whole is more than the sum of its parts. This means that you can not necessarily predict the behavior of the entire by the analysis of its parts.
- The whole adopts a behavior whom detailed knowledge about the parts does not provide full information about the whole.

Let illustrate this with an image of *Stewart* and *Cohen*[9]. Ian Stewart and Jack Cohen show that the concept of emergence is a crossing point to explain the macroscopic properties that can not relate to properties of components alone, and so on: indeed, If it is evident that cats are strongly attracted to the mice, it seems absurd to infer that the cat 's molecules are directly attracted by the mice 's molecules. The cause of this attractiveness must therefore be sought in the internal organization of these, or even more complex structures such as those of the nervous and hormonal systems.

Extract: autonomic elements and social intelligence

This passage is taken from article by Jeffrey O. Kephart and David M. Chess, *The Vision of Autonomic Computing*[21]. It is a metaphor for the emergence of the behavior of an autonomic system to the emergence of social intelligence from a colony of ants.

> *Autonomic systems will be interactive collections of autonomic elements, individual system constituents that contain resources and deliver services to humans and other autonomic elements. Autonomic elements will manage their internal behavior and their relationships with other autonomic elements in accordance with policies that humans or other elements have established. System self-management will arise at least as much from the myriad interactions among autonomic elements as it will from the internal self-management of the individual autonomic elements, just as the social intelligence of an ant colony arises largely from the interactions among individual ants.[...]*

Indeed, in social insects such as ants or termites, it appears an emergent behavior, overall effect resulting from the application of local rules. Studies by ethologists have shown that some collective behavior of social insects were self-organized. Self-organization characterizes processes in which structures emerge at the collective level, from a multitude of simple interactions between insects, without being explicitly coded at the individual level. For example, the fact that a termite has more chance to make a divot in a place where there are already, this will arise the construction of a mound in a group of termites.

2.2.2 Phase diagrams

A *phase* of a complex system consisting of many interacting components is an area of the operating space in which the aggregate behavior of the components can be characterized concisely. For example, each component consisting of a specific system, may be in the same state and have the same dominant set (same behavior). Often, the components have identical specifications, but this is not always the case.

Phases appear in many complex systems consisting of large numbers of interacting components, not just in physical systems (such as water) where the components are molecules or atoms, but also in computing systems (such as peer-to-peer networks) where the components are software components, and also in social systems (such as human organizations) where the components are human beings. Different parts of the system can be in different phases. The important point is that phase transitions, where the system changes phase, require no global coordination, but only local interaction between nearby components. Boundaries between phases can be sharp or

diffuse, and critical points may occur. A critical point (we will discuss this topic in more detail later) is a small part of the operating space in which tiny changes in operating conditions can result in large parts of the system changing phase.

A phase diagram is a graphical representation used in thermodynamics representing the fields of the physical state of a system based on variables chosen to facilitate understanding of the studied phenomena.
When all phases are represented in different physical states, we talk about *change of state diagram.*
Illustrate this with an example for water[2]:

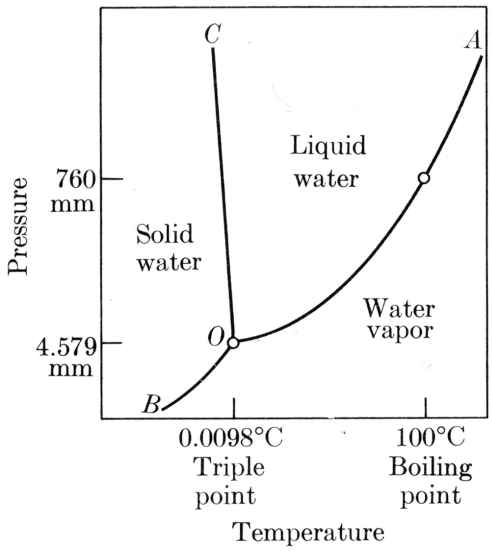


Figure 2.3: Phase diagram for water

- The *triple point* is the point where the three phases coexist which happens at a specific temperature and pressure.
- The curve of change of liquid-vapor stops at a point called *critical point* (not shown on the picture) beyond which the water has only a single fluid phase, rather closest (in terms of its physical properties) of a gas for pressures below the critical pressure, rather close to a liquid at pressures above the critical pressure.

It is important to understand the workings of phase diagrams because, in a system, for each phase or phase transition, a specific behavior will be appropriate.

2.3 Robustness and Scalability

The robustness of a system is defined as the stability of its performance. In computer science, it means that the system does not crash at the slightest disturbance.

[2]http://www.earth.northwestern.edu/people/seth/202/new_2004/H2Ophase.html

2.3.1 Three Laws of Scalability

According to Peter Van Roy [38], a system is *scalable* if it is able to handle growing amounts of work in an acceptable manner. There are three laws of scalability:

1. New things happen at each new scale.
2. In the limit of increasing scale, large systems have only local control.
3. (the CAP theorem)Pick any two of consistency, availability, and partition tolerance.

We have so far met the first two laws in our examples, from respiratory system (among others).
We will now discuss these three laws. This discussion is also inspired by the slides presentation of Peter Van Roy [38].

First law : *New things happen at each new scale.* It means the at each new scale, the situation changes. We can apply this law to physics for example. At each new level of energy, new laws and new physics appears (see box below). This example is inspired from an essay of the *Absolute Motion Institute* : The Joules of the Universe[3].

Scalability in nature

The diagram on the next page shows all the units of energy found in nature. The smallest (stationary photon) to the largest (total output since the creation of the universe).
Note the scale change: indeed, firstly, at 10^{-38} joule, the unit is considered as energy of an individual photon (such as radio waves, television waves, the light spectrum, etc...). Then from 10^{-12} joule of energy the next element, hydrogen fusion, does not take into account the photons but the kinetic energy of the nuclei of helium and hydrogen speeding away from one another. It's the same with nuclear fission of $Uranium_{235}$ or the proton-antiproton annihilation.
The next few items on the scale are units of energy used in science and commerce. The *erg* is the energy of a gram of mass moving at a velocity of 2 centimeters per second. A *foot pound* is the energy of one pound moving at 2 feet per second. A *calorie* is the amount of energy required to raise the temperature of one gram of water one degree Celsius.[16]
And so on ... This example clearly illustrates our first physical law: Novelty at Each Scale.

[3] http://www.circlon-theory.com/HTML/joules.html

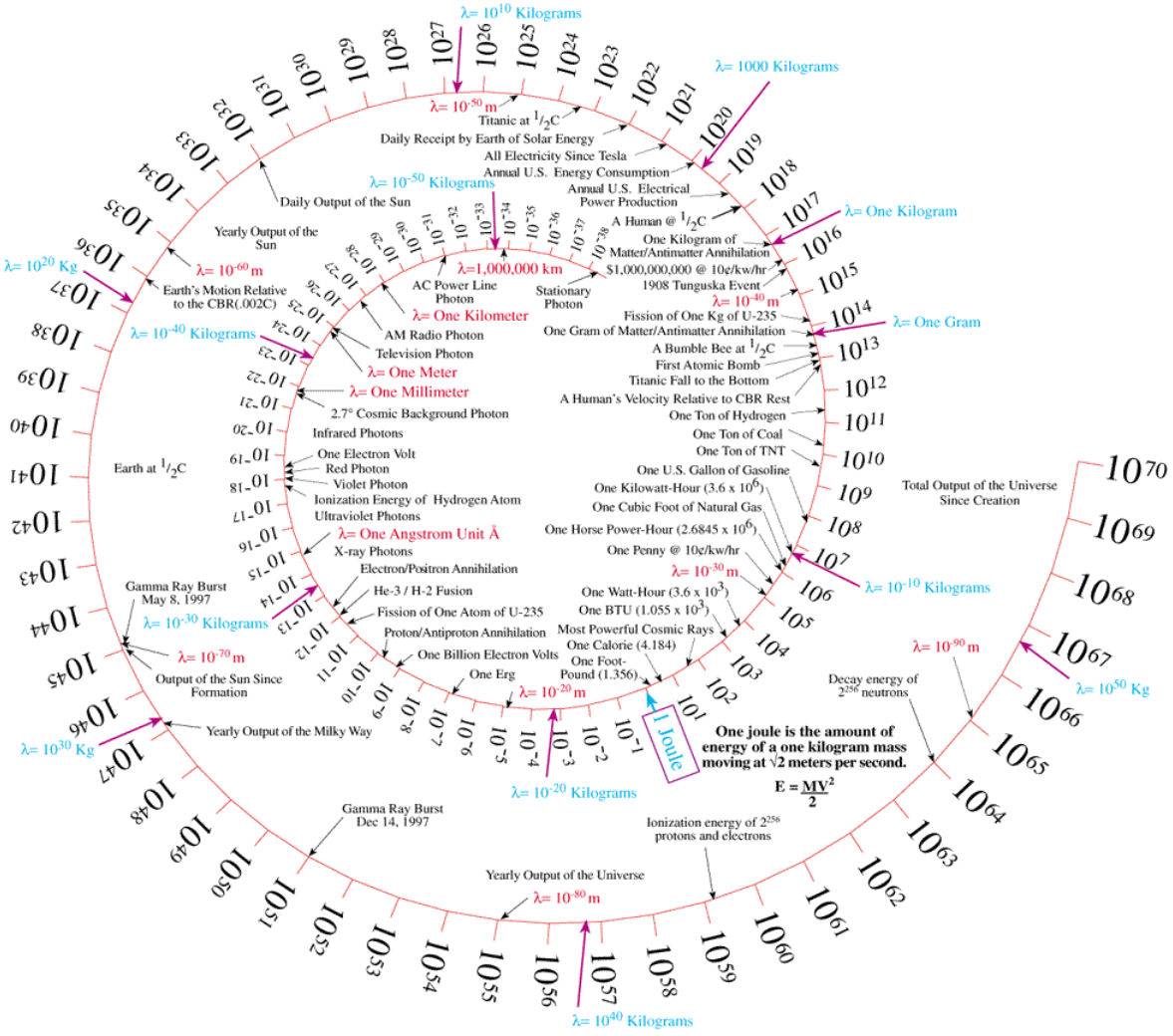


Figure 2.4: The Joules of the Universe. This diagram shows the energy transfer of a number of familiar events over the whole energy spectrum.

Note that a problem remains: What about WIFS introduced in section 1.2? How can we make the correspondence between it and the first law? Indeed, it seems that in this case there is only one level of scale.

The solution follows from the second and third law: considering that there are WIFS structures at each level. Take for example the respiratory system introduced previously and we are interested in a random single component: Measuring CO2 in blood. If we change the level, this component can be a WIFS structure because it can be assumed it is constituted of sensors that are themselves feedback structures (because composed of cells).

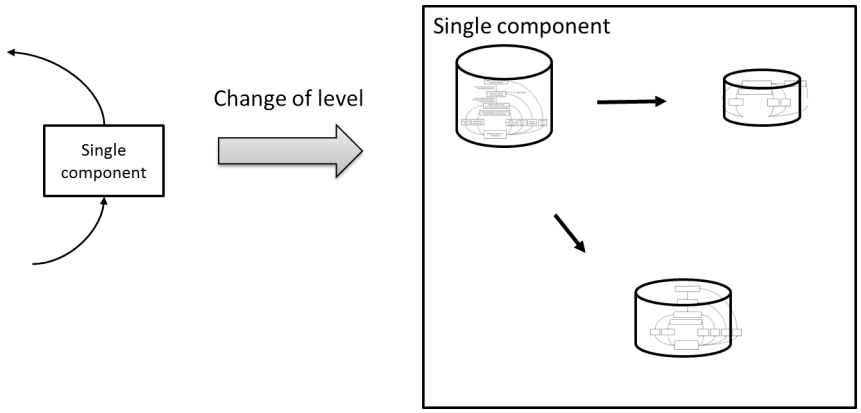


Figure 2.5: Single component as a WIFS

Second law : *In the limit of increasing scale, large systems have only local control.*

The fact that the control is only local, involves apparation of concurrency, nondeterminism and asynchrony. The Asynchrony is due to the fact that the message may take an arbitrary time for transmission. Therefore, this implies the failures are very difficult to detect.

Technical note

Once we have to program these kind of big systems, we must use the right paradigms. Most of the time, the paradigms used will include *message passing* (to simulate asynchronicity). It turns out that the simplest paradigms for concurrent programming are *deterministic dataflow concurrency* and *message-passing concurrency*.

Note that global control may be possible in some cases (less and less as the scale increases). Mostly, it is very expensive or impossible.

But there is not only disadvantages: as the control is local, the failures are local as well. We can therefore consider that the parts of a system are mostly independents, but not quite, there is little interaction between them. Hence the name of *weakly interacting* feedback structures.

Examples: *Mostly* Independent Parts

These examples are from [38], these are examples of large systems consisting of mostly independent parts:

- Gas in a box: molecules mostly independent, occasional interaction when two molecules collide.
- Peer-to-peer network: peers mostly independent, occasional interaction between neighbors only.
- Swarm intelligence: collaborative behavior among large numbers of simple agents (e.g., flocking and swarming). Each agent interacts with only a small number of neighbors.
- Gossip algorithm: nodes mostly independent, occasional interaction between random pairs. Can efficiently solve many global problems such as diffusion, search, aggregation, monitoring, and topology management such as *T-Man* algorithm where the topology emerges progressively through the cycles.

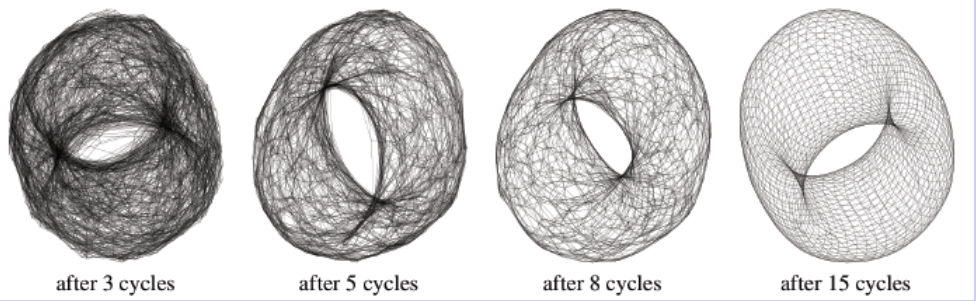


Figure 2.6: T-Man algorithm. Each node periodically picks a random node and exchanges information with it

Third law (the CAP Theorem) : *Pick any two of consistency, availability, and partition tolerance.*

The fact is that for all systems, at all levels of abstraction, and at all sizes, it is impossible to implement an object that guarantees the following properties in all fair executions:

- Consistency: all operations are atomic (totally ordered)
- Availability: every request eventually returns a result
- Partition tolerance: any messages may be lost

According to the theorem, the system can only satisfy two of them at the same time.

2.3.2 Link between Scalability and Robustness

It goes without saying that greater scalability leads to a greater robustness. In fact, if the system adapts his behavior and maintains its functionality and performance according to the demand, it implies that the system has less misfortune to crash at the slightest disturbance.

Actually there is no comprehensive approach to design system robustly. In my opinion, this is a set of elements to consider: scalability, reliability, flexibility, sustainability, upgradeability and reusability. Among these, we focused on the scalability because according to the *CoreGrid* Technical Report on self management in distributed systems [54], *[...] it is not possible to achieve scalability without first achieving self-management, since a large-scale distributed system will otherwise be completely unmanageable.* This is why, by following some of the guidelines of this work (ie self-management systems), we grant much importance to scalability.

2.4 Non-Linearity and Chaos

In Section 1.3, we have already discussed the organization and classification of systems. Now we'll focus on chaotic systems.

As you can see on the Strogatz mapping, chaotic systems are nonlinear. Actually, most of real life problems imply nonlinearity. According to Wikipedia[4], a *nonlinear system* is any problem where the variable(s) to be solved for cannot be written as a linear combination of independent components.

Nonlinear systems can exhibit completely unpredictable behavior, which may even seem random (although it is a completely deterministic systems). This unpredictability is called *chaos*.

[4] http://en.wikipedia.org/wiki/Nonlinear_system

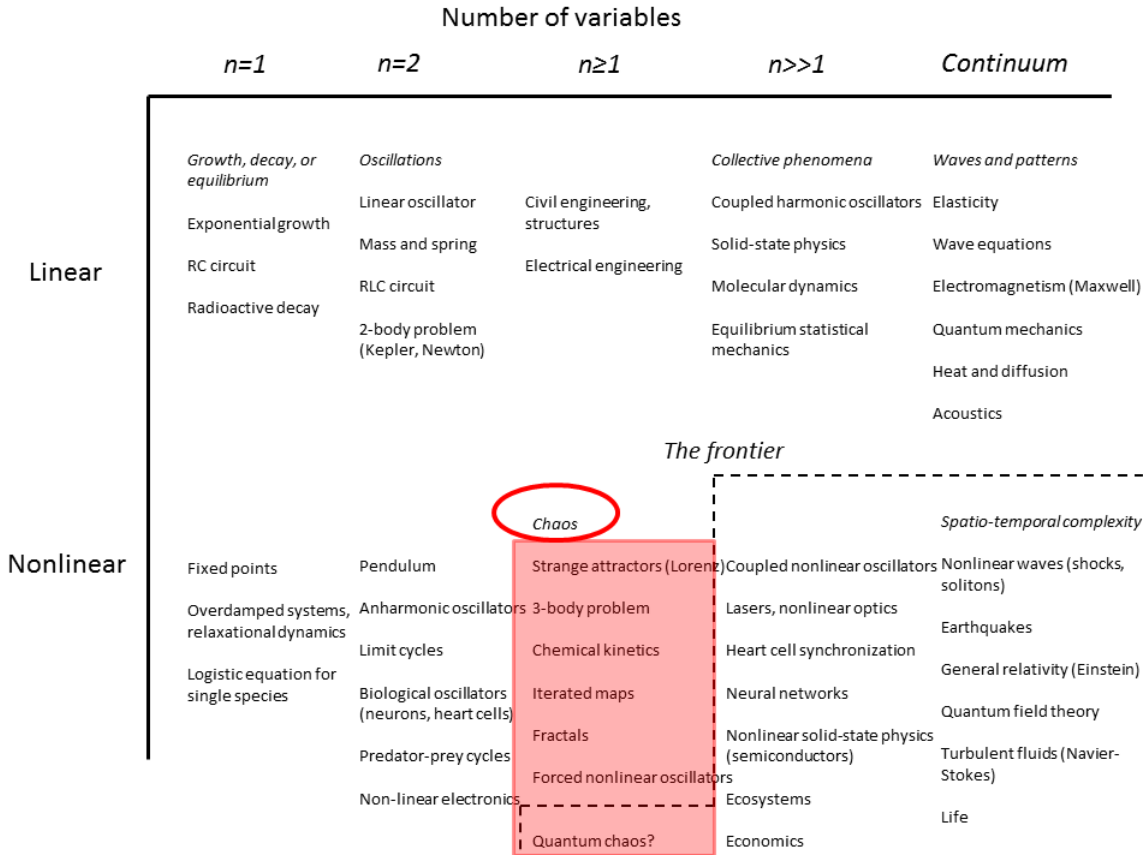


Figure 2.7: Chaos theory focuses mainly on the description of these systems with a small number of degrees of freedom, often very simple to define, but whose dynamics appears as messy

In his book[43], Strogatz tries a definition for *chaos*:

Chaos *is aperiodic long-term behavior in a deterministic system that exhibits sensitive dependence on initial conditions.*

He states that so far no definition of the term chaos is universally accepted. But almost everyone would agree on these three ingredients:

1. *Aperiodic long-term behavior* means that there are trajectories which do no settle down to fixed points, periodic orbits, or quasiperiodic orbits as $t \to \infty$. For pratical reasons, we should require that such trajectories are not too rare. For instance, we could insist that there be an open set of initial conditions leading to aperiodic trajectories, or perhaps that such trajectories should occur with nonzero probability, given a random initial condition.

2. *Deterministic* means that the system has no random or noisy inputs or parameters. The irregular behavior arises from the system's nonlinearity, rather than from noisy driving forces.

3. *Sensitive dependence on initial conditions* means that nearby trajectories separate exponentially fast, i.e., the system has positive Liapunov exponent.

Chaotic System example : *Arnold's cat map*

This function is used to illustrate chaotic behavior in dynamical systems theory. It relates this unusual name because Vladimir Arnold described it in 1967 with the help of a drawing of a cat.
As we are in the computersciences, we are interested in discrete version of this function, on a set of discontinuous points. In practice, we take an image consisting of disjoint pixels. The points have integer coordinates in the range $\{0, ...N-1\}$. We set:

$$\begin{array}{rccc} \psi: & I_{N,N} & \rightarrow & I_{N,N} \\ & P\begin{pmatrix} x \\ y \end{pmatrix} & \mapsto & P'\begin{pmatrix} (x+y) & mod & N \\ (x+2y) & mod & N \end{pmatrix} \end{array}$$

Applying this transformation repeatedly, we see a radically different behavior and surprising. Indeed, we find after a finite number of iterations the original image.

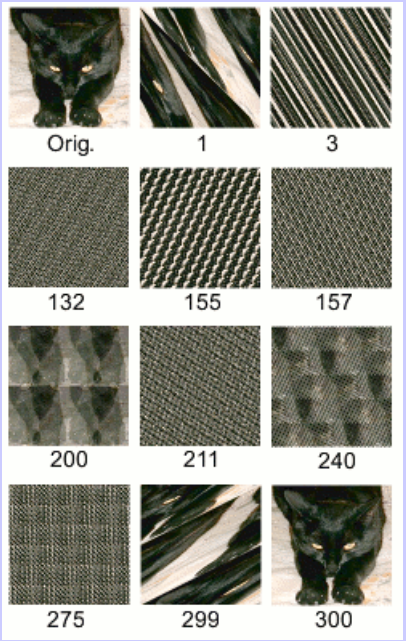


Figure 2.8: Arnold's cat map sample. Back to the original after 300 iterations. ©Claudio Rocchini, 2006

Computer processing is to swap the pixels without losing information; possible permutations of the image being finite in number, it can only fall into a cycle.

It should be pointed that chaos is a form of behavior that often occurs in complex

systems. The system does not have to be very big for chaos to occur, it just has to be nonlinear with a sensitive dependence on initial conditions. In a complex system, chaotic behavior can occur on many scales. In the WIFS methodology, it can occur within a single FS or between several WIFS. Chaos is both good and bad. It is bad because it introduces a fundamental unpredictability in a system's behavior. It is good because it can improve certain characteristics of a system. For example the reaction time of a system can be reduced by keeping the system in a chaotic region of the operating space.

Note

The relationship of chaos to the WIFS methodology is similar to the relationship of complex components to the WIFS methodology. A complex component is often necessary, since it encapsulates an algorithm that can give very good behavior in a part of the operating space. However, the complex component has to be handled carefully, since it can introduce instability. Similarly, a component with chaotic behavior can be very good in part of the operating space, but chaos has to be handled carefully as well, since it can also introduce instability (undesired behavior). Chaos is harder to handle than complex components, since chaos can exist at larger scales. It may exist between several WIFS at the largest scale, or it may exist inside a component. In both cases, it has to be handled carefully.
[Peter Van Roy, 2012].

2.5 Irregularity in Large System

In physics, the analysis of complex systems usually begins by linearization in order to reduce their complexity. This approach is valid for conditions and assumptions precise and restrictive. This leads to quantitative results which allow physicists to get an idea of the processes taking place close to equilibrium. While the system is close to its equilibrium point, it is possible to keep the commonly used techniques for linear systems to get answers about the behavior of the nonlinear system[5]. But this is inefficient when you move away a little bit of an equilibrium point.

A nonlinear system involves in fact, in most cases, unpredictable behavior, as can be the reversal of Earth's magnetic field. Physics has developed techniques to try to control or to enable a better understanding of certain nonlinear systems, such as the method Pyragas based on a series of feedback loops[3]. So for large systems, *abnormal* events (such as failures) are normal occurrences.

As systems become larger, their inherent fragility becomes more and more apparent. Software errors and partial failures become common, even frequent occurrences. This

[5] *Lyapunov stability*

fact implies that software errors can not be completely eliminated and we have to be dealt with [55]. It's the reason why large systems must be designed carefully, otherwise the system will not behave well when stressed.

2.6 History: Cybernetics

It is important to have some historical concepts to understand the motivations and background which eventually led to this master's thesis.

The main thing to note is that a lot of fields have their origin in cybernetics, more precisely, it is Norbert Wiener who laid the foundation in 1948 in his book that founded the discipline: *Cybernetics, or Control and Communication in the Animal and the Machine*[30]. We can define cybernetics as a science of self-regulating systems that is not interested by components or their interactions, but their overall behavior is taken into consideration first. This is a modeling of the relationship between elements of a system, by the study of information and principles of interaction. This is the science made by all theories of process control and communication and control in living beings, machinery and systems sociological and economic.

Cybernetics refers primarily a means of knowledge, which examines information in the sense of physics, in the definition given by Norbert Wiener: *As entropy is a measure of disorganization, the information provided by a series of messages is a measure of organization.* In the first sense, cybernetics is a phenomenological approach that examines the information, its structure and function in systemic interactions. Which can be translated as the general science of control and communications in natural and artificial systems.

Feedback is highlighted by this approach because it is essential to develop a logic of self-regulation. Thus we see the emergence of feedback loops, a mechanisms that connects effect and his own cause, with or without delay.

Most participants in the cybernetic movement are major authors in their discipline. So the concepts are spreading rapidly. Cybernetics marks the time of a major epistemological break that has profoundly influenced all areas of science and its benefits are countless.

As Napoleon says "*Un bon croquis vaut mieux qu'un long discours*" (A good sketch is better than a long speech.), we will therefore illustrate the influence of cybernetics by a diagram representing all disciplines that led from this matter. This map is from Wikipedia[6].

[6]http://en.wikipedia.org/wiki/Complex_systems

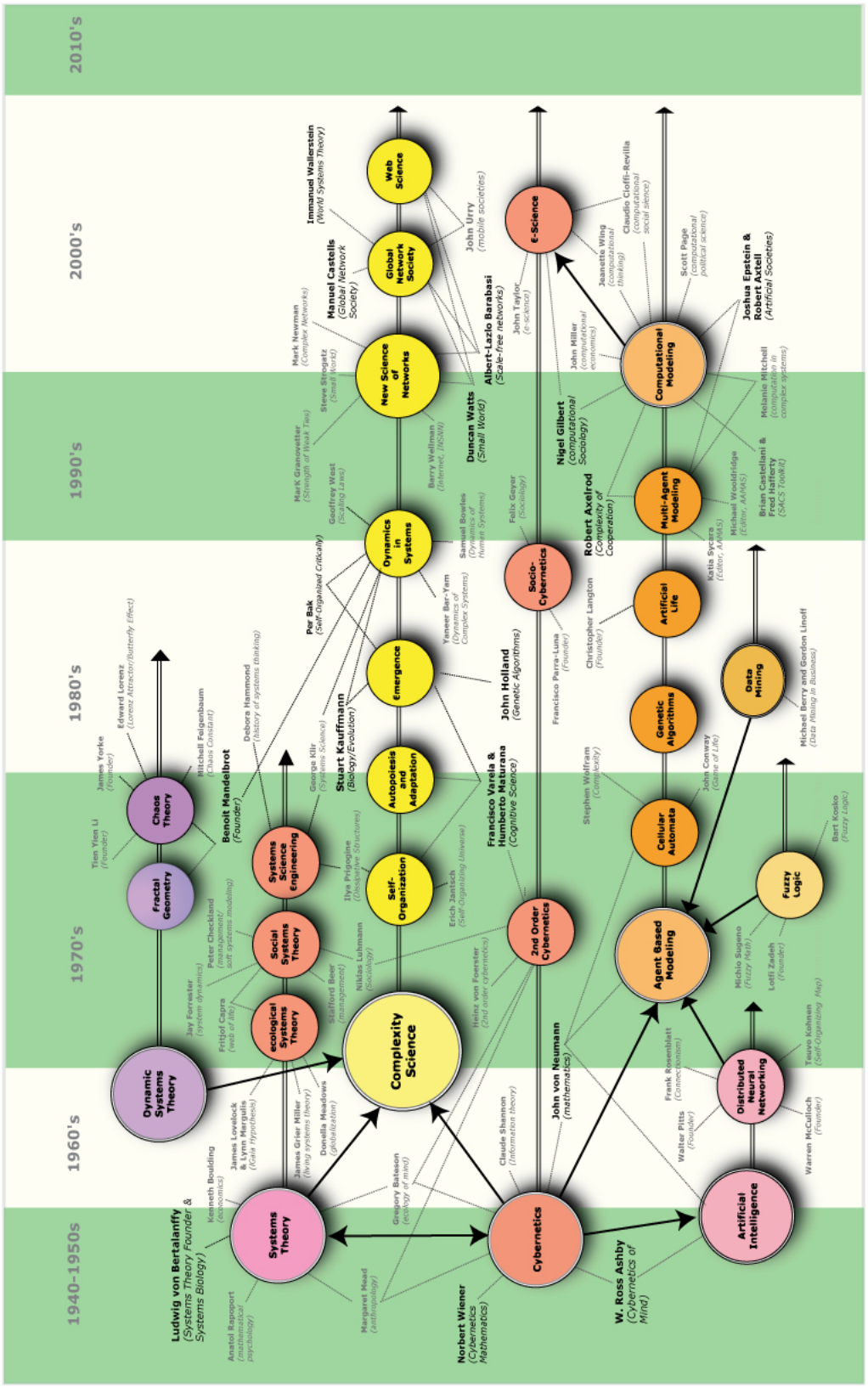


Figure 2.9: "*Everything*" started from Cybernetics

Chapter 3

Design with Feedback Structures

Contents

This chapter is a continuation of what we discussed in the section on feedback thinking (section 1.2). The concepts discussed previously are here more detailed.

At first, we discuss the motivations for using such qualitative structures. Then we will detail the concept of *feedback loop* and *feedback structure* and analyze the interactions between the feedback loops. After that, we will see how a system model as a feedback structure is linked with state diagram. Next, and this is a key part, we will group elements discussed above and making the link with the concept of *emergence* and then deducing a formalism. Finally, we analyze a case study with the curve of prey-predator equation (Lotka-Volterra).

3.1 Nonlinearity in the world

Nonlinear effects are effects that do not occur in direct proportion to the action. As we have seen previously, this is the case for a lot of real-world effects, because

the world contains living organisms that are full of nonlinear mechanisms. These interesting systems (including *life* itself) are so much harder to analyze quantitatively than linear ones. The reason is simple: in linear systems, unlike nonlinear, the parts can be analyzed separately and then being combined (superposition principle, compositional systems)[38]. So it is very difficult to accurately reproduce and analyze these nonlinear systems *quantitatively*.

But there's an interesting point: actually the majority of nonlinear systems can be analyzed *qualitatively*. How? by a combination of geometrical reasoning and some analysis. This theme is out of scope of this master's thesis. However, for more information, Strogatz's book in 1994 *Nonlinear Dynamics And Chaos* is a reference and is highly recommended [43].

The most common nonlinearities are:

Critical threshold Below a certain value, nothing happens. Above, an effect starts.

Saturation Beyond a certain input value, the output value will not change.

Hysteresis Same input value correspond to different values of output depending on whether the input value is increasing or decreasing.

Quantification When the input value increases continuously, the output value only changes stepwise.

Again, you can see how to qualitatively analyze these concepts in Strogatz's book.

What we must know is that any intelligent behavior results from nonlinearity. But the more the system is nonlinear, the more it is fragile. Indeed, if there are many links, if an element is affected by an external event, its neighbors will be too. It follows that the system is often more robust to small local perturbation than it would be without the links. *Complexity theory taught us that many simple units interacting according to simple rules could generate unexpected order*[44]. However, intelligent systems must be as robust as possible. That's why they are made of *weakly* interacting subsystems.

3.2 System as Feedback Structures

This section introduces the various components of the WIFS hierarchy: interacting single components form a feedback loop, feedback loops form a feedback structure,

feedback structures form a WIFS. We will conclude with a simple example to illustrate the concepts discussed in detail. A large proportion of this section is inspired from Alexander Bultot master's thesis, *A survey of systems with multiple interacting feedback loops and their application to programming*[7].

3.2.1 Feedback Loop

A feedback loop consists of three components that interact together with a subsystem. We saw it earlier in the introduction, this is just a reminder. These components are: [7]

- **Monitoring agent:** monitors the state of the subsystem and sends the state information to the *calculating agent*.
- **Calculating agent:** calculates a corrective action to apply to the system and sends this correction to the *actuating agent*.
- **Actuating agent:** applies the corrective action to the subsystem.

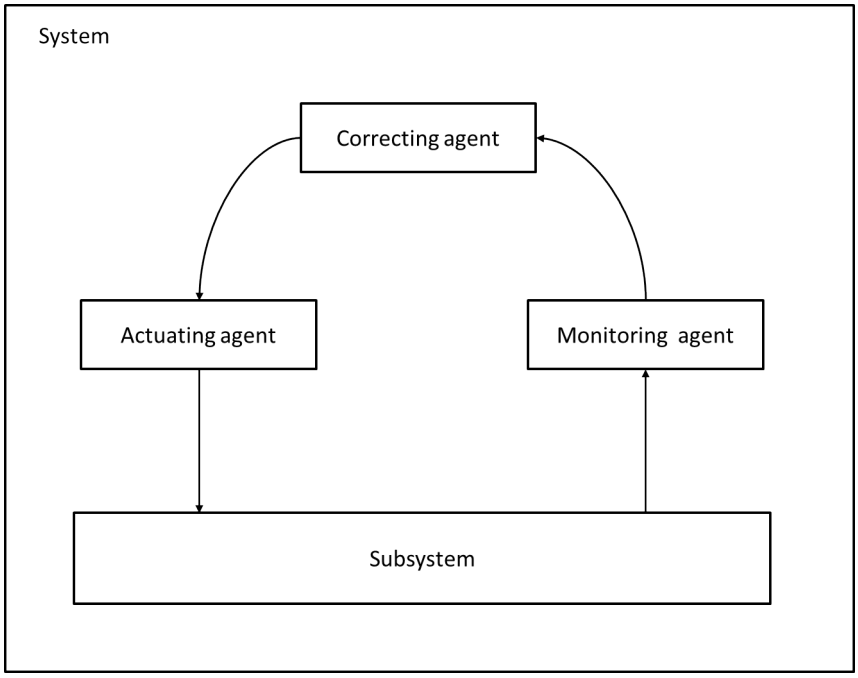


Figure 3.1: Reminder of the general structure of a feedback loop

Loop example : *Freeing memory in the garbage collector*

If we look at Figure 3.1, we can see that the big box on the bottom is the subsystem with which the feedback loop interacts. In the case of the garbage collector, the subsystem is the *System Memory*. The monitoring agent detects when the free memory is below a determined threshold to activate the garbage collector.

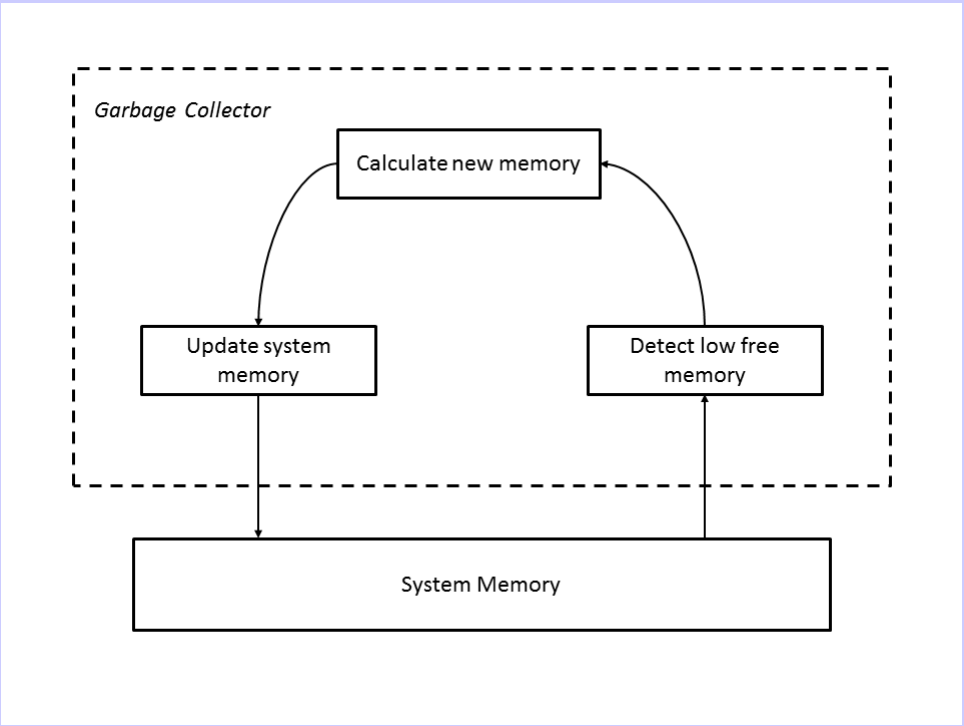


Figure 3.2: Garbage collector operation as a feedback loop

The box on the right (monitoring) detects low free memory. The top box (correcting agent) calculate the new free memory and then the next box (actuating agent) update the *system memory* with the new free memory.

Proactive and Reactive

Regarding the feedback loop, there are two types of activation: a loop can respond to an event by triggering (reactive), or a loop can be triggered itself by anticipating this event (proactive).

We can have both of these activation for the same loop, for example in the garbage collector:

- The threshold of free memory is reached, it causes the garbage collector to be run.

- The running application can decide to invoke the garbage collector directly (it bypasses the detection of low free memory). This is done commonly just before a critical piece of code, to make sure the garbage collector is not run unexpectedly during the critical code.

Note that a proactive loop has an extra agent that anticipates events. That is, instead of receiving an event, this agent can query the state of the resource and if necessary create events for the next agent. [7]

Positive and Negative

We distinguish two types of feedback loops:

- The *positive feedback loops*: the injection of the output data as input facilitates and accelerates the transformation in the same direction, this has the effect of amplifying the behavior of the system. The effects are cumulative (effect *snowball*) and the behavior is divergent, either in the form of exponential growth or explosion, or in the form of an exponential decay which leads to a blockage of the action.
- The *negative feedback loops*: the output act in the opposite direction to previous results. Effects tend to stabilize the system.

The positive feedback loops are those where the variation of an element spreads throughout the loop so that the initial variation has increased, hence the name of *reinforcing loop*.

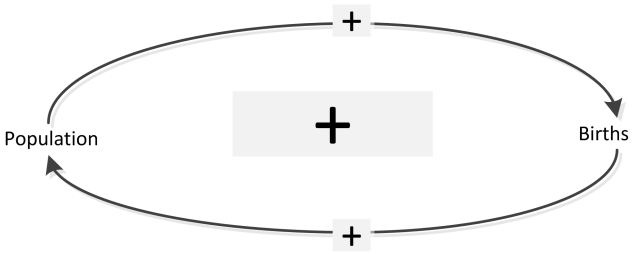


Figure 3.3: A positive feedback loop : births

Feedback is positive if it contains an even number of negative relationships or only positive relationships.

Extract : *Reinforcement Loop*

In his admirable book *The Fifth Discipline*[40], Peter Senge talks about the reinforcement loop:

> *The behavior that results from a reinforcing loop is either accelerating growth or accelerating decline. For example, the arms race produces an accelerating growth of arms stockpiles.[...] Positive word of mouth produced rapidly rising sales of Volkswagens during the 1950s, and videocassette recorders during the 1980s. A bank run produces an accelerating decline in a bank's deposits. Folk wisdom speaks of reinforcing loops in terms such as snowball effect, bandwagon effect, or vicious circle, and in phrases describing particular systems: the rich get richer and the poor get poorer. In business, we know that momentum is everything, in building confidence in a new product or within a fledgling organization. We also know about reinforcing spirals running the wrong way. The rats are jumping ship suggests a situation where, as soon as a few people lose confidence, their defection will cause others to defect in a vicious spiral of eroding confidence. Word of mouth can easily work in reverse, and (as occurred with contaminated over-the-counter drugs) produce marketplace disaster.*

In the negative feedback loop, a variation on an element is transmitted throughout the loop so that it determines a variation of opposite sign on the element itself. The behavior of these loops is characterized by self-corrective action. Any variation produced on one of the elements of the loop aims to cancel out. Such kind of loop tends asymptotically to bring the corresponding structure in a state of equilibrium, hence the name of *balancing loops*.

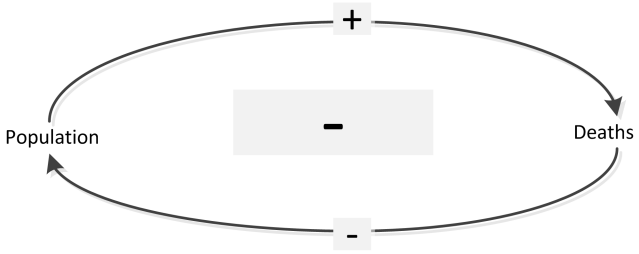


Figure 3.4: A negative feedback loop : deaths

A feedback loop is negative if it contains an odd number of negative relationships. It means there is a relationship between two elements where a change in the *cause* variable generates a variation in the opposite direction of the variable *effect*.

Extract : *Balancing Loop*

Another extract from Peter Senge[40]

> *Balancing feedback processes are everywhere. They underlie all goal-oriented behavior. Complex organisms such as the human body contain thousands of balancing feedback processes that maintain temperature and balance, heal our wounds, adjust our eyesight to the amount of light, and alert us to threat. A biologist would say that all of these processes are the mechanisms by which our body achieves homeostasis - its ability to maintain conditions for survival in a changing environment. Balancing feedback prompts us to eat when we need food, and to sleep when we need rest, or -as shown in the diagram above- to put on a sweater when we are cold.*

3.2.2 Interactions

In this subsection, we present the different techniques of interaction between feedback loops, how they communicate.

Stigmergy

Stigmergy is a method of indirect communication in a self-organized emergent environment, where agents communicate with each other by changing their environment. Stigmergy was first observed in nature: ants communicate by depositing pheromones behind them, so that other ants can follow the trail to the food (or the colony) as needed. This is called a *stigmergic system*.

In stigmergy, two loops monitor and affect a common subsystem. In our previously stated example, taken from Norbert Wiener [30] and resumed by Alexandre Bultot[7], one can observe a tribesman in a hotel lobby where the temperature is regulated by a thermostat. The tribesman is cold and as he is primitive, he starts a fire. His behaviour is represented by the outer loop. On the other hand, the room is air-conditioned (the inner loop). Thus, as the tribesman stokes the fire, the room temperature increases, causing the air-conditioning to work harder. The final result is that the more the tribesman stokes the fire, the lower the temperature will be.

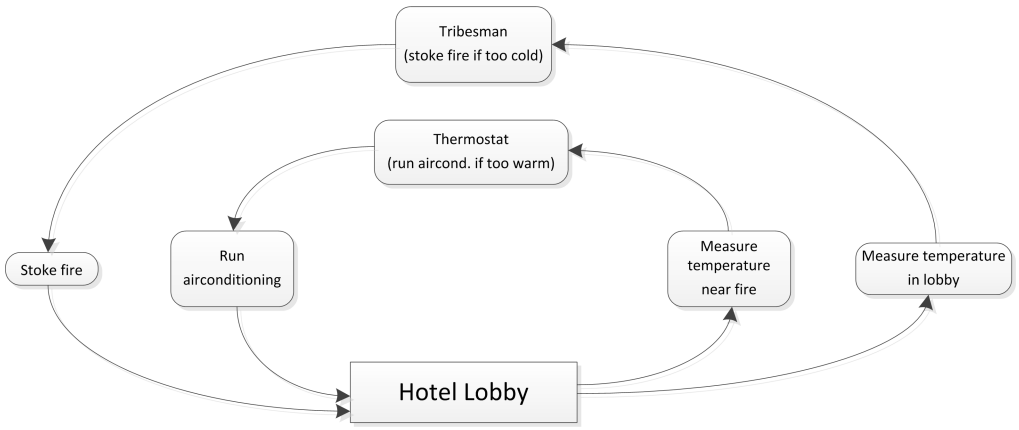


Figure 3.5: Stigmergy at hotel lobby

The two loop affect interdependent system parameters: the temperature in different parts of the room.

This example is an example of *uncontrolled stigmergy*: the two loops will compete and this may lead to a runaway situation (i.e. the hotel being set on fire)[7]. In general, stigmergy should then be used with care[36].

Management

In management, one loop directly controls another loop. Based on the previous example, the correct solution is that the tribesman can evolve and learn his lesson. Instead of starting a fire to keep warm, he now knows that he must adjust the thermostat to the desired temperature.[7]

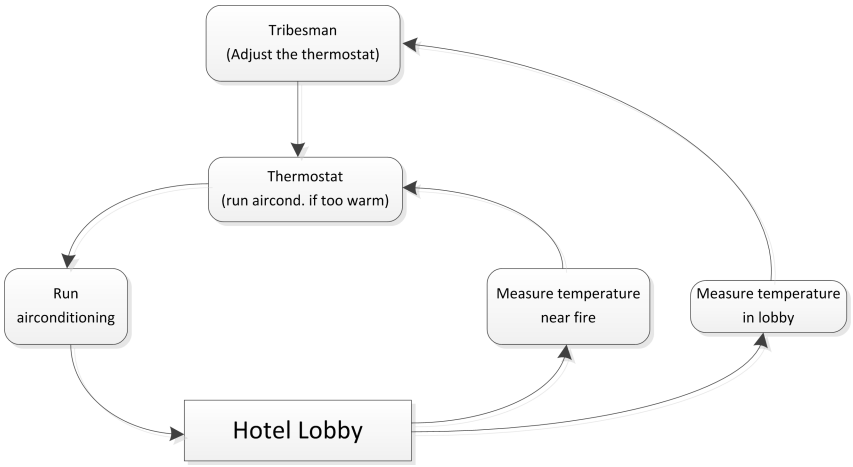


Figure 3.6: Management at hotel lobby

The outer loop is now managing the inner one and the system tends to a stable state. This example with the tribesman shows that sometimes management is preferable to stigmergy. The primitive tribesman should control the temperature by using the thermostat. This is a case where stigmergy is undesirable.

3.2.3 Feedback Structure

This extract is for understanding the relationship between the analyzed system and the feedback structures:

Each feedback structure consists of a set of feedback loops that together maintain one global system property. Depending on the system's operating conditions, different parts of the feedback structure will be active, which enables it to adapt to a wide range of operating conditions. [...] Each feedback structure consists of a set of feedback loops that together manage one system property. [...] It is important to distinguish between the system level (feedback structures) and the building block level (feedback loops). A feedback structure is built using feedback loops as building blocks and maintains a global system property by combining the goal-driven behaviors of its constituent feedback loops.[34]

The question is, *what is a system property*? This is what we will answer in the next section by establishing the relationship between a state diagram and the active loops.

3.3 Binding *Feedback Structures* and *State Diagram*

All components of a feedback structure can not be active at the same time. For example, let imagine that we're playing chess, what happens in our brain? We will use an article by Dr. Jordan Grafman dealing on the topic[29].

"*Imagine yourself as a chess player about to checkmate your opponent*", Grafman said in describing the work of the brain. "*All your knowledge and experience are being retrieved for your next move. First, you perceive the pieces on the board and mentally separate the color-coded pieces. Then you analyze their positions on the board, identify the value of the different pieces, and retrieve the rules of the game for any move. If you are a skilled player like the 10 subjects in this study, you also recognize specific patterns that signify when you have an advantage over your opponent. Finally you have to analyze the consequences of your potential moves and the countermoves of your opponent*"

For chess players, color separation and spatial discrimination activate parts on both sides of the brain toward the back of the head (1) known to be associated with visual processing. Rule retrieval activates two parts on the left side of the brain, a small structure deep within the brain associated with indexing memories (2) and a structure in an area near the left ear associated with memory storage (3). Checkmate judgment activates areas on both sides near the front of the brain crucial for planning (4) and in the back of the brain important for generating images (5).

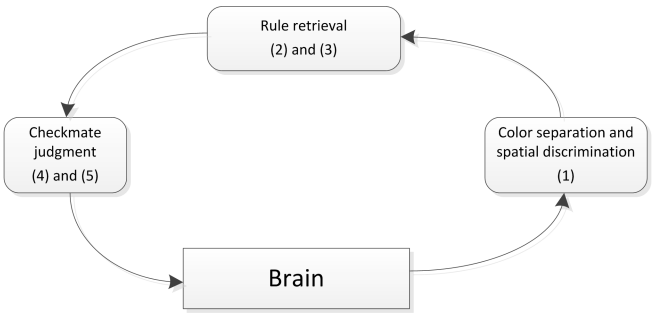


Figure 3.7: Brain activity in chess playing

Now imagine that in addition to playing chess, we experience the feeling of hunger. What happens in our brain?

Actually it is within the hypothalamus that is performed the integration of different types of signals for the regulation of appetite. In scientific terms, the signals - the inputs - are both classical neurotransmitters and neuropeptides. The effect on appetite - the output - then passes through different signaling pathways. This response of the hypothalamus acts on many brain structures involved in memory, motivation, planning, decision making, learning, action and motor control, which are all functions required for food.

According to the *lipostatic theory* of control of food intake described by Ganong[52], adipose tissue produces a humoral signal which is proportional to the amount of fat in the body and which acts on the hypothalamus to decrease food intake and increase energy expenditure.

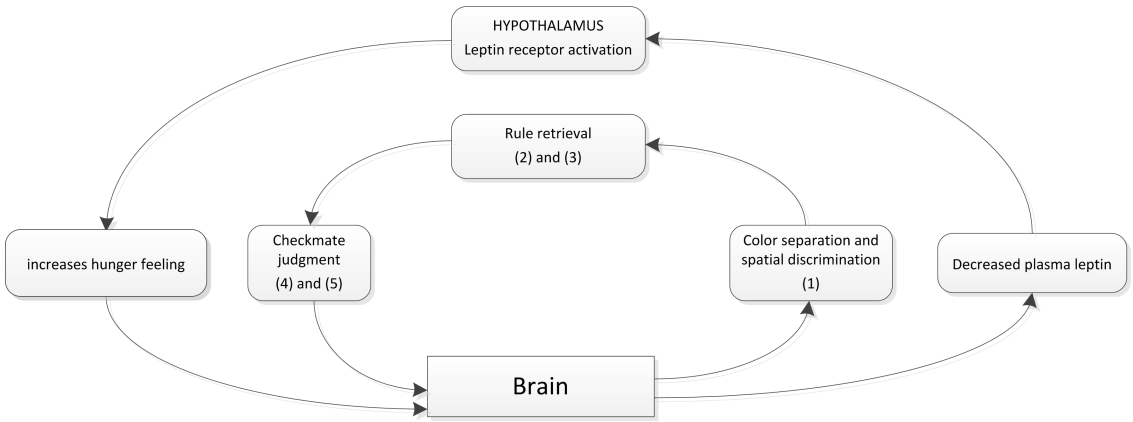


Figure 3.8: Brain activity in chess playing when hungry

Now, what happens when we want to annihilate that? The answer is simple: we eat.

But what happens in the background? Exactly the opposite effect of previously: the absorption of food will increase the concentration of leptin. Thus, the receptors in the hypothalamus are more solicited, which implies that the hypothalamus give us a feeling of satiety.

This can be seen in the diagram below:

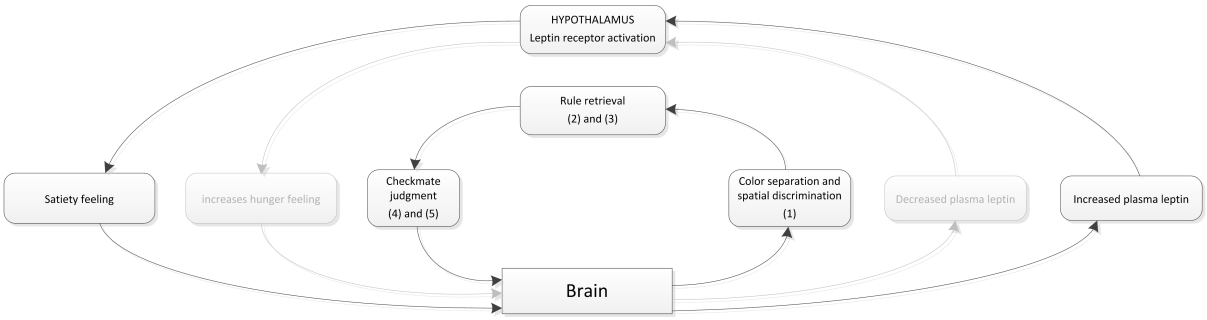


Figure 3.9: Brain activity after eating

As the affected subsystem (brain) is the same as that used in a game of chess, thinking about the game of chess will be influenced by the feeling of hunger. It is therefore a **stigmergy** interraction.

Assuming we play chess by default(we freeze the feedback loop *playing chess* to *active*), there are two possible states: $S1$, *chess playing - hungry* and $S2$, *chess playing - not hungry.*

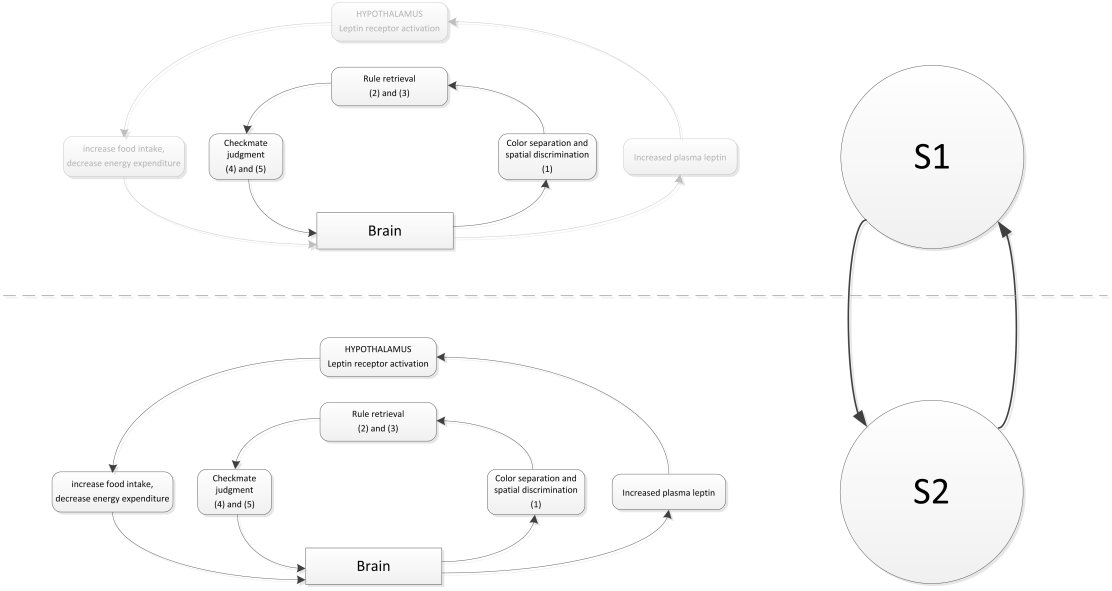


Figure 3.10: Each combination of active loops has a corresponding state

So as stated above, each possible combination of active loops has a corresponding state, and inversely, each possible state of a complex system has a combination of active feedback loop.

Comment

We can not deduce a state diagram based solely on a feedback structure. We need the system specification and execution rules to know which combinations of feedback loops are allowed in the feedback structure.
In the other way, we can not deduce a feedback structure only based on a state diagram: we know nothing about the different components by watching a state diagram.
So we can go from one to the other, but we need additional information to perform this.

3.4 Feedback Structure Formalism

This section will allow us to formalize the concepts and intuitions that we had in the previous points. As a first step, we will discuss the complex components, which is a particular type of component. After that we will undertake to define a formalism for feedback structure. And finally we will give a comprehensive analysis of some patterns of feedback structure.

3.4.1 Complex Component

We define component as complex if it can do nontrivial reasoning.

If we start from the example of the respiratory system or the chess player, we see that there are components with complex behavior. They are respectively the *conscious control of breathing and body* and the *hypothalamus*. A complex component implements an algorithm that completely solves a given problem within a particular (small) part of the operating space. From the viewpoint of the system containing the complex component, the latter is a black box: we do not look inside the box, we use it as a primitive. The behavior of a complex component cannot be predicted by the rest of the system, since otherwise they would not be needed. Most of the time, they play a key role in the system wherein they are included. But this reasoning is only valid in a particular part of the system's state space and should be ignored in other parts.

It should be noted that some feedback structures do not contain complex component because they simply do not have in need. This is the case for the hypothalamus system (see section 6.2).

If we go back to the respiratory system, we see that the behavior of complex component is to activate/deactivate the feedback loop *respiratory system*. This loop handles the details of the control of the respiratory muscles and clocking human respiratory cycles. *The complex component does not have to understand these details, but interacts through several parameters in the respiratory loop: the timing of the cycle and the depth of the breathing.*[31]

Impact of complex component: drowning

Look at the role of complex component in the respiratory system. Assume that we fall into the water. What's going on?
Drowning does not necessarily result in the penetration of large amount of water in the lungs. Water penetration, even in minute quantities in the airways, causes reflex apnea: the epiglottis is closed by laryngeal spasm to protect the airway, preventing breathing even when the head is above water. Therefore, the available oxygen in the body decreases: this is called hypoxia. If prolonged cerebral hypoxia, the conscious control is no longer activated, which causes that spasm rises, which activates the breathing reflex feedback loop, allowing the entry of water in the airways.

The presence or not of a complex component can have a significant impact on the design of feedback structures. For example, a complex component like conscious control may introduce instability that needs fail-safe protective mechanisms. To design system that contains complex components, we need to follow special rules that are included in the next chapter.

3.4.2 Formalism

We present a formal approach to define the composition of a feedback structure.

Global property of a feedback structure

Each feedback structure maintains one global property of the system. That is, it has a corresponding predicate (a logical sentence) that is function of the system's parameters and of time. When this predicate is true, the feedback structure is said to be operative (working). When the predicate is false, the feedback structure has failed. The feedback structure is supposed to adapt itself to environmental conditions so that the predicate is true as often as possible.

Definition of a system

A System consists of a set of feedback structures together with their dependency graph.

$$Sys = (WS, DG)$$

where WS is a set of feedback structures and DG is their dependency graph. The specification of the system is the conjunction

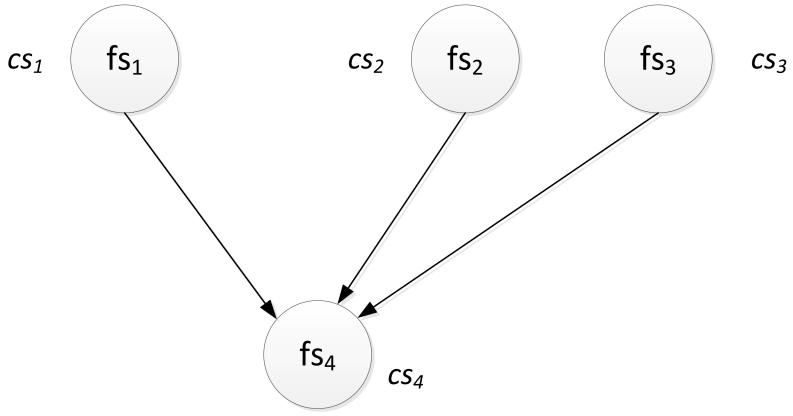
$$S_1 \wedge S_2 \wedge ... \wedge S_n$$

where each S_i is the **global property** corresponding to a feedback structure fs_i in WS.

$$WS = \{fs_1, fs_2, ..., fs_n\}$$

What is the effect of the dependency graph? There are different possibilities, but it seems that the basic effect must be one of operational dependency: if $fs_1 \rightarrow fs_2$ in the graph then fs_1 must be operative (S_1 is true) for fs_2 to be working. I.e., if fs_1 is in an inoperative state then fs_2 eventually becomes inoperative (after a finite number of transitions - this could be expressed as a formula in temporal logic). Note that this is an interesting kind of implication that we can formalize with temporal logic. If fs_1 is inoperative intermittently this may or may not cause fs_2 to become inoperative, depending on how fs_2 is built.

In addition of this condition of operativeness, there is another one: the state of fs_1 determines which are the possible states that fs_2 can be in. It may be that fs_1 has reduced functionality (even though it implements S_1) that can effect the operation of fs_2. For example, in the human respiratory system, the conscious control can fall away because O_2 level is too low, but the respiratory system is still functioning!



Let's feedback structures fs_1, fs_2, fs_3, fs_4 with current states cs_1, cs_2, cs_3, cs_4, respectively. We can define a function f to restrict the possible states.

$$f_i(cs_1, cs_2, cs_3) \subseteq (possible states of fs_4)$$

so $cs_4 \in f(cs_1, cs_2, cs_3)$.

Definition of a Phase

Phases occur in systems that have many instances of a feedback structure with the same architecture, such as a peer-to-peer system or a parallel system. These instances will generally interact with each other according to a dependency graph. In such a system, a phase p is a subset of instances with the same state. All phases p_i form a partition of the system.

With this definition of phase we can study the global behavior of the system. Indeed, we can rank the phases in terms of the fraction of the system that each of them describes:

$$|p_i| \geq |p_j| \geq |p_k| \geq ...$$

where $|...|$ means cardinality of a set. This ranking order is a function of time.

We can then define **macroscopic** phase transitions, for example, one possibility is when the global ranking order of the phases changes.

The idea is that a system controlled by feedback loops may have several macroscopic (global) states, similar to phases. If the system is exposed to a hostile environment, it may change its global state. In order for the system to survive such changes, they should be reversible. [36]

Also, we can see that changes in the phase partition do not require any global synchronization: the state of each feedback structure is a purely local property that changes according to the local environment of that feedback structure.

The next step after the formalization is to apply it to examples, to see whether it helps in understanding how a system works.

Definition of a feedback structure

A feedback structure is a structure

$$fs = (G, D, A)$$

where:

- G is a directed graph (V, E) of components V and communication links E.
- D is a finite state machine $(S, init, death)$ with states S, initial state $init$ and death state $death$. Each state s in S is a *mode of operation* of the feedback structure, where the mode determines the set of components that are active.
- A is a function from $S \rightarrow P(V)$, that is, for each state s in S, $A(s)$ is the set of components that are active.

Coupled to the FS, there is his behavior:

$$fs_{behavior} = (E, T)$$

- E is an infinite sequence that represents the execution of a feedback structure.

$$E = [\underbrace{s_1}_{internal}, \overbrace{e_1}^{external}, s_2, e_2, ...]$$

  where s_i is a state (internal condition) and e_i is an event $\in C$(external condition). We note that s_i is a time interval and e_i is a time instant, we have:

$$\begin{aligned} time(s_i) &= [t_i; t_{i+1}] \\ time(e_i) &= t_i \end{aligned}$$

- T is a time duration:

$$T : (S \cup C) \times \mathbb{N} \rightarrow \mathbb{R}^+$$

  where C is the set of external conditions.

Each state transition in E has a condition that is a function of the external environment, under which it can be taken. For example, in the human respiratory system, the transition

(conscious, holding breath) $\Rightarrow$ *(unconscious, holding breath)*

can be taken if the oxygen level falls below a particular number, which is a function of how the biological system works. These conditions connect the formal model to the real biological system. It's important to make this connection, and it lets us use the formal model to reason about the real system, as described in the next paragraph.

One important purpose of the formal model of the feedback structure is to verify the behavior of the feedback structure. We should be able to derive a set of conditions under which the feedback structure will behave correctly (that is, where current state $s \in S$ is always $\notin$ *death*). These conditions need to be proved of the real system. If the condition becomes false, then the system will break - so we have found a limit of the system. We can therefore use the formal model to debug the real system. For example, if we remove the topmost component *unconscious failsafe* from the human respiratory system, then the resulting system has a bug: holding your breath can make you die. The formal model should be able to derive that the transition

(conscious,holding breath) $\Rightarrow$ *(*,breathing)*

is impossible (condition to make it possible is never true) because of the lack of the failsafe component.

Another purpose of the formal model is to analyze what happens when several feedback structures are part of the same system (*Weakly Interacting Feedback Structures*). Large systems very often contain several feedback structures. These feedback structures will interact through some shared environment variables. The formal model should be able to tell us when this interaction is ok and when something goes wrong.

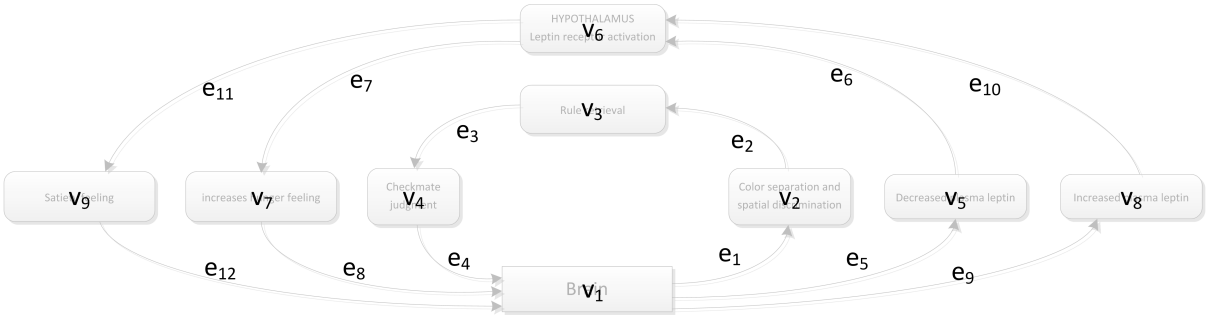
3.4.3 Formalism Application

We believe that the formal model as described in this section fulfills the desired expectations. We will illustrate this formalization on the chess player system (CP).

$$(FS_{CP}) = (G, D, A)$$

where

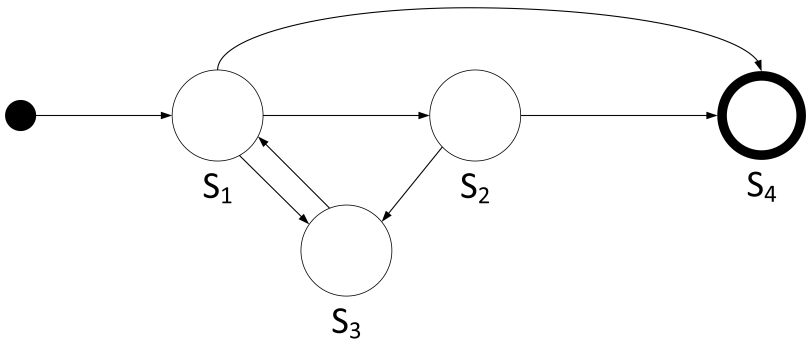
G is the following graph:



$\{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9\} \in V$

$\{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}\} \in E$

$D = (S, init, death)$ is the following state machine :

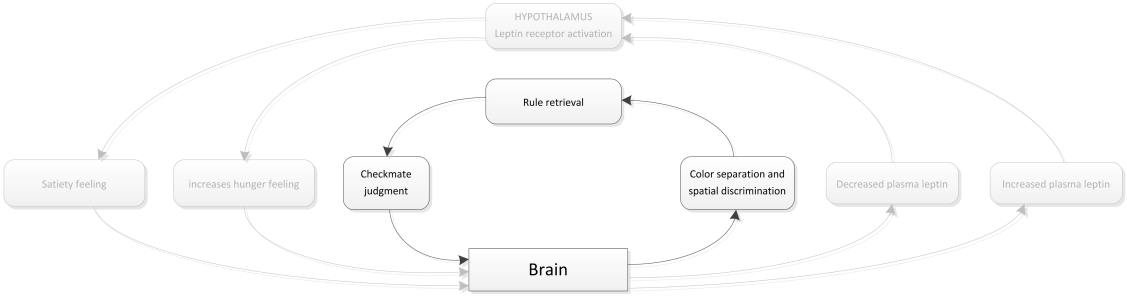


where

- $\{s_1, s_2, s_3, s_4\} \in S$
- $init = s_1$
- $death = s_4$

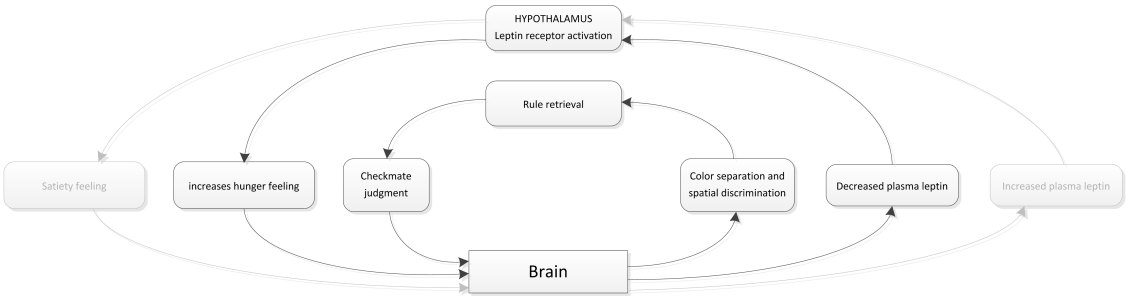
A is the following function:

$A(s_1) = \{v_1, v_2, v_3, v_4\}$

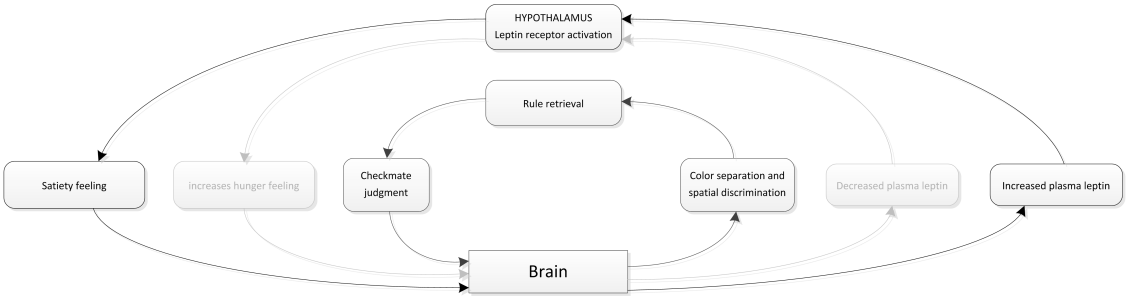


Assuming you're not hungry when you start playing.

$A(s_2) = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$

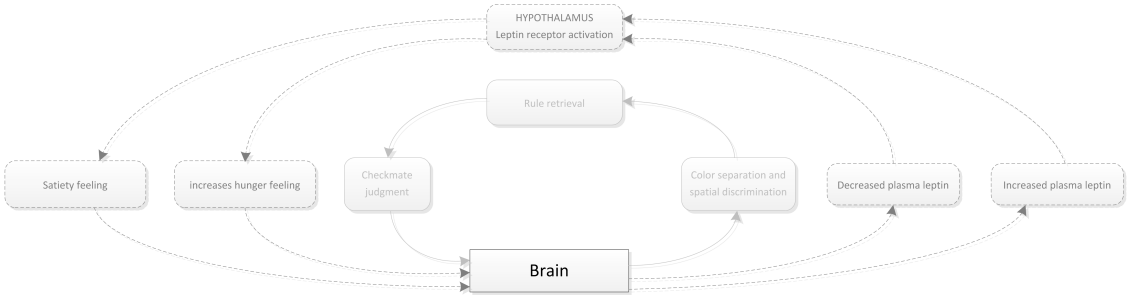


$A(s_3) = \{v_1, v_2, v_3, v_4, v_6, v_8, v_9\}$



Assuming that you can eat while playing

$A(s_4) = \{\}$



This is a death state. We stop playing chess, regardless the other loops.

E is represented as a transition table which takes into account external events.

Current state (internal)	Event (external)	New state
S_1	e_1: decides to eat	S_3
S_2	e_1: decides to eat	S_3
S_1	e_2: decides to stop playing	S_4
S_2	e_2: decides to stop playing	S_4
S_1	e_3: game ends	S_4
S_2	e_3: game ends	S_4

We can see that different events can make the same transition (ex : $S_1 \Rightarrow S_4$)

3.4.4 Patterns

We can identify some basic patterns. We will take four from [7].

Fail-safe A failsafe system is constantly checks to ensure perfect functioning and bringing the process to a safe state in case of error. This pattern consists of two feedback loops: one negative and one complex (neither negative nor positive). The mode of interaction between these two loops is *management*: the negative loop manages the complex one.

Example of fail-safe: CRC

A Cyclic Redundancy Check (CRC) is a tool for detecting transmission errors. The mechanism is to protect data blocks by adding a control code. This *CRC* code contains redundant elements compared to the transmitted data to allow the detection of errors, but also repair them in some cases. It is used in the case of transmission of a large series of bytes.
This code is based on the fact that any bit string can be used to construct a polynomial, each of the bits gives the value of a corresponding polynomial coefficient. Example:

$$0101 \rightarrow 0x^3 + 1x^2 + 0x^1 + 1x^0 \rightarrow x^2 + 1$$

The implementation of the CRC requires choosing a reference polynomial called the *generator* often named $G(x)$. The sender will then divide the message by $G(x)$. But we can not directly perform this division. We will therefore make a division, byte by byte, input frame. The receiver of the frame divides the sum of the bytes of the frame and the value of the CRC by the same polynomial. If the remainder is NULL, it means the frame is correctly received. Otherwise, the receiver will request a retransmission.

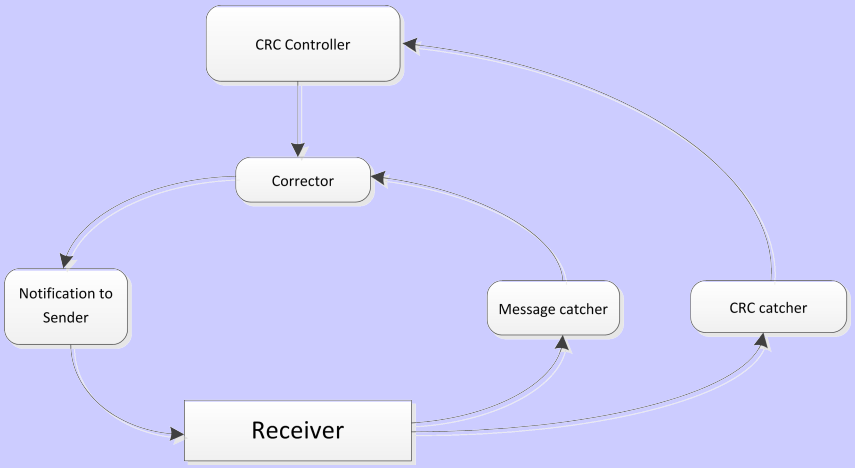


Figure 3.11: Notification to the sender depends on the result obtained by the controller

Data abstraction In data abstraction, two feedback loops interact through management. In this case, a complex loop manages another loop that can be either positive or negative. The complex loop does not need to know how the other loop works, it just manages it.[7]

Example of data abstraction: Garbage collector

As previously said, a running application can interact with the garbage collector. This running application can decide to invoke the garbage collector directly (it bypasses the detection of threshold). This is done commonly just before a critical piece of code, to make sure the garbage collector is not run unexpectedly during the critical code. This is an example of management.

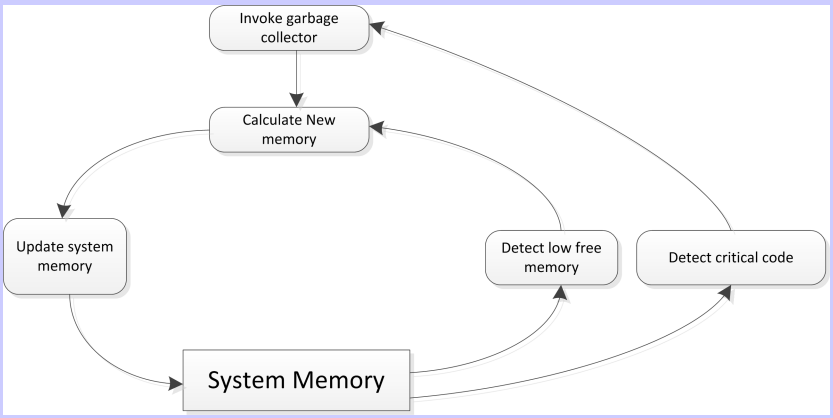


Figure 3.12: The inner loop is the garbage collector, the outer one is the running application.

Management tower This pattern is a combination of patterns *data abstraction* and *fail-safe*. In fact, it is a tower management which is formed by a combination of feedback loops. We can observe one in the three outer loops from the example of human respiratory system.

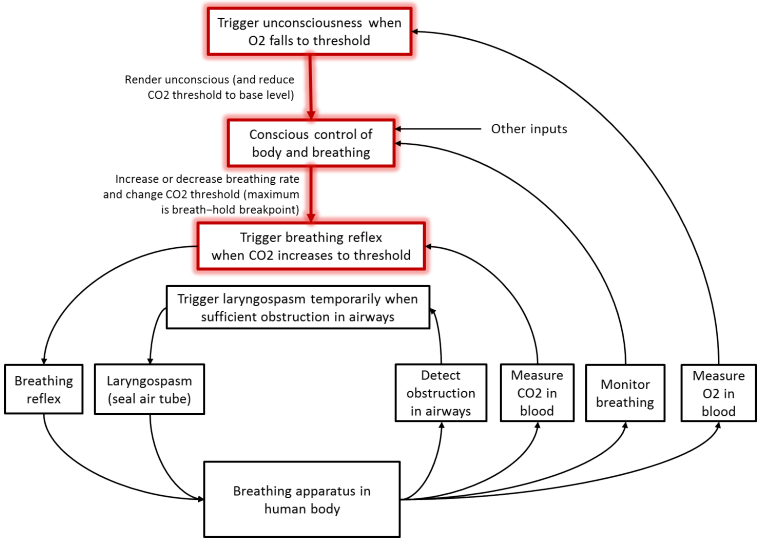


Figure 3.13: *Conscious control* is secured by the fail-safe system and implements the breathing system by an abstraction pattern.[7]

Event inhibitor A slow negative loop controls a fast positive loop by *management*. In general terms, an inhibitor is something that slows or prevents a process.

Example of event inhibitor: Nuclear control rod

In the field of nuclear, a chain reaction occurs when a neutron causes fission of a fissile *Uranium-235* atom producing more neutrons which in their turn cause further fissions. The chain reaction can be monitored and used in a nuclear reactor to produce energy. How? By using *control rods*. This is a movable neutron-absorbing material, for decreasing the neutron multiplication factor by neutron sterile-capture.

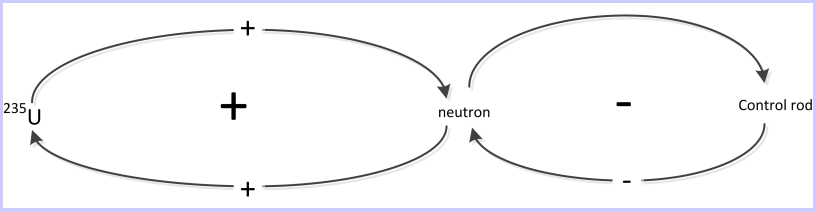


Figure 3.14: Control of the nuclear fission

In feedback structure that looks like this[a]:

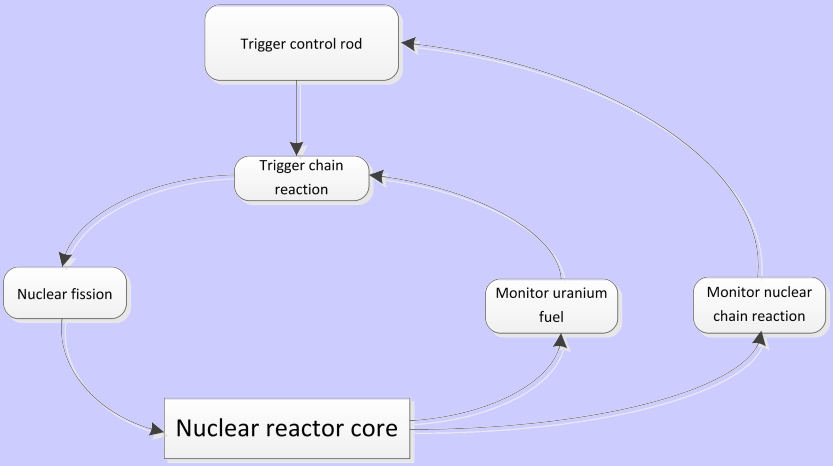


Figure 3.15: Feedback structure of a portion of the core of a nuclear reactor

[a]Note that for clarity we do not consider the other elements of the reactor.

Chapter 4

Elaboration of Design Rules

Contents

The topic of this chapter is to devise a methodology at a similar level of abstraction than an existing methodology of software construction. The goal is to elaborate some design rules that, starting from system description, build the corresponding system in a qualitative way, using weakly interacting feedback structures.

There are three sections: Firstly we propose a qualitative methodology. After that we will link this methodology with quantitative techniques such as system dynamics, model checking and control theory. It is important to discuss with these concrete technical notions. Finally we'll launch tracks to reduce the scope of this methodology from *system design* to *software design*.

4.1 Proposed Methodology

As mentioned previously, interactions between the feedback structures are weak, for this reason the design process divides the system naturally into single feedback structure and combining them to form the complete system.

This process consists of three main steps that are taken from [34]:

1. We first decompose the system specification into separate properties, each of which is assigned to one feedback structure.
2. We then design each feedback structure separately.
3. Finally, we combine these feedback structures to form the complete system.

Sometimes, the complete design may need several iterations due to potential undesirable dependencies.

Remember that our formal approach of a system is decomposed into a set of feedback structures (WS) and a dependency graph (DG).

$$Sys = (WS, DG)$$

whereas our formal approach of a feedback structure (FS contained in WS) is

$$FS = (G, D, A)$$

with G, D and A as components.

We will initially graphically illustrate our methodology:

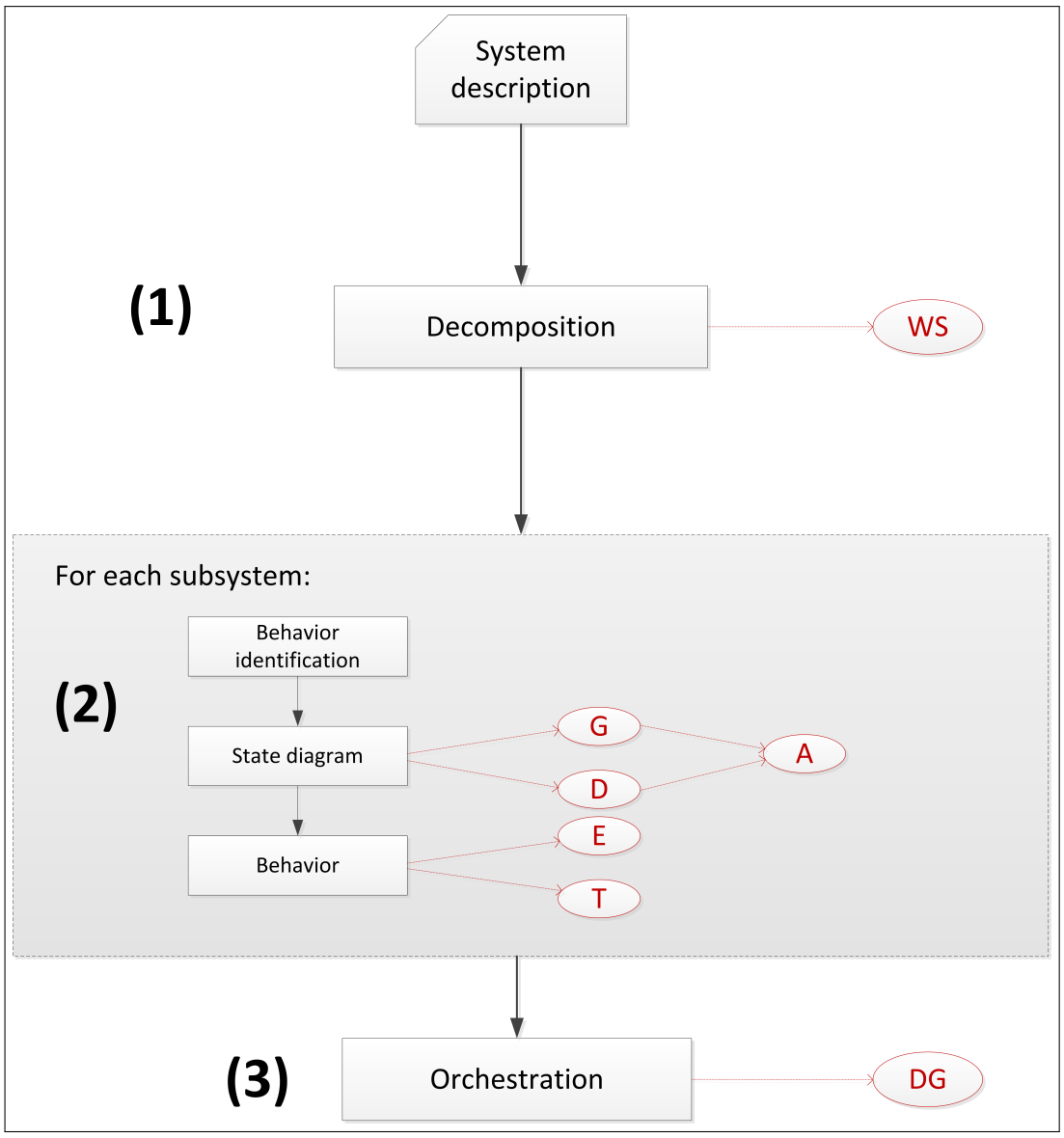


Figure 4.1: Schematic view of our design methodology

The goal is to allow to design systems with desired global properties. The three main parts outlined above will be discussed in detail in the following subsections.

4.1.1 Decomposition

Actually, the specification of a system is nothing but a combination of properties. For example, let's take Scalaris from Appendix A. Scalaris is a scalable, transactional, distributed key-value store. It can be used for building scalable Web 2.0 services[1]. In their article [34], Peter Van Roy *em et al* define Scalaris specifications as a conjunction

[1] more info in Appendix A and http://code.google.com/p/scalaris/

of six properties:

$$S_{Scalaris} = S_{key-value} \wedge S_{connectivity} \wedge S_{routing} \wedge S_{load} \wedge S_{replica} \wedge S_{transaction}$$

The approach is to choose right properties that facilitates reasoning about the program and its specification.

Once we have these properties, each of them will be performed by a single manager, where a manager corresponds to a feedback structure. Actually dividing system functionality into feedback structures is *a natural way to define and to separate concerns in real systems* [34].

We should not care for relevant links of dependencies and interraction among the feedback structures, we will deal with this problem later.

Extract: global properties in physics

In this passage from an article called *Self Management and the Future of Software Design* by Peter Van Roy [37], the author established a link between the global properties of systems and current developments in physics. This text is out of scope of this master thesis but we wanted to publish it informatively.

> *An important part of any general system theory concerns the global properties of a system. Can they be determined for an existing system and can we design systems with desired global properties? The latter question is especially important for large-scale computer systems, such as the Internet or distributed systems built on top of the Internet. Some of the important points are the system's stability, its behavior when stressed, and whether the system's imminent collapse can be detected before it happens. Answers to some of these questions exist for complex systems in physics. Such systems consist of large numbers of very simple components, but they can sometimes be a useful approximation to computer systems. For example, Krishnamurthy et al [22] have done an analytic study of the Chord structured overlay network using a master equation approach. Another example is the belief propagation algorithm. This algorithm is defined in terms of message passing between large numbers of simple nodes[56]. It has been used to give solutions to the SAT problem and other problems. Belief propagation is a general technique that can determine global properties of a system in terms of local properties. It can be used for monitoring global properties as part of a feedback loop.*

4.1.2 Building each feedback structure

Here we will build each structure independently feedback. By simple manipulations, we will find successively all the feedback structure 's components.

Behaviors Identification

This informal stage is quite simple and consists in identifying all possible behaviors of the feedback structure studied. We will use the example of the respiratory system:

Example: respiratory system

As a first step, we define the elements on which we will identify the different states. There are two: the state of the **larynx** and the state of our **consciousness**.

- For the larynx, we identify two behaviors: *breathing* and *laryngospasm*.
  - The breathing reflex is the default behavior. The breathing reflex is a semi-automatic mechanism that controls breathing and lack of conscious control of it, causing the sensation of suffocation in the presence of an excessive amount of carbon dioxide dissolved in the blood plasma.
  - Laryngospasm is an involuntary contraction of the muscles surrounding the larynx. The laryngospasm occur suddenly during brief attacks usually lasting less than a minute. When a laryngospasm occurs, vocal cords suddenly crashing making it impossible to breath. This can be caused for example when we fall into the water.
- For the consciousness, we identify three states: *normal*, *conscious* and *unconscious*.
  - Normal, as its name indicates is the default behavior. We breathe, we hold our breath without realizing it.
  - Conscious means that we *realize* that we breathe. For example the doctor asks us to breathe slowly for the auscultation, or when we make fun competitions of apnea.
  - Unconscious means that we have lost consciousness. For example, when we are for too long under the water, we lost our consciousness and become uncscious.

Once we have that, we need to combine these different states to obtain all possible states of the feedback structure:

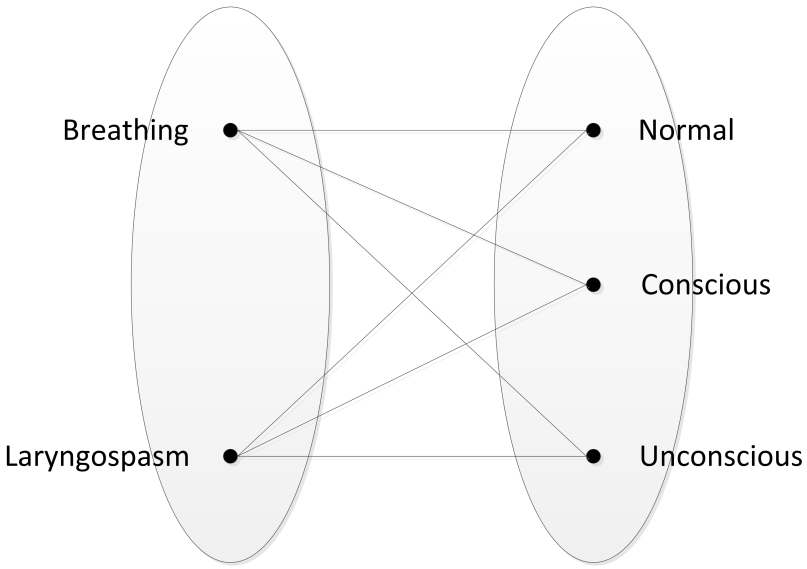


Figure 4.2: Combination of states

With this picture, we see that the feedback structure behavior will have 6 possible states.

Building the state diagram

We already have the states {*Conscious breathing*, *Conscious laryngospasm*, *Unconscious breathing*, *Unconscious laryngospasm*, *Normal breathing*, *Normal laryngospasm*}. The idea is to find the links and conditions between these states based on the system description.

We obtain this diagram, taken from [38]:

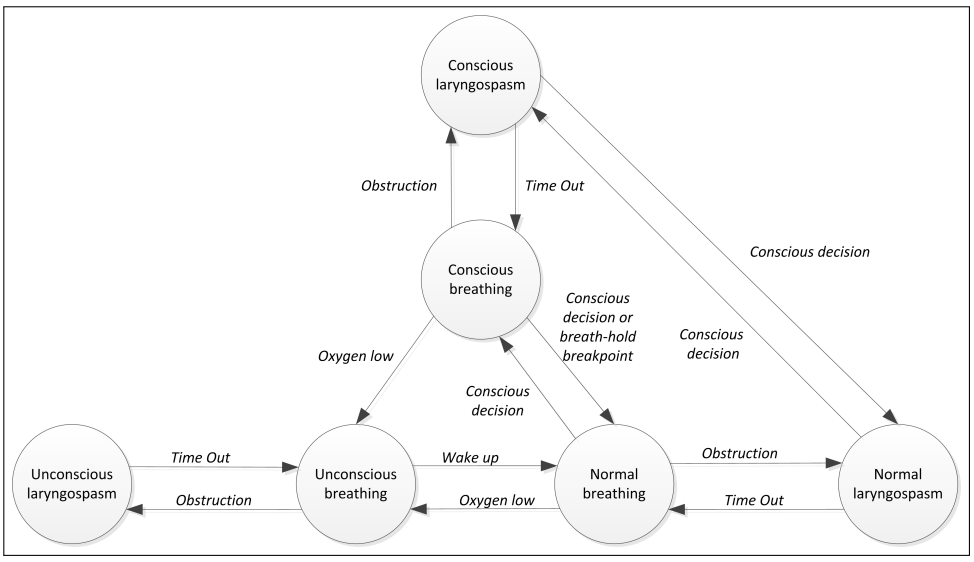


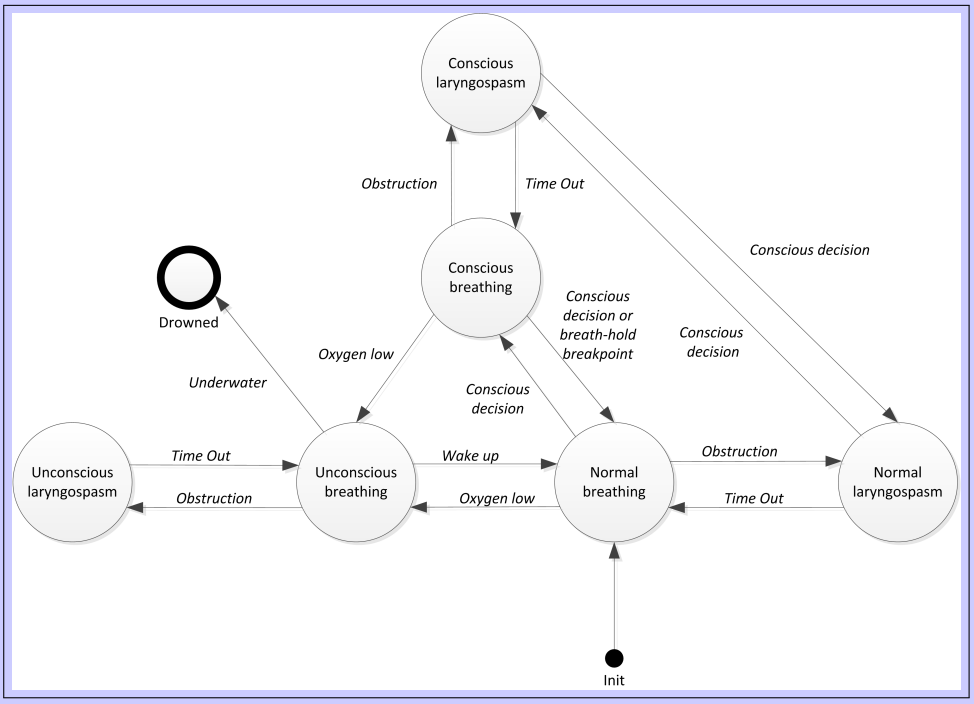
Figure 4.3: The human respiratory system can be seen as a state diagram

Don't forget to add *init* and *death* states. We can deduce these states from the system description.

death and _init_ states in respiratory system

- For the *init* state, we start from hypothesis that we are initially in the most common situation: *normal breathing*.
- For the *death* state (it may there be many, but in our case there is only one), we took the scenario where the person dies. Actually the person dies of drowning. What's going on? The person is unconscious and underwater. As there is an obstruction (water), we are in the state *Unconscious laryngospasm*. Then the person is for too long in this state (time out) and passes into the state *Uncoscious breathing*. The person breathes underwater, water enters the lungs ⇒ dead.

The state diagram $D = (S, init, death)$ becomes:



with $init = \{$ *Normal breathing* $\}$ and $death = \{$ *Drowned* $\}$.

We have therefore the state diagram D, the first component of our feedback structure. The next step is to find G.

Building Graph

The idea of development is to build each feedback loop separately and after assembling them to obtain the final structure of G.

Let us recall briefly the description of a feedback loop. There are three agents attached to a subsystem: monitoring agent, correcting agent and actuating agent. The actuating agent is the only one that does something *concrete*, that performs an action. In respiratory system there are two actuating agents: breathing reflex and laryngospasm. In Garbage Collector system there is one actuating agent: update system memory. It means that, in the final feedback structure, there will be two and a one left curve for respectively respiratory and garbage collector system.

After that we need to associate for each actuating agent the corresponding monitoring and corracting agents. For this we must rely on our system description. In respiratory system we know that the breathing reflex occurs when a certain level of CO_2 in the blood is reached. It can be triggered by other causes such as our conscience but they are exceptions. We will return later but for now we focus on the main causes, in the *normal* state. Therefore two elements are needed: one that will monitor our CO_2 and one that will serve as triggering. We can easily make the matching with the two agents. We thus obtain our first feedback loop:

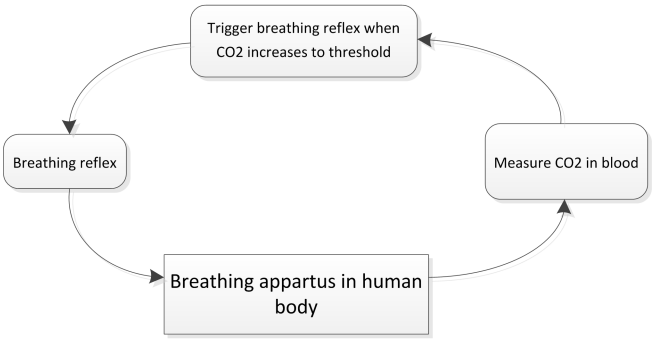


Figure 4.4: Our first feedback loop is built

Similarly (based on system description), we obtain our second feedback loop for laryngospasm:

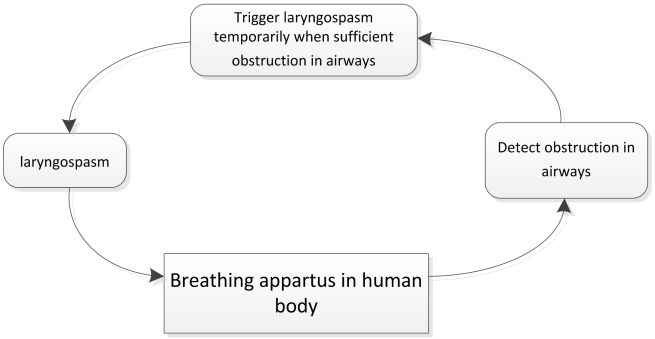
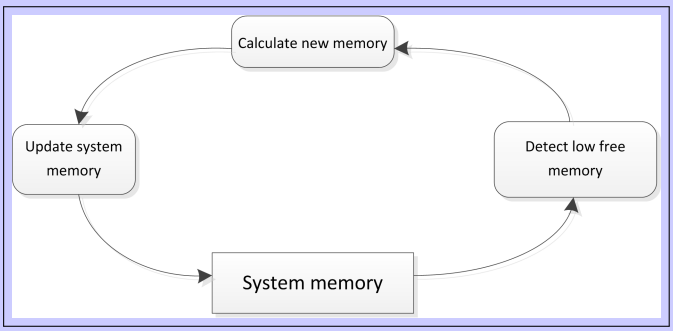


Figure 4.5: Our second feedback loop is built

in garbage colletor system

In the description of the system garbage collector, we identified the only action that is *update system memory*. This may be due to two things: A threshold of low system memory is reached or critical code is detected. As said earlier, we are now interested in the default cause. In this case this is low system memory. In fact *critical code* can be considered as an exception. This is a routine that interrupts the normal process of garbage collector. We therefore obtain this feedback loop:



We must now establish an order between the feedback loops. In our example what is the inner loop and the outer loop? Simply refer to the system description. What happens if the CO_2 has reached its threshold and in the same time there is an obstruction in the airways? We see that after a while, the breathing reflex takes over the laryngospasm. This means that the breathing reflex is the outer loop and laryngospasm the inner one.

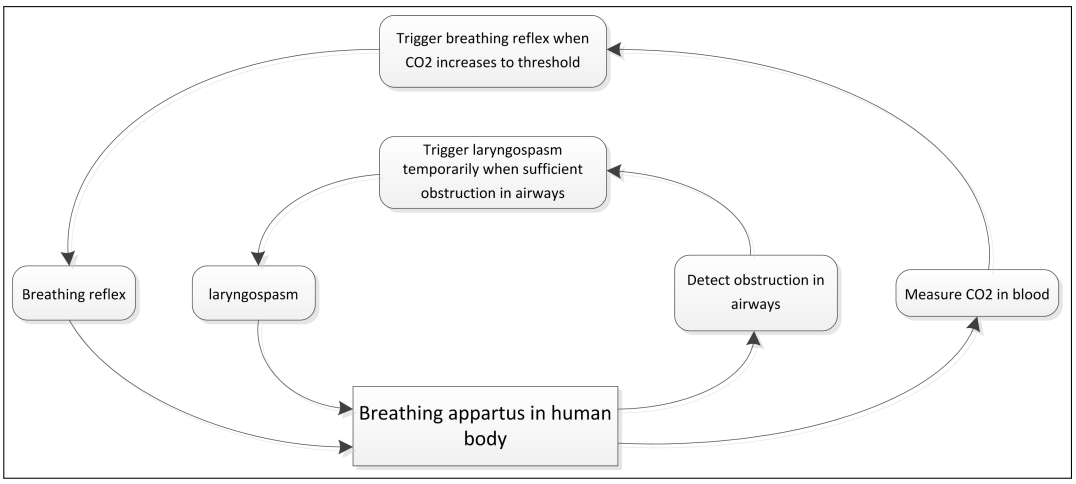


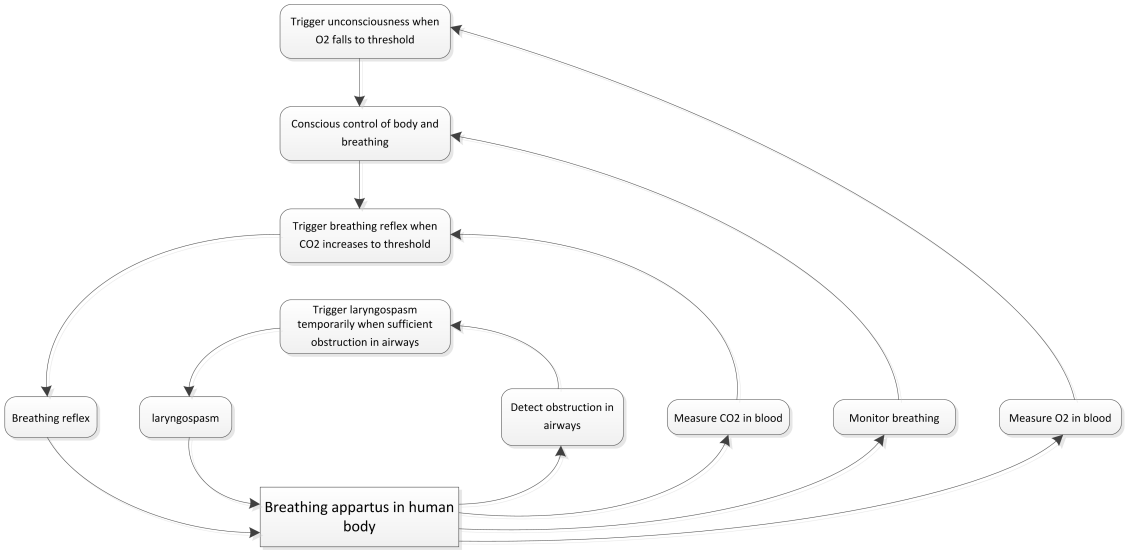
Figure 4.6: The feedback structure emerges progressively

The breathing reflex loop implements normal breathing independently of the laryngospasm loop [34]. Each feedback loop can then be designed and optimized separately using control theory [19] or discrete systems theory [8].

We must now integrate the remaining states: *conscious* and *unconscious*. We are clearly here in a hierarchical relationship:

$$unconscious \succeq conscious \succeq normal$$

So based on system specification we just use the hierarchical pattern called *management tower* seen previously in the section 3.4.4. We obtain this graph:



If we are attentive, we note that there is no triggering event type to activate the loop consciousness. In fact this is due to the presence of a complex component

(see previously). This component is the link that communicates with other feedback structure of our whole system. For example there could be another feedback structure called ”brain” that communicate via this component.

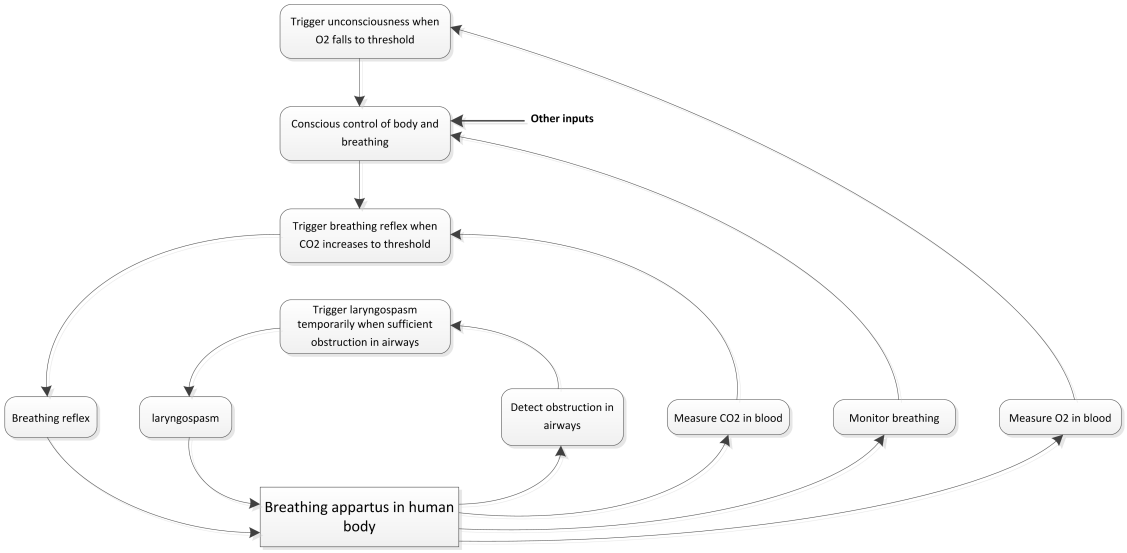
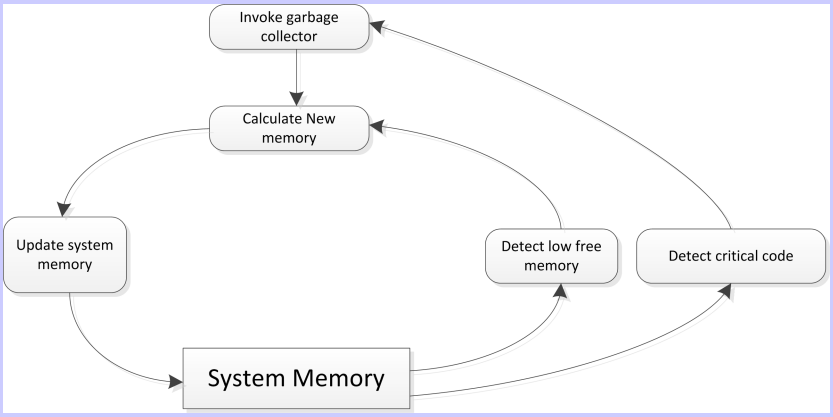


Figure 4.7: Adding complex component

garbage collector

In the case of the garbage collector, the pattern to be applied is *data abstraction*.



Note: Complex components should be sandboxed

This extract is taken from [36], it highlights the key role of complex components.

> *In the human respiratory system, there is a conscious control of the breathing apparatus. This has the advantage that all the power of conscious reasoning can be brought to bear in the case of catastrophes. For example, if the person is in a car that falls into a river, the conscious control can stop breathing temporarily until the person is outside of the car. Conscious control is also dangerous, however: it can introduce instability. If the person decides to stop breathing (for example, because of a wager), then the system must somehow defend itself. This can be done by having an outer loop observe the conscious control. In the case of the human respiratory system, the brain falls unconscious if the blood oxygen level drops too low. When this happens, the conscious control disappears and the breathing apparatus starts working normally again. The general rule is that complex components can improve the power of the system (for example, they can stabilize an unstable system, like a pilot who stabilizes an unstable airplane) at the price of possibly introducing instability at other occasions. They must therefore always be observed by an outer loop that can take action when this happens.*
>
> *With respect to stability, there is no essential difference between human components and programmed complex components; both can introduce stability and instability. Human components excel in adaptability (dynamic creation of new feedback loops) and pattern matching (recognizing new situations as variations of old ones). They are poor whenever a large amount of precise calculation is needed. Programmed components can easily go beyond human intelligence in such areas. Whether or not a component can pass a Turing test is irrelevant for the purposes of self management.*

At this stage, we built the graph G and the state diagram D. Let us continue our process!

Finding A

We have seen previously that a very important part of a feedback structure is its state diagram. To find A, the idea is that each state in this diagram corresponds to a set of active feedback loops in the feedback structure. For a given state, we call this the *dominant set*. By a qualitative analysis, we divide the whole operating space into

regions, where each region corresponds to one dominant set. At any given instant, the system's behavior corresponds to a point in this operating space. When the system's behavior changes, this point moves. When a feedback loop becomes active or inactive, then the dominant set changes, and the point will make a transition from one region to another.

Therefore the goal of this step is to bind G and D. There are two possible ways:

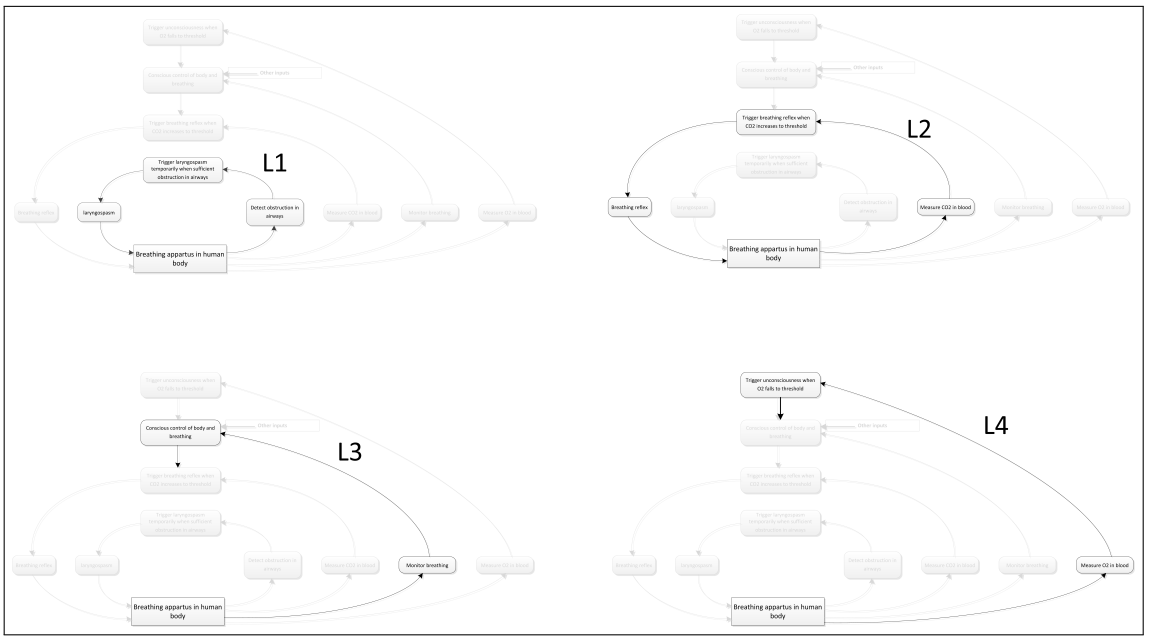
$$G \Rightarrow D \tag{4.1}$$

or

$$D \Rightarrow G \tag{4.2}$$

By experience, we'll choose the second one (4.2). It means that we start from all the possible dominant sets of active loops and we'll try to match each of these combinations to a state. This is a robust aproach. Indeed we cover not only the cases from the states but also all possible cases, include cases that *normally* not occur.

The easiest way to proceed is drawing an array with all the possibilities. Example for the respiratory system:



Then we build the combination array, based on system description:

possible states: {*Conscious breathing*, *Conscious laryngospasm*, *Unconscious breathing*, *Unconscious laryngospasm*, *Normal breathing*, *Normal laryngospasm*, *Drowned* }

L1	**L2**	**L3**	**L4**	corresponding state
0	0	0	0	Death (drowned)
1	0	0	0	Normal laryngospasm
0	1	0	0	Normal breathing
1	1	0	0	*none* (mutual exclusion)
0	0	1	0	Death (drowned)
1	0	1	0	Conscious laryngospasm
0	1	1	0	Conscious breathing
1	1	1	0	*none* (mutual exclusion)
0	0	0	1	Death (drowned)
1	0	0	1	Unconscious laryngospasm
0	1	0	1	Unconscious breathing
1	1	0	1	*none* (mutual exclusion)
0	0	1	1	*none* (mutual exclusion)
1	0	1	1	*none* (mutual exclusion)
0	1	1	1	*none* (mutual exclusion)
1	1	1	1	*none* (mutual exclusion)

Based on this table, we can deduce $A(s_i)$:

$A(normalBreathing) = \{L2\}$
$A(normalLaryngospasm) = \{L1\}$
$A(consciousBreathing) = \{L2, L3\}$
$A(consciousLaryngospasm) = \{L1, L3\}$
$A(unconsciousBreathing) = \{L2, L4\}$
$A(unconsciousLaryngospasm) = \{L1, L4\}$
$A(death) = \{\neg L1, \neg L2, *\}$

Behavior

Let firstly start to find E. As the first example was a bit simplistic, here we will introduce a new step in our process: finding internal variables. How to do that? Just by taking the monitored variables and the description of the system deduce whether it is an internal or external.

In our example, we have two internal variables: CO_2 in blood and O_2 in blood. These internal variables will be present in the form of ratios and can take values in the range $[0, 1]$. These variables are useful because they determine certain state transitions.

We have listed only three external events:

1. the person is underwater (obstruction from environment).
2. the person is emerged from the water (no obstruction).
3. conscious decision (voluntary holding/releasing breath).

We're not going to spill too much on the methodology used to construct state transition table. It is relatively easy but out of scope of this thesis. For more info, there are suitable books like [41].

Current state (internal)	Event (external)	New state
$normalBreathing$	e_1: underwater	$normalLaryngospasm$
$normalBreathing$	$O_2 \leq t_{O_2}$	$unconsciousBreathing$
$normalBreathing$	e_2: conscious decision	$consciousBreathing$
$normalLaryngospasm$	e_2: conscious decision	$consciousLaryngospasm$
$consciousLaryngospasm$	e_2: conscious decision	$normalLaryngospasm$
$consciousBreathing$	e_2: conscious decision	$normalBreathing$
$consciousBreathing$	$O_2 \leq t_{O_2}$	$unconsciousBreathing$
$consciousBreathing$	e_1: underwater	$consciousLaryngospasm$
$unconsciousBreathing$	$O_2 \geq t_{O_2}$	$normalBreathing$
$unconsciousBreathing$	$O_2 \leq t_{O_2} \wedge CO_2 \geq t_{CO_2} \wedge underwater$	$drowned$
$unconsciousBreathing$	$O_2 \leq t_{O_2} \wedge CO_2 \leq t_{CO_2} \wedge underwater$	$unconsciousLaryngospasm$
$normalLaryngospasm$	no obstruction	$normalBreathing$
$consciousLaryngospasm$	no obstruction	$consciousBreathing$
$unconsciousLaryngospasm$	no obstruction	$unconsciousBreathing$

where t_{O_2} is a specific threshold below which the person is unconscious, and where t_{CO_2} is a threshold beyond which the breathing reflex is triggered. We keep our E.

Note that, in case of drowning the water rushes into the lungs of the person, causing death.

We can add constraint using T function. For example:

$$|T(consciousbreathing)| \leq breath - holdbreakpoint$$

Note: Take advantage of different time scales

This example initially from Norbert Wiener[30], taken in extract from [36] allows us to realize the power of using time scales:

> *Different parts of a system, such as two feedback loops, can take advantage of different time scales. For example, one loop can work at short times and another at long times, thus avoiding interference. Or one loop can gather information using short times, and then pass this information to another loop that works at long times. Norbert Wiener [30] gives a simple example of a human driver braking an automobile on a surface whose slipperiness is unknown. The human tests the surface by small and quick braking attempts; this allows to infer whether the surface is slippery or not. The human then uses this information to modify how to brake the car. This technique uses a loop at a short time scale to gain information about the environment, which is then used for regulation at a long time scale. The fast loop manages the slow loop.*

4.1.3 Orchestration

Now that we have successfully designed each of them, we need to handle the interactions between feedback structures and coordinating their actions to avoid interference and to manage the whole system.

Ideally, the goal is to have minimal interaction. Regarding interactions, remember that *W* in WIFS means *weakly*. Therefore it should not there be a lot of links between our feedback structures. But some interaction always exists, therefore some coordination is always required.

Finding dependencies

We will not reinvent the wheel and use the methodology that our predecessors Peter Van Roy *em et al* left us [36] [34]: *For a successful orchestration, it is crucial to perform the right decomposition. [...] The interactions between feedback structures form a dependency graph, which is a directed graph whose nodes are feedback structures and where each edge indicates an interaction. Because interactions can be subtle [...], it is important to simplify them as much as possible at design time.*

A first simplify example can be a simplify version of our respiratory system.

$$S_{Respiratory} = S_{CO_2 stabilizing} \wedge S_{consciousness} \wedge S_{O_2 transition}$$

with the feedback structure previously designed responsible for $CO_2 stabilizing$, and let's assume a feedback structure *brain* that is responsible for *consciousness*, and another one, *lungs* that is responsible for $O_2 transition$.

We have these dependencies:

$S_{consciousness} \rightarrow S_{CO_2 stabilizing}$

$S_{CO_2 stabilizing} \rightarrow S_{O_2 transition}$

For the first one, we've seen previously that brain can trigger our breath. For the second one, we saw that it was the analyed feedback structure that send the air to the lungs and control breathing. There is therefore a dependency.

An other example is from Scalaris. We've seen previously that this system can be seen as a conjunction of properties:

$$S_{Scalaris} = S_{key-value} \wedge S_{connectivity} \wedge S_{routing} \wedge S_{load} \wedge S_{replica} \wedge S_{transaction}$$

Extract from paper from [34] and present in appendix A relieves these dependencies:

$S_{connectivity} \rightarrow S_{routing}$

$S_{connectivity} \rightarrow S_{replica}$

$S_{routing} \rightarrow S_{replica}$

$S_{routing} \rightarrow S_{load}$

$S_{replica} \rightarrow S_{transaction}$

For example from [36], the replicated storage has to take the routing into account. If a node fails, then the replicated storage has to create a new replica, but taking the routing into account which must also be repaired.

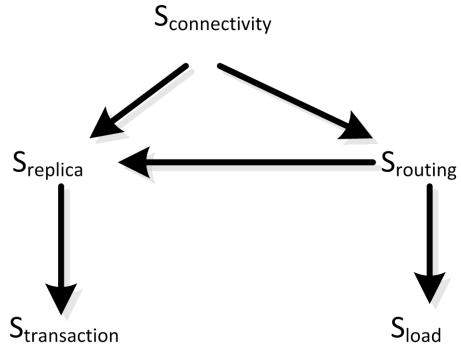


Figure 4.8: Schematic view of our dependencies

It's very instinctive to establish these dependency links. It's very instinctive to establish these links. We have to rely on the system description, trying to interact as little as possible feedback structures among-them.

Note: Extract

This extract is taken from [34]:

> *To reduce the interaction of feedback structures it is important to add them in the right order. If done correctly, each new feedback structure can be added in (almost) orthogonal fashion to the system. The acyclic part of the dependency graph can be ordered according to a topological sort. Cycles are handled separately.*

Analyzing dependencies

For dependencies, the channel of communication between feedback structures will be via their complex component. The idea is that passing through this component is the only way to interact with the feedback structure.

Regarding the analysis, we propose to create a table for each dependency 's link. We include both agents communicators (e.g. complex components) and note the event(s) associated with this link. We will illustrate this with the link $S_{consciousness} \rightarrow S_{CO_2 stabilizing}$ in this table[2]:

$Component_{S_{consciousness}}$	$Component_{S_{CO_2 stabilizing}}$	effect
rachidian bulb	conscious control of body and breathing	event e_2 : conscious decision

Once this step is completed, we have built our whole system.

4.2 Software Design Context

In this section, we are particularly interested in information systems and more particulary in Self-management systems. Is the methodology also works for these systems? Actually the methodology is the same as the previous section, except that we'll adapt it for self-management system and that we add an additional step: the *mapping*. The mapping is used to link *resources* and *feedback structures*. This step takes place after the orchestration.

But do not go too fast, first recall briefly what a self-managing application is.

[2]more info: http://www.cognifit.com/fr/parties-du-cerveau/

Note that a major part of this section is based on the excellent article by Ahmad Al-Shishtawy *em et al* called *A Design Methodology for Self-Management in Distributed Environments* [1].

4.2.1 Self-management

The selfmanagement initiative advocates self-configuring, self-healing, self-optimization and self-protection (also called *self-**).

We can extract three parts from a self-managing application: the functional part, the touchpoints, and the management part. The functional part of an application can be designed, for example, by defining interfaces, components, component groups, and bindings [1]. The touchpoints consists of a set of sensors and effectors used to interact with resources (get status and perform operations), for example *measure* O_2 in respiratory system. The management part is determined by management requirements.

Management part is the key, actually it can be decomposed into a number of managers. Each of them responsible for a specific self-* property or alternatively application subsystems [1]. We can easily make the connection with our methodology by decomposing the part of management like that:

$$S_{Self-Management} = S_{self-configuring} \wedge S_{self-healing} \wedge S_{self-optimization} \wedge S_{self-protection}$$

Based on that we can build feedback structures that will coordinate their activities in order to achieve management objectives. Each of these feedback structures will be implemented by *autonomic managers*.

4.2.2 Autonomic Manager

Definition This definition is from [1] :

> *An autonomic manager is the key building block in the architecture. Autonomic managers are used to implement the self-management behaviour of the system. This is achieved through a control loop that consists of four main stages: monitor, analyze, plan, and execute. The control loop interacts with the managed resource through the exposed touchpoints.*

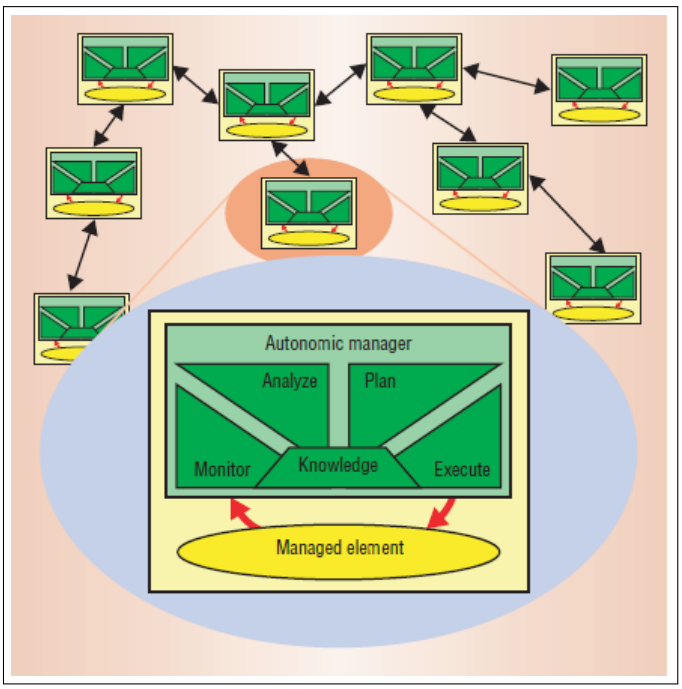


Figure 4.9: Structure of an autonomic manager. The whole is called autonomic element. Elements interact with other elements and with human programmers via their autonomic managers. [21]

Actually this is a particular feedback loop:

- *Monitor* matches *Monitoring agent*
- *Execute* matches *Actuating agent*
- *Analyze*, *plan* and *knowledge* matches *Actuating agent*
- *Managed element* matches *Subsystem*

As mentioned above, it is better to decompose management into several autonomic managers which cooperate. Indeed, according to [1] Decomposition can also be used to enhance the management performance by running different management tasks concurrently and by placing the autonomic managers closer to the resources they manage. Also it will increase scalability and robustness [1].

4.2.3 Methodology for self-managing application

We will use our iterative methodology and adapt it to self-management by working with autonomic managers.

Actually the methodology remains practically the same. The actions and objectives of stages are related to classical issues in distributed systems such as partitioning and separation of concerns, and optimal placement of modules in a distributed

environment[1]. Only two changes are observed: an additional main step (4 stages instead of 3) and the interactions between autonomic managers.

We will therefore first take the 3 main steps of our methodology and add one, then we will discuss about the interactions.

Decomposition

The idea is to decompose the management into properties. There are two ways for the decomposition.

- Functional : as previously done, we divide management based on self-* properties.
- Spatial: tasks are defined based on the structure of the managed application [1].

The criteria that decides whether we use a spatial or functional way is the fact that each task (or group of related task) must be performed by a single feedback structure.

Assignment

Each property is then assigned to a feedback structure. These feedback structure are composed of autonomic managers. Each of which becomes responsible for one or more management tasks.

Regarding the formation of autonomic managers, it is exactly the same process as the feedback loops.

Orchestration

As feedback structures manages the same system, there will necessarily be interactions and coordination between them. Therefore it avoids conflicts and interferences and it manages the system properly.

Do not forget that we need a minimum of interaction. So the interaction will be more inside of the feedback structure, at autonomic managers level (feedback loops).

Mapping

This is the new step and it is relatively simple. As said in [1], the set of autonomic managers are then mapped to the resources. A major issue to be considered at this step is optimized placement of managers and possibly functional components on nodes in order to improve management performance.

Interactions between autonomic managers

Here we will reformulate the patterns and interactions between autonomic managers. There are four way of interactions [1]:

1. Stigmergy: in self-management, agents are autonomic managers and the environment is the managed application.

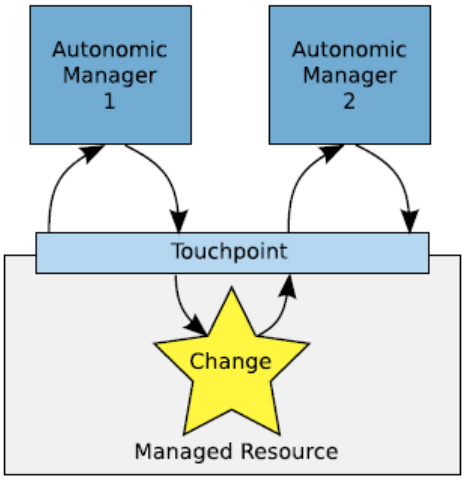


Figure 4.10: stigmergy effect [1]

According to [1], *the stigmergy effect is, in general, unavoidable when you have more than one autonomic manager and can cause undesired behaviour at runtime. Hidden stigmergy makes it challenging to design a self-managing system with multiple autonomic managers. However stigmergy can be part of the design and used as a way of orchestrating autonomic managers.*

2. Hierarchical Management: We keep our famous tower management. Communication between levels is done using touchpoints.

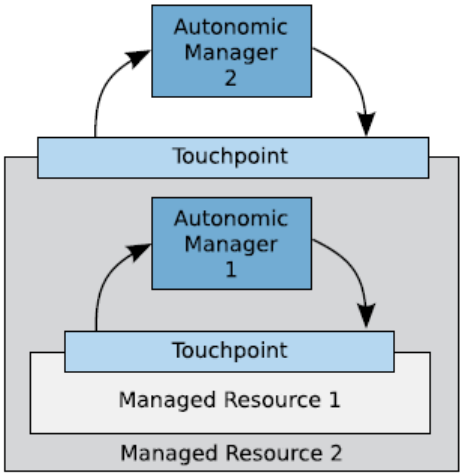


Figure 4.11: hierarchical management [1]

Note about time scales [1]: *Autonomic managers at different levels often operate at different time scales. Lower level autonomic managers are used to manage changes in the system that need immediate actions. Higher level autonomic managers are often slower and used to regulate and orchestrate the system by monitoring global properties and tuning lower level autonomic managers accordingly.*

3. Direct Interaction: We can have a direct interaction between autonomic managers by directly linking the appropriate elements between them.

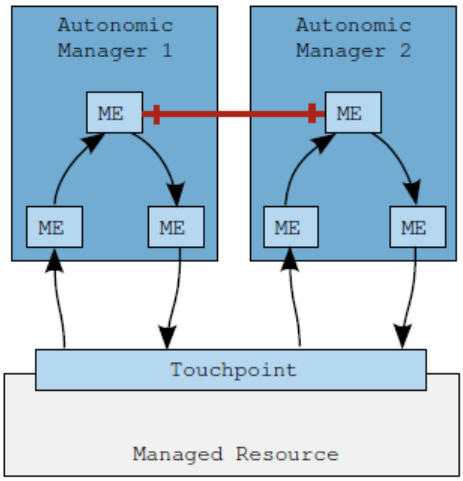


Figure 4.12: direct interaction [1]

According to [1], *cross autonomic manager bindings can be used to coordinate autonomic managers and avoid undesired behaviors such as race conditions or*

oscillations.

4. Shared Management Elements: the fourth way to communicate is that autonomic managers interacts by a sharing element.

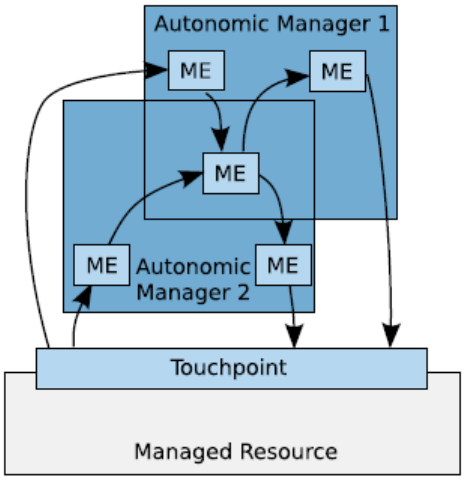


Figure 4.13: shared management elements [1]

This can be used to share state (knowledge) and to synchronise their actions [1].

Chapter 5

Relationships With Quantitative Techniques

Contents

The goal of this chapter is to show the relationship between a *global* and *qualitative* methodology (using weakly interaction feedback structures) and several *local* and *quantitative* methodologies (like control theory, system dynamics, model checking,...). These quantitatives methodologies can justify our qualitative work. It is important to formally justify our methodology to avoid unexpected behaviors.

The problem is that working with these quantitative methodologies require strong formalizations in order to use them (in opposition of feedback structures). Our methodologie can be used in complementary way to existing quantitative methodologies.

For example, in our operating space, there is a relationship between *qualitative behavior* and *quantitative behavior*. Qualitative behavior is derived from state space and dominant sets in feedback structures, quantitative behavior can be calculated within

an area in the operating space that corresponds to a single state. Quantitative behavior can be calculated with a quantitative method such as control theory or system dynamics, since these disciplines are designed for a fixed set of feedback loops.

Note from [34]

> *Systems can show unexpected behavior that becomes clear only through formal analysis. For example, techniques from theoretical physics have been used to show that structured overlay networks exhibit phase transitions when network communications become very slow [23].*

It is clear that our methodology can be linked with many disciplines. We will discussed them one by one. As stated above, they are complementary to our methodology.

5.1 System Dynamics

We are in a quantitative approach. System dynamics allows robust simulation *if* the modelization is precise enough. We will, as a first step, look at a specific example of System Dynamics. After that we will deduce in a concrete way the limits of this approach. Finally we will link these approach and our methodology and we will see how the use of Weakly Interacting Feedback Structures may improve the use of System Dynamics.

5.1.1 A System Dynamics example

Our example is from the book *Dynamic Modeling of Diseases and Pests*[13]. It is a model dealing with Lyme disease.

> *Over the past 20 years since Lyme disease was first diagnosed, it has been identified as the most common vector-borne disease in the United States. The repopulation of white-tailed deer in the United States of America has been associated with the emergence of this disease. The tick vector, Ixodes scapularis, harbors Borrelia burgdorferi (B.b.), the organism responsible for Lyme disease[33]. The larval and nymphal stages feed on intermediate hosts, which are mostly small mammals and birds. The adult tick prefers to feed on deer, but will also feed on dogs and people. The main intermediate host in the northeast United States is the white-footed mouse. Mice and chipmunks may serve as reservoirs for B.b. in nature since they maintain active infections for at least 3 to 4 months. In the Midwest, however, it has been determined that the eastern chipmunk may be equally important as an intermediate host. The ticks appear to follow the migration of deer but deer may be simply an*

amplification host; they are able to clear infection from B.b. within a few days.

This was for the introduction. Now we will skip the development step (finding the different parameters and then assigning to quantitative values to these parameters, given hypothesis). In fact, what interests us is not the development by using System Dynamics approach (beyond the scope of this paper), but the final result, and in particular, the final graph.

Here is the graph from[13]:

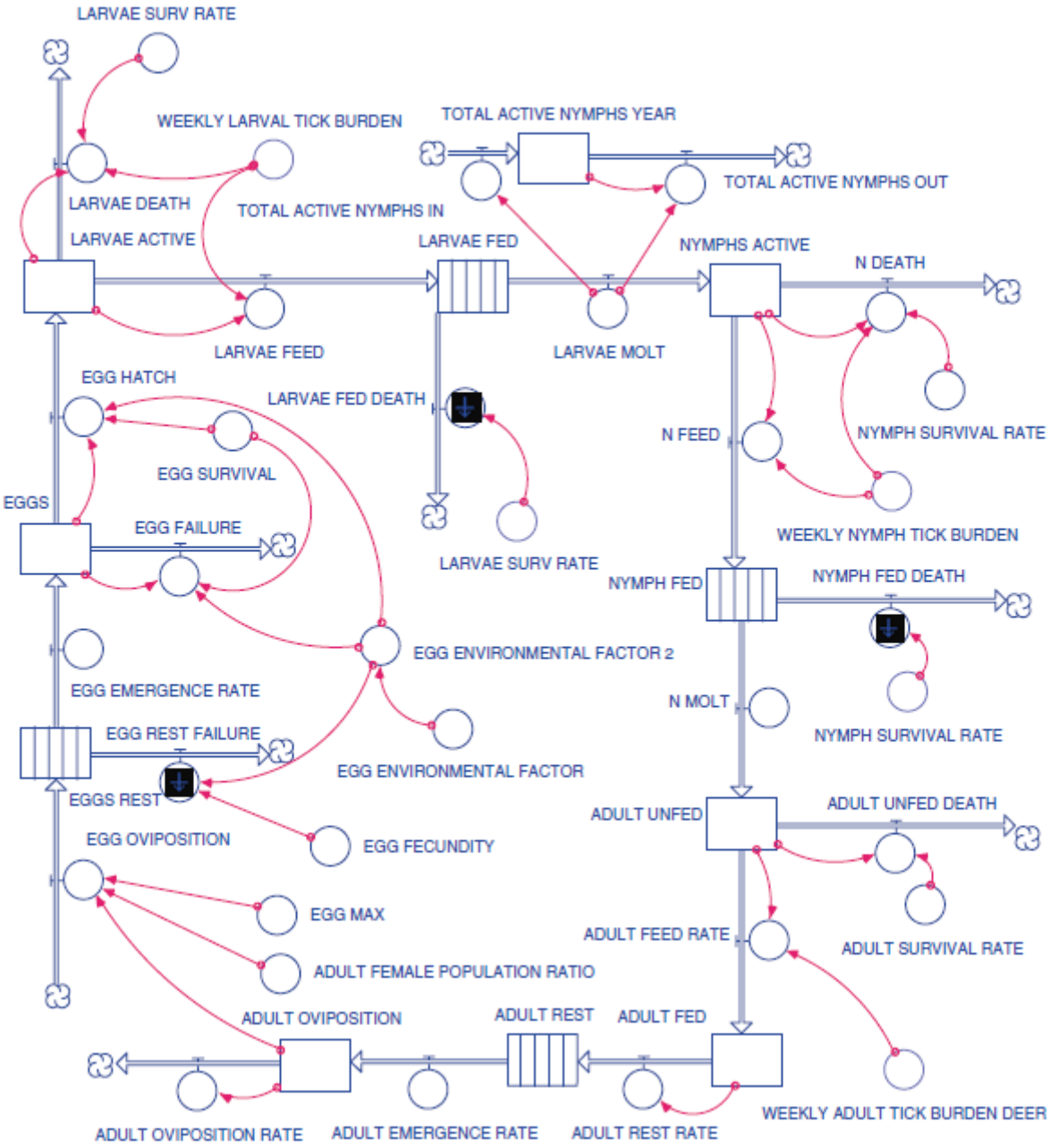


Figure 5.1: Population dynamics of the blacklegged tick

> *Ticks move in sequential fashion from egg to larva, to nymph, to adult during a 2-year period in which they are in resting stages prior to molting to the next life stage. Conveyors have been used to simulate the rest periods. Additionally, we assumed that intermediate and definitive host densities account for the carrying capacity and feeding success of the tick stages dependent on a blood meal to molt.*

The parameters and variable names are quite eloquent and allow to realize easily without much notions of what the System Dynamics approach is.

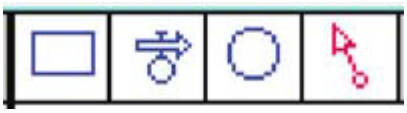


Figure 5.2: The four following diagrams reprensents respectively *building blocks*, for stocks, *flows*, *converters*, and *connectors* (information arrows).

Now that we have this, we can proceed to the next step: identify the limitations of this approach.

5.1.2 Limit of the model

This model provides us interesting strong results. But it is only valid for a given environment. A change of environment, let's assume a change of climate, can affect the overall results[6]. It has shown that the temperature and humidity strongly influenced the behavior of ticks carrying Lyme: they are twice more mobile, and are actively looking for prey when it is hot and dry. The fact that this model does not consider climate change makes it less realistic.

Indeed, suppose a sudden change in climate. Each tick will have change his behavior because it is a new environment. This change in behavior will not be seen in the System Dynamics model because it considers only one behavior.

Actually the different loops that constitute a graph of System Dynamics are always activated. But this is not the case in reality. The first behavior of a tick female is *laying*, but during freeze period his first behavior becomes *survival*. Actually several weeks may pass before the female begins to lay eggs if conditions are not favorable. It may happen that the female dies before laying. So the idea is that in reality, there is an activation/deactivation of loops in a system after a change of environment, that there is no in System Dynamics.

This is a general criticism of System Dynamics. Usually this approach build a system with lots of loops that only work in a part of the environment. If an environmental change occurs, even if it is small, it can distort the whole system. There is no adaptibility in System Dynamics.

To remedy this, we might consider breaking the System Dynamics in subsystems and making them interact according to environment changes, but unfortunately the concept of system decomposition is not clear in System Dynamics

5.1.3 Links with our methodology

How can we make the System Dynamics approach (which despite its flaws is still a very interesting quantitative approach) adaptive by combining it with our approach with Weakly Interacting Feedback Structures?

In Feedback Structures, it is noted that according to a given environment, we have a superposition of feedback loops. These superposition is actually a combination of active feedback loops. **For each of these combination will correspond one System Dynamics graph**. For example, let's take a simplify feedback structure like this:

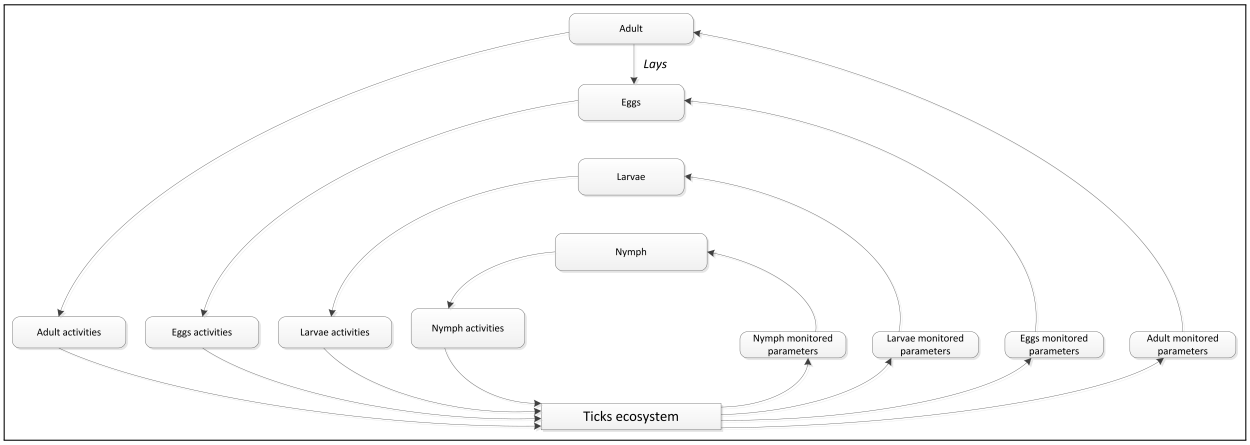


Figure 5.3: Feedback Structure of blacklegged tick ecosystem in normal case

This graph corresponds to what we have seen in System Dynamics example (without detailing all quantitative data, variables and parameters). There is a synergistic interaction between the differents development stages of the tick via the ecosystem. The only direct interaction (direct management) is when the adult tick lays eggs, she then directly determines the amount of eggs.

Now let's assume that, as in our example of the limitation of the System Dynamics approach, the environment is changing and we are in a big freeze period. What

happens at the feedback structure level? As the adult female tick stops laying to concentrate on surviving the cold, there is a change in the activation of feedback loop. There is no more direct link from the adult tick to eggs.

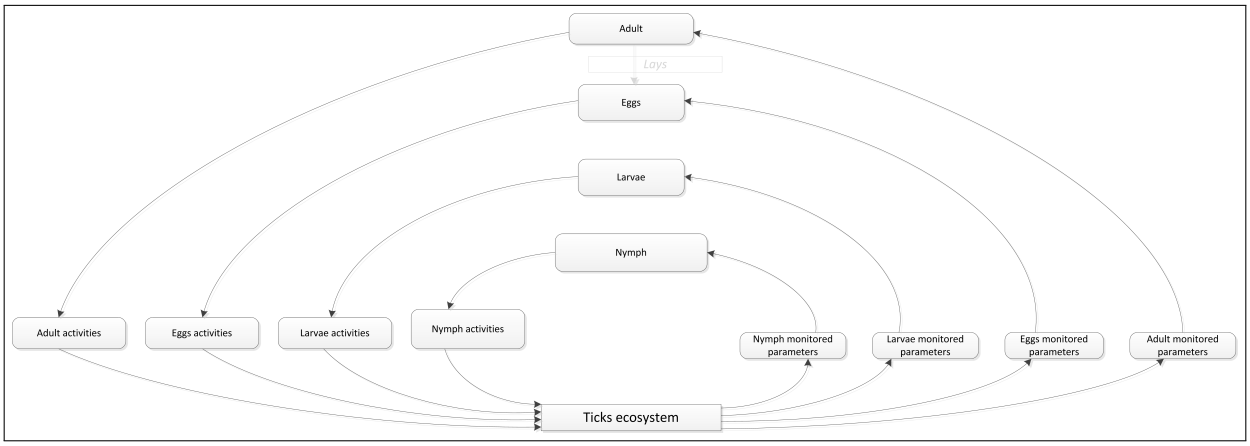


Figure 5.4: Feedback Structure of blacklegged tick ecosystem in frozen period

For this Feedback Structure graph, there is a corresponding graph in System Dynamics which can be totally different from the one we saw. Indeed, the cycles are not the same (eg adults no longer lay) and the values are also different (for example ticks, whatever their stage of development, are twice less active in cold weather).

So to summarize what has been said, each System Dynamics graph can be considered as a specific combination of feedback loops into a feedback structure.

Another possibility?

The notion of system decomposition is not very clear in System Dynamics. But this decomposition is possible and would be useful because when we look at the graphs, we notice that they are usually very (too?) large. If we can find a reliable decomposition methodology, we could then scale up and considering that a System Dynamics graph can be partitioned. For each of these part can correspond a Feedback Structure graph and each part could slightly interact with other (weakly interacting). Then a System Dynamics graph could be seen as a Weakly Interacting Feedback Structures.

5.2 Control Theory

Control theory has a mathematical foundation with many theoretical results *if* the systems obey certain formal rules. Control theory is used to maintain a goal. Examples are varied:

- On a commercial airplane the vertical acceleration should be less than a certain value for passenger comfort.[18]
- All new cars are, for some years now, equipped with an electronic control unit that manages, in addition of the safe part of the engine, the engine management part that includes the injection control or also the control of the turbo.

We will directly go to the example without dwelling on the mathematical development who, although it is the key in Control Theory, requires too many concepts that are beyond the scope of this thesis.

5.2.1 A Control Theory example

This simple example is from [18]. This is about *Keck I* telescope. At the time of the example (1990), it was not yet built[1]. It illustrates well what the Control Theory is without going into the details of the mathematical development:

> *A very interesting engineering system is the Keck astronomical telescope, currently under construction on Mauna Kea in Hawaii. When completed it will be the world's largest. The basic objective of the telescope is to collect and focus starlight using a large concave mirror. The shape of the mirror determines the quality of the observed image. The larger the mirror, the more light that can be collected, and hence the dimmer the star that can be observed. The diameter of the mirror on the Keck telescope will be 10m. To make such a large, high-precision mirror out of a single piece of glass would be very difficult and costly. Instead, the mirror on the Keck telescope will be a mosaic of 36 hexagonal small mirrors. These 36 segments must then be aligned so that the composite mirror has the desired shape.*
>
> The control system to do this is illustrated in the figure above.

[1] Keck I was completed in 1993.

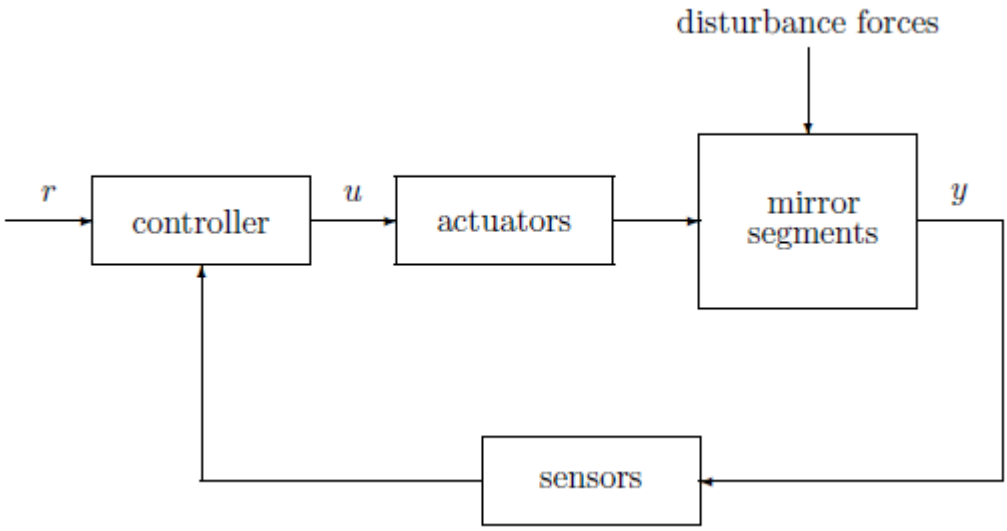


Figure 5.5: Block diagram of Keck telescope control system [18]

As shown, the mirror segments are subject to two types of forces: disturbance forces (described below) and forces from actuators. Behind each segment are three piston-type actuators, applying forces at three points on the segment to effect its orientation. In controlling the mirror's shape, it succes to control the misalignment between adjacent mirror segments. In the gap between every two adjacent segments are (capacitor-type) sensors measuring local displacements between the two segments. These local displacements are stacked into the vector labeled y; this is what is to be controlled. For the mirror to have the ideal shape, these displacements should have certain ideal values that can be pre-computed; these are the components of the vector r. The controller must be designed so that in the closed-loop system y is held close to r despite the disturbance forces. Notice that the signals are vector valued. Such a system is multivariable.

Our uncertainty about the plant arises from disturbance sources:

- *As the telescope turns to track a star, the direction of the force of gravity on the mirror changes.*
- *During the night, when astronomical observations are made, the ambient temperature changes.*
- *The telescope is susceptible to wind gusts.*

and from uncertain plant dynamics:

- *The dynamic behavior of the components-mirror segments, actuators, sensors-cannot be modeled with infinite precision.*

At this point, we understand the usefulness of the Control Theory and how the development process will be managed. There will be a mathematical development that which will lead to find an optimized solution for this system.

5.2.2 Limit of the model

As we have seen, the Control Theory is primarily used to optimize complex systems. The calculations used for this optimization take a while. And for this reason we rarely see systems with more than one feedback loop. There are some rare examples with two loops that we can see in the book *Feedback Control of Computing Systems*[19] [2].

So a first limit of this model is that **it only works with small complex systems**. This is unmodeled when the system becomes large.

A second limit is that it works only in a part of space. For example let's take a fan motor, the fan is optimized specifically for a certain type of engine with a determined cylinder capacity. It is optimized for some specific behavior. If the fan is placed on a new motor (different from the previous one), it is likely that this fan is not optimized for this motor and his behavior. And personally, I would not take the risk of driving a car with this included...

We can conclude that a second limit of this model is that **systems with Control Theory are only optimized in some part of space**, there are other situations in which these systems are not optimized.

5.2.3 Links with our methodology

As previously stated, Control Theory is usefull typically to optimize throughput or some other quantitative property. Let's take our previous example and put it under the form of a feedback loop.

[2] chapter 10, last chapter of the book

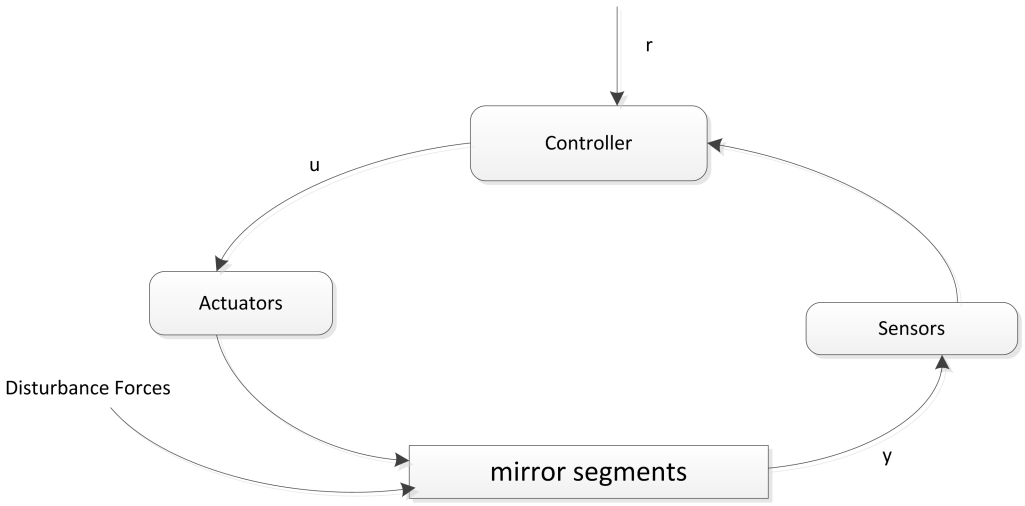


Figure 5.6: Keck I as a feedback loop

By this example, we see that we can easily do the matching between the feedback loops and the schemes of Control Theory.

The idea to link Control Theory and Weakly Interacting Feedback Structures is to apply Control Theory to each feedback loop. As applying the theory of control takes a subsequent time, we will only apply this theory on feedback loops that need to be optimized.

- In the example of breathing system, the goal is to optimize the O_2 and CO_2 concentration in blood. So if the goal is to optimize this system, we will apply the Control Theory only on two of four feedback loops.
- In the chess player example, we will only apply Control Theory on inner loop which is to optimize the chess strategy of the player. Indeed, we don't think that eating at this time or another will affect our chances of winning. The important is to eat to keep the clearness, that's all.

Applying Control Theory on each feedback loops works well when the feedback loops are mostly independent. Actually, the less feedback loops are independent, the most there are parameters to be considered in the optimization.

5.3 Model Checking

The Model Checking is a family of techniques for automatic verification of dynamic systems. It is algorithmically verify whether a given model, the system itself or an abstraction of the system satisfies a specification, often phrased in terms of temporal logic.

There is a big difference between *Model Checking* and *Testing* because the testing does not check everything but takes only a sample. Testing complex systems can easily turn into finding a needle in a haystack.

Quote

"*Program testing can at best show the presence of errors but never their absence.*" Edsger Dijkstra.

Model Checking as a *formal method* does not depend on guesses. At least as the theory goes, if there is a violation of a given specification, model checking will find it. Model checking is supposed to be a rigorous method that exhaustively explores all possible behaviors [20]. If there is only one little bug, Model Checking will find it. There is no probability in this model and that's a good thign because software robustness is not based on probability.

The idea is to consider all possible executions of the system. It takes time but we can reduce this time by taking some shortcuts and algorithms proper to model checking (out of scope, [53] and [10]). This is a quantitative way to verify if a system works.

5.3.1 Limitations of using the Model Checking

Model Checking allows verification *if* the formalization is precise enough. Sometimes systems can not be so easily formalized and extracting rules and syntax may be hard to do. Let's take the immune system, we can easily modeling it qualitatively but it's hard to find a right and strong formalism that can apply Model Checking.

5.3.2 Using Model Checking in WIFS

Actually, the right way to study discrete systems such as feedback structure is by modeling each component in a feedback structure as a state machine. The interacting components are then modeled as interacting state machines. **The way to prove properties of such systems** (for example, that they are robust) **is actually by using Model Checking**. By this way we'll do exhaustive analysis of all the relevant behaviors. As previously said, no assumptions on probability distributions are made, so the results are always correct.

We have seen that it's hard to verify a big system like Weakly Interacting Feedback Structures. The right way to proceed is to verify locally. Indeed, when a feedback structure is in some state, we can verify by Model Checking. For each feedback structure, there is a state diagram. So from each of these state diagram (e.g from each part of WIFS), we can apply Model Checking.

Chapter 6

Rules Application

Contents

This chapter consists of examples that implement our methodology. As the design of a feedback structure takes non-negligible time, we prefer to deal with several feedback structures from different areas rather than dealing with multiple feedback structure

from a same area and build a WIFS. Therefore, we'll focus strongly on the second step of our process which involves the construction of feedback structures.

6.1 TCP

We start with a first example IT-oriented: the *TCP* protocol. We briefly recall its principle in the system description. The idea is not to be focused on the protocol, the format and structure of a TCP segment, but rather be focused on the behavior of this system.

6.1.1 System description

We identify five properties for *Transmission Control Protocol*, we will briefly describe them one by one.

1. **Reliability of transfers**: $P_{reliableTransfer}$. Actually, the TCP has a system of acknowledgment that allows the client and server to ensure proper mutual reception of data.
2. **Connection establishing**: $P_{connectionEstablishement}$. Both machines must synchronize their sequences through a mechanism called three way handshake.

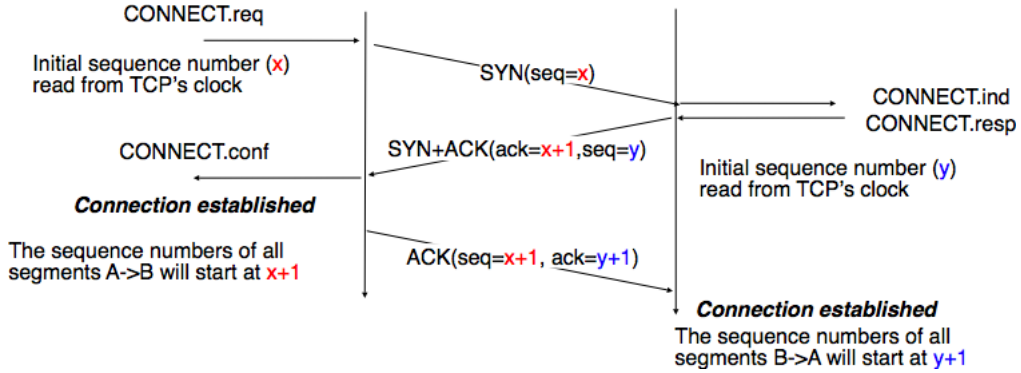


Figure 6.1: Establishment of a TCP connection [5]

3. **Sliding window**: $P_{slidingWindow}$. In many cases, it is possible to limit the number of acknowledgments by setting sequence number after which an acknowledgment is required. Sliding window defines a range of sequences that do not require acknowledgment, and it moves as and when acknowledgments are received. In addition, the size of this window is not fixed.
4. **Connexion releasing**: $P_{connectionRelease}$. The receiver can request to end a connection as well as the sender. The mechanism is again three way handshake.

5. **Congestion**: $P_{congestionManagement}$. The size of the *sliding window* will change depending on whether or not segments transfer are successful.

6.1.2 Behavior identification

Instinctively, we feel that we can firstly regroup two of these properties: Establishing and releasing the connection can form the *connection management*. We have then three properties:

$$S_{TCP} = P_{connectionManagement} \wedge P_{slidingWindow} \wedge P_{reliableTransfer} \wedge P_{congestionManagement}$$

Each of these four properties will be implemented by a feedback loop. Feedback loops will form together a feedback structure.

Here are the elements on which we will determine the states: the *transfer*, the *sliding window* and the *connection*.

- Connection can be *established* or *not established*.
- The state of the sliding window depends on its length (L_{SW}): $L_{SW} = 0$ or $L_{SW} > 0$.
- Transfer can be *ongoing* or *stopping*.

We deduce that there are 8 possible combinations of states (2^3). This does not necessarily mean that there are 8 possible *behaviors*. In fact, according to system specifications, some combinations are not possible: for example transfert ongoing while connection is not established is impossible.

Connection	L_{SW}	Transfer	possible?	name
not established	$= 0$	*stopping*	V	**Init**
not established	$= 0$	*ongoing*	X	
not established	> 0	*stopping*	X	
not established	> 0	*ongoing*	X	
established	$= 0$	*stopping*	V	**Transfer** (sliding window = 0)
established	$= 0$	*ongoing*	X	
established	> 0	*stopping*	X	
established	> 0	*ongoing*	V	**Transfer**

We have finally three possible behaviors.

6.1.3 State diagram D

Here is our state diagram with death state and init state added:

$S =$

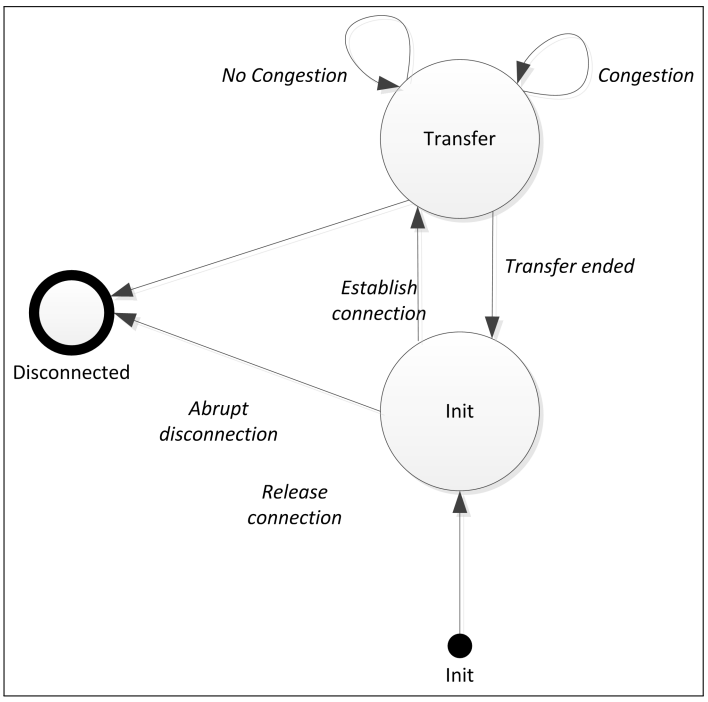


Figure 6.2: State diagram of a simplify TCP

We obtains $D = (S, init = Init, death = Disconnected)$.

6.1.4 Building graph G

Let's build the feedback loops one after one.

Connection

The monitoring agent checks for *connection* segments (SYN, FIN, ACK). It also regularly watch if there was no abrupt disconnection.

The correcting agent checks the validity of the segments and decides which segments will be sent in response.

The update agent sends segments and apply possible change of state.

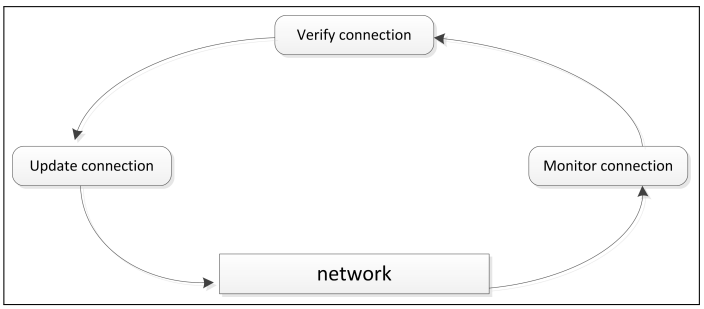


Figure 6.3: Connection feedback loop

Transfer

The monitoring agent checks for segments into packets.

The correcting agent prepares the new segment to send.

The update agent sends segments and apply possible change of state.

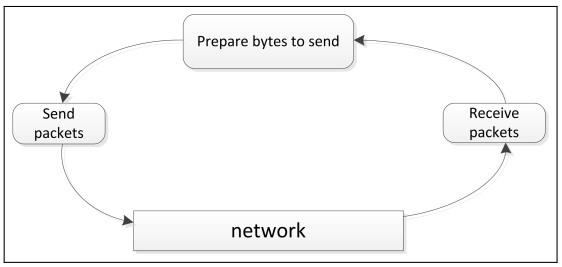


Figure 6.4: Transfer loop

Sliding Window

The sliding window determines which packet to send (a hole in the sliding window), which packet to give to the application and when sliding the window (to the left, when the packet is given to the application).

As there is a direct management interaction between the sliding window and the transfer feedback loop (the state of the sliding window determines whether there is transfer or not), we can apply the *fail safe* pattern.

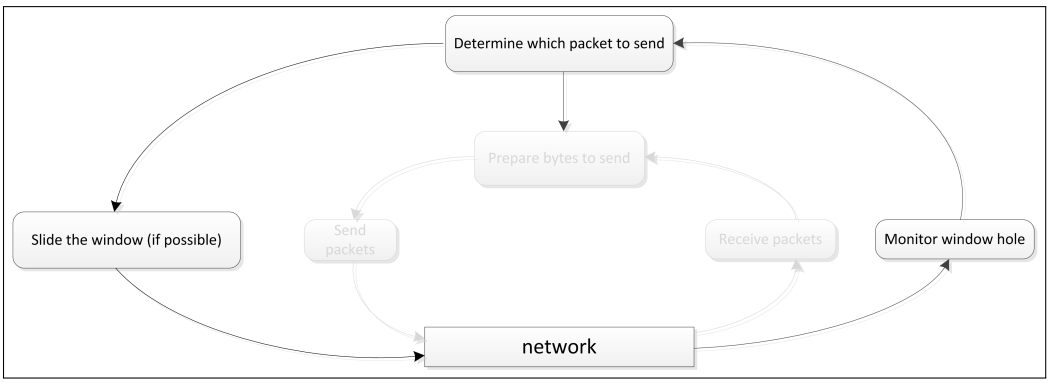


Figure 6.5: Safe fail pattern application for the *sliding window* and *transfer* feedback loops

Window Management

There is also a direct management interaction between the sliding window and the window management, we can apply the *fail safe* pattern.

We have a monitoring agent that monitors the throughput.

The correcting agent calculate policy modification and determine if the transfer is allowed or not.

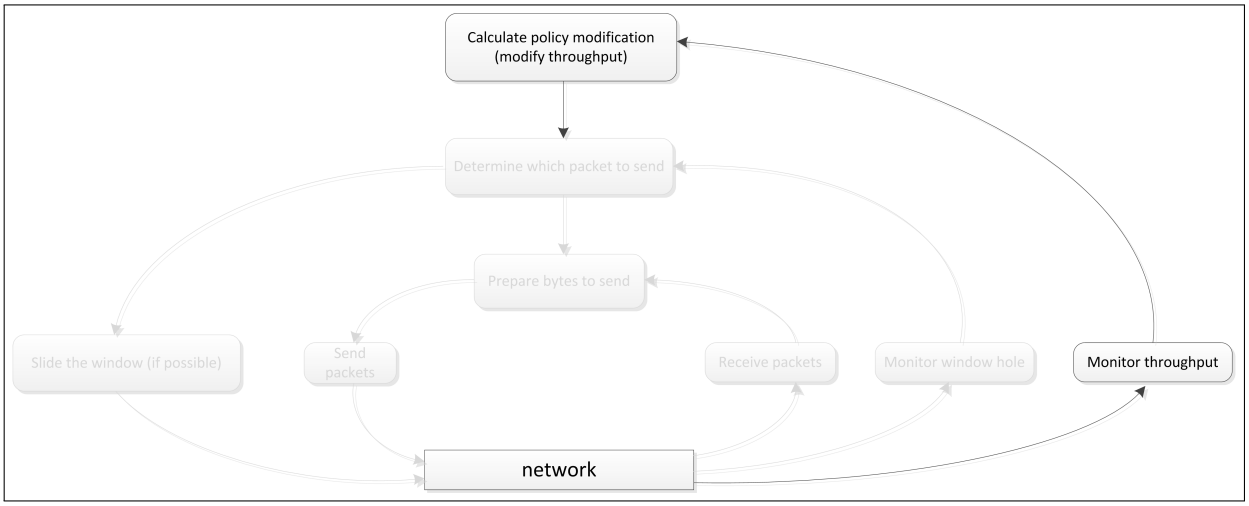


Figure 6.6: Safe fail pattern application for the *window management* and *sliding window* feedback loops

Final graph

The feedback loop *connection* interacts by stigmergy with the previous feedback loop, we have our feedback structure graph by assembling these feedback loops.

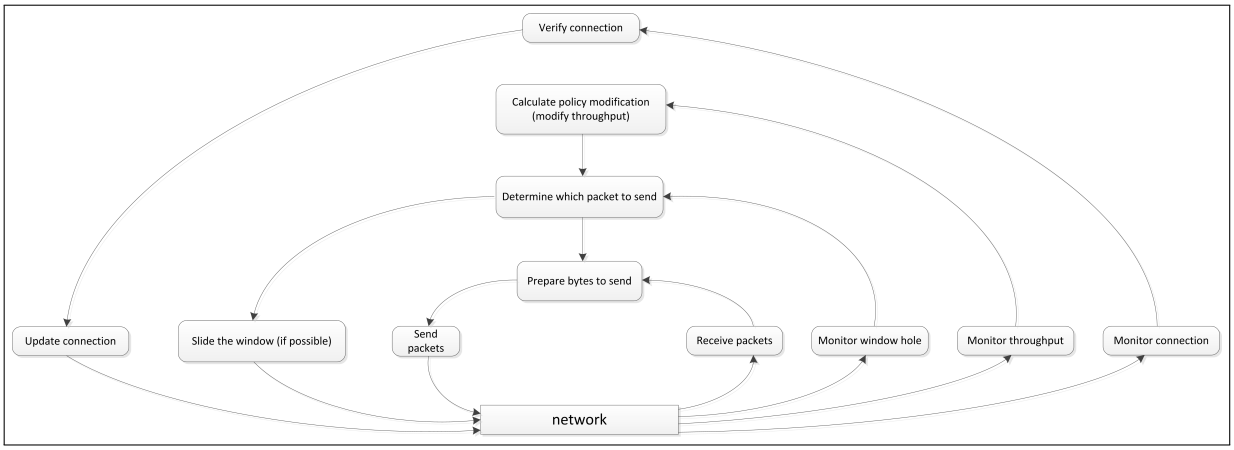


Figure 6.7: Graph G of TCP feedback structure

In the TCP/IP implementation, an inner loop handles the sliding window protocol and an outer loop manages the inner loop to control congestion. The outer loop does this in a simple way: it changes the size of the sliding window. This changes the bandwidth needed by the inner loop, since the size of the liding window determines the number of packets that can be in transit at any moment in time. This is an example of a natural management: the outer loop adjusts a simple parameter of an inner loop. No other interaction is needed. [37]

6.1.5 Finding A

Let's build the combination array. The possible states are the following: { *Init*, *Transfer*, *Disconnected* }

To save space in the table, we'll link *sliding window* and *window management*, because one can not work without the other.

Connection	**Sliding window**	**Transfer**	corresponding state
0	0	0	Disconnected
0	0	1	Disconnected
0	1	0	Disconnected
0	1	1	Disconnected
1	0	0	Init
1	0	1	*none* (no throughput with $L_{SW} = 0$)
1	1	0	Transfer
1	1	1	Transfer

6.1.6 Behavior: E

Internal variable:

- L_{SW}
- $ConnectionReleasing$

External events:

- e_1: data segment received
- e_2: transfer complete
- e_3: valid SYN+ACK segment receive
- e_4: valid ACK segment receive
- e_5: valid FIN+ACK segment receive
- e_6: valid FIN segment receive
- e_7: abrupt disconnection

E

Current state (internal)	Event (external) and/or condition	New state
$Init$	e_3: valid SYN+ACK segment receive	$Transfer$
$Init$	e_4: valid ACK segment receive $\wedge ConnectionReleasing = 0$	$Transfer$
$Init$	e_5: valid FIN+ACK segment receive	$Disconnected$
$Init$	e_4: valid ACK segment receive $\wedge ConnectionReleasing = 1$	$Disconnected$
$Transfer$	e_1: data segment receive	$Transfer$
$Transfer$	$L_{SW} = 0$	$Transfer$
$Transfer$	e_2: transfer complete	$Init \wedge$ $ConnectionReleasing = 1$
$Transfer$	e_7: abrupt disconnection	$Disconnected$
$Transfer$	$L_{SW} > 0$	$Transfer$

6.2 Thermoregulation in human body

We will describe in some detail the phenomena to ensure consistent body temperature because it is a good example of hypothalamic control level. In mammals, this consistency is essential for the proper functioning of the body: the speeds of all the biochemical reactions are temperature dependent, and for us, the system works best between 37 and 38 °C.

Description of the system that we will see is inspired by [25]. In this system, the *hypothalamus* has a key role.

6.2.1 System description

The body regulates its central temperature through two main mechanisms: the control of the production of heat and the control of heat loss.

It is mainly through the somatomotor system that the human ensures production of additional heat in the thermoregulatory phenomena: the thrill accelerates the metabolism. Heat loss is set by regulating the cutaneous circulation. Excess heat generated in the body is transported by the blood flow to the skin and there it is dispersed into the environment by radiation. Heat loss is set by regulating the cutaneous circulation. Excess heat generated in the body is transported by the blood flow to the skin and there it is dispersed into the environment by radiation.

The evaporation of sweat actively secreted by the skin is an important mechanism for cooling, especially when the ambient temperature is high. In addition to these mechanisms, sensations of heat or cold can induce specific behavior, such as extreme temperatures avoidance, the choice of particular garments, etc.. This type of behavior can obviously be considered as part of the regulatory mechanisms of the body temperature.

If the body *knows* when to lose heat and when it should happen, is that he has receptors (thermosensitive elements) capable of measuring temperature. Such receptors are found in the anterior region of the hypothalamus and in the skin. Those of the anterior hypothalamus are specialized neurons that measure the core temperature rises (warm-sensitive neurons). Receptors in the skin are cold-sensitive neurons that detect the lowering of the surface temperature. Thus, any drop in temperature comes to knowledge centers through the cold receptors before the core temperature has been reduced. The cutaneous receptors sensitive to hot probably did not matter in thermoregulation. [51]

Information from hot-sensitive and cold-sensitive neurons converge on the center of control of the hypothalamus. The hypothalamus then adjusts the heat production or loss.

Elevation of body temperature can cause an increase in skin blood flow and sweat emission, these two phenomena contribute to heat loss. An increase in the activity of the sweat glands fibers increases the secretion of sweat. A decrease in the activity of fibers that innervate the cutaneous blood vessels, causes expansion of these and thus increasing blood flow at this level.

When these receptors are activated by a drop in peripheral temperature, then the heat production is increased by increasing the metabolic activity and heat loss is reduced by decreased blood flow to the skin. There is also provocation of *shivering*:

the body will activate all the muscles under the skin to warm the skin without the need to use hot-blooded, focusing this time on the vital organs.

When the temperature difference becomes too high, our body can be in extreme state: hypothermia[50] or hyperthermia[49]. In this cases, we are unconscious.

Here is a shematic view of our description[1]:

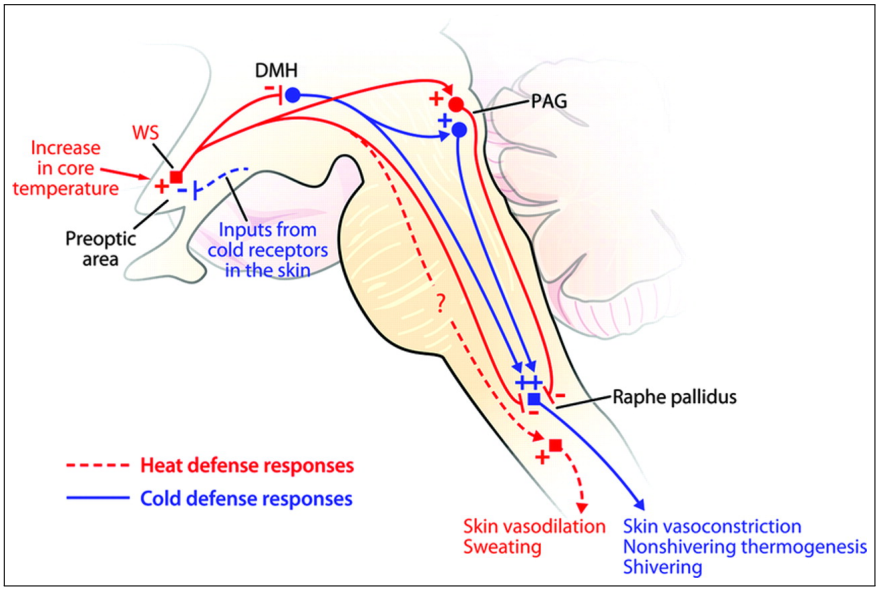


Figure 6.8: Thermoregulation: Anatomy diagram representing the relationship between the most important thermal regulation items

Obviously this is a rather simplistic description of the system. We should involve other regions of the central nervous system such as spinal and brainstem which are also thermosensitive and thermoregulatory functions: biological thermoregulation is considerably more complex than it seems in the figure 6.8.

6.2.2 Behavior identification

There are four different elements:

- activated or avoided sweat production
- increased, decreased or normal blood flow
- activated or avoided shivering
- conscious or unconscious

Increased blood flow and sweat production are bonded, same for decreased blood flow and shivering.

[1]picture taken from www.neurology.org

Consciousness	Sweat	Blood flow	Shivering	possible?	name
yes	*no*	*decreased*	*no*	X	
yes	*no*	*decreased*	*yes*	V	**fighting against heat loss**
yes	*no*	*normal*	*no*	V	**normal**
yes	*no*	*normal*	*yes*	X	
yes	*no*	*increased*	*no*	X	
yes	*no*	*increased*	*yes*	X	
yes	*yes*	*decreased*	*no*	X	
yes	*yes*	*decreased*	*yes*	X	
yes	*yes*	*normal*	*no*	X	
yes	*yes*	*normal*	*yes*	X	
yes	*yes*	*increased*	*no*	V	**fighting against heat raise**
yes	*yes*	*increased*	*yes*	X	
no	*no*	*decreased*	*no*	X	
no	*no*	*decreased*	*yes*	V	**hypothermia**
no	*no*	*normal*	*no*	X	
no	*no*	*normal*	*yes*	X	
no	*no*	*increased*	*no*	X	
no	*no*	*increased*	*yes*	X	
no	*yes*	*decreased*	*no*	X	
no	*yes*	*decreased*	*yes*	X	
no	*yes*	*normal*	*no*	X	
no	*yes*	*normal*	*yes*	X	
no	*yes*	*increased*	*no*	V	**hyperthermia**
no	*yes*	*increased*	*yes*	X	

6.2.3 State diagram D

S

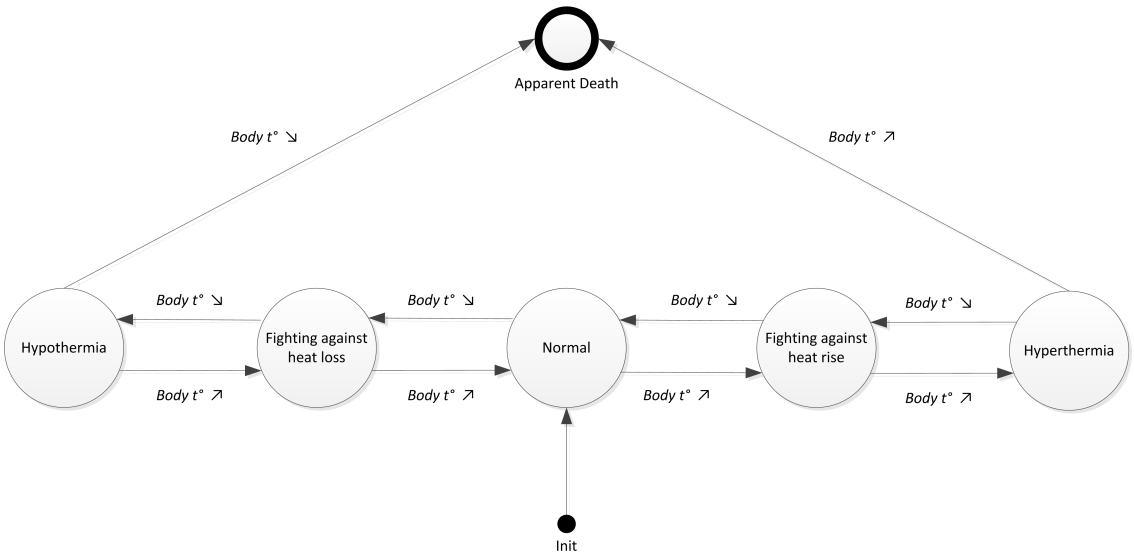


Figure 6.9: State diagram of our thermoregulation

$D = (S; init = Normal; death = ApparentDeath)$

Based on [51], we can determine the core temperature range of these states:

Apparent death	$]41.5°C; [$
Hyperthermia	$]38.3°C; 41.5°C]$
Fighting against heat raise	$]37.5°C; 38.3°C]$
Normal	$]36.5°C; 37.5°C]$
Fighting against heat loss	$]35.0°C; 36.5°C]$
Hypothermia	$]30.0°C; 35.0°C]$
Apparent death	$]; 30.0°C]$

6.2.4 Building graph G

Here is the final graph:

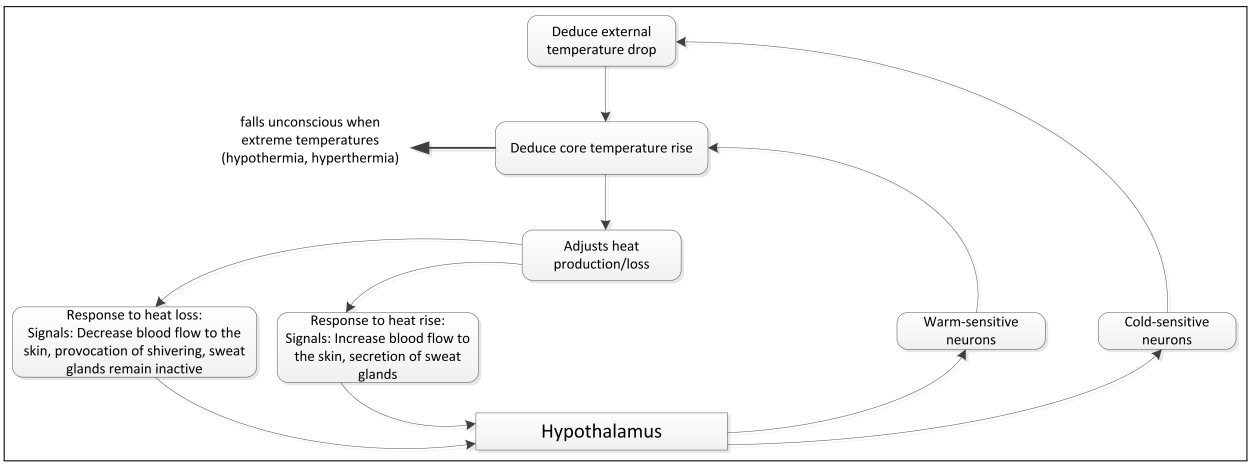


Figure 6.10: Graph G of a hypothalamus feedback structure

6.2.5 Finding A

We first identify the different loops from the graph G:

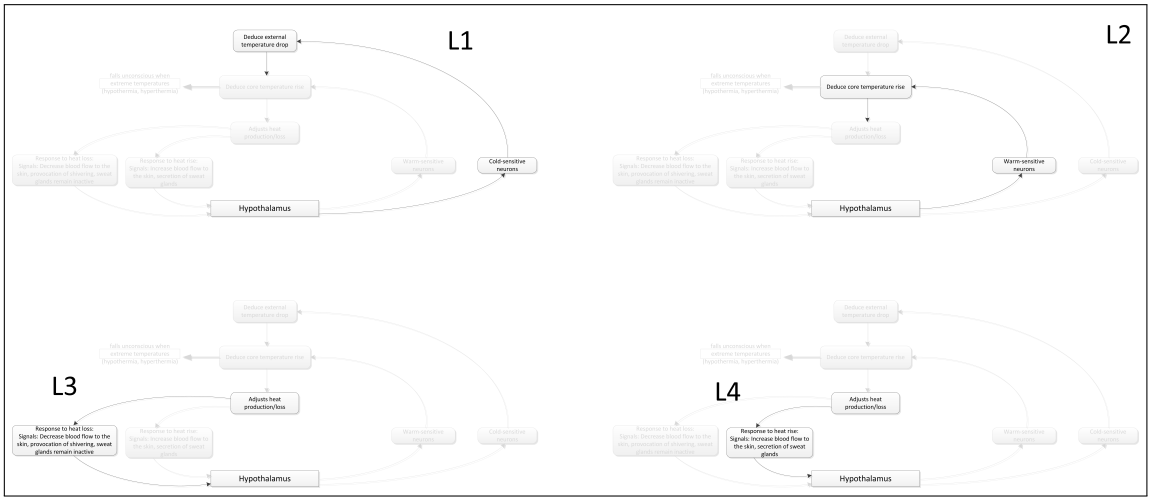


Figure 6.11: Graph G of a hypothalamus feedback structure

Now, let's build the array. Possible states are the following: $\{ Normal, FightingAgainstHeatLoss, Hypothermia, FightingAgainstHeatRise, Hyperthermia, ApparentDeath \}$

L1	**L2**	**L3**	**L4**	**Conscious**	corresponding state
0	0	0	0	0	*ApparentDeath*
0	0	0	0	1	*none*
0	0	0	1	0	*none*
0	0	0	1	1	*none*
0	0	1	0	0	*none*
0	0	1	0	1	*none*
0	0	1	1	0	*none*
0	0	1	1	1	*none*
0	1	0	0	0	*none*
0	1	0	0	1	*none*
0	1	0	1	0	*none*
0	1	0	1	1	*none*
0	1	1	0	0	*none*
0	1	1	0	1	*none*
0	1	1	1	0	*none*
0	1	1	1	1	*none*
1	0	0	0	0	*none*
1	0	0	0	1	*none*
1	0	0	1	0	*none*
1	0	0	1	1	*none*
1	0	1	0	0	*none*
1	0	1	0	1	*none*
1	0	1	1	0	*none*
1	0	1	1	1	*none*
1	1	0	0	0	*Normal*
1	1	0	0	1	*none*
1	1	0	1	0	*FightingAgainstHeatRise*
1	1	0	1	1	*Hyperthermia*
1	1	1	0	0	*FightingAgainstHeatLoss*
1	1	1	0	1	*Hypothermia*
1	1	1	1	0	*none*
1	1	1	1	1	*none*

6.2.6 Behavior: E

It is observed that the monitored variables are dependent on both the behavior of the body, but also the environment in which the body is located. To avoid having to deal with complex formulas such as hope of survival in the water [17], we will consider that the only variable that will change will be the body temperature.

Body temperature evolves over time, but as mentioned above, it is considered as a variable that depends from external parameters of the system. We can not predict its

behavior because it depends on too many external variables (the person's behavior, climate outside the body, person immersed in water, etc.).

Actually E is similar to the table obtained in the behavior section. The variable is body temperature (BT)

Current state (internal)	Event and/or condition	New state
$Hyperthermia$	$BT > 41.5°C$	$ApparentDeath$
$Hyperthermia$	$BT \leq 38.3°C$	$FightingAgainstHeatRaise$
$FightingAgainstHeatRaise$	$BT > 38.3°C$	$Hyperthermia$
$FightingAgainstHeatRaise$	$BT \leq 37.5°C$	$Normal$
$Normal$	$BT > 37.5°C$	$FightingAgainstHeatRaise$
$Normal$	$BT \leq 36.5°C$	$FightingAgainstHeatLoss$
$FightingAgainstHeatLoss$	$BT > 36.5°C$	$Normal$
$FightingAgainstHeatLoss$	$BT \leq 35.0°C$	$Hypothermia$
$Hypothermia$	$BT > 35.0°C$	$FightingAgainstHeatLoss$
$Hypothermia$	$BT \leq 30.0°C$	$ApparentDeath$

6.3 A web server

The third example is an IT example with which we interact every day: a web server.

6.3.1 System description

This is a web server of type *Apache + Php + MySQL*. We will not detail this description. Indeed, it is quite simple and quite long. It is supposed to be known by anyone with a basic knowledge in computer science.

6.3.2 Behavior indetification

There are three elements:

- Web server have received a URL or not.
- Web server can issue a query to the database or not.
- Web server can respond by sending HTML code.

Received	Data querying	HTML sending	possible?	name
no	*no*	*no*	V	**Listening**
no	*no*	*yes*	X	
no	*yes*	*no*	X	
no	*yes*	*yes*	X	
yes	*no*	*no*	V	**Loading And Interpreting Code**
yes	*no*	*yes*	V	**Sending Response**
yes	*yes*	*no*	V	**Waiting For Data**
yes	*yes*	*yes*	X	

6.3.3 State diagram D

$S =$

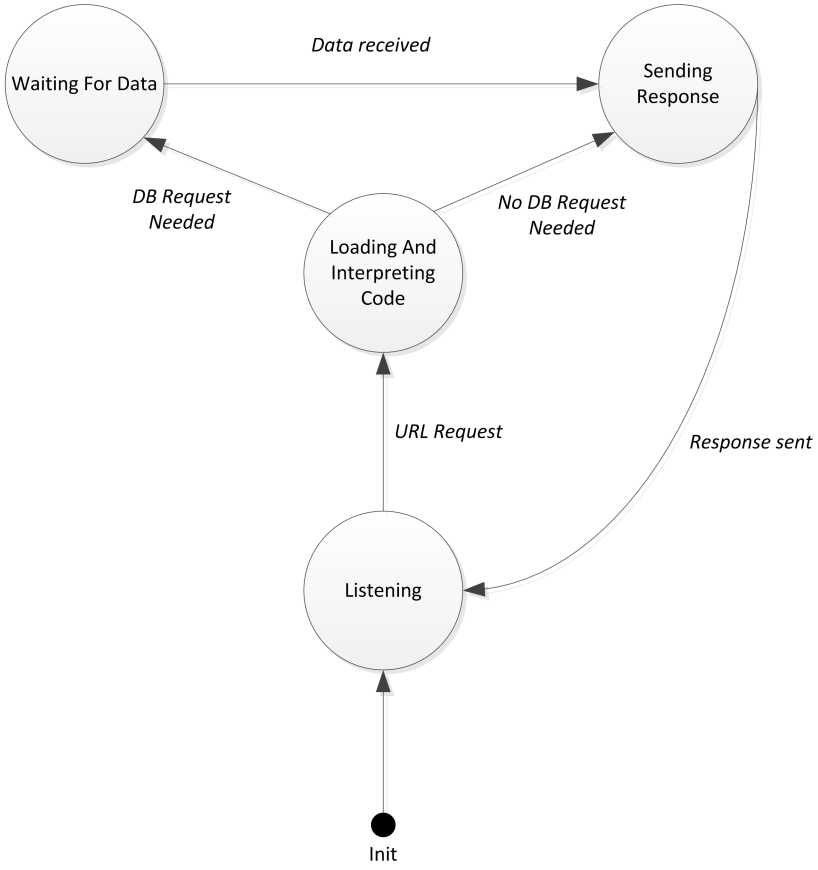


Figure 6.12: Global State Diagram of a webserver

$D = (S, Init = Listening, Death = none)$

6.3.4 Building graph G

Here is our graph G:

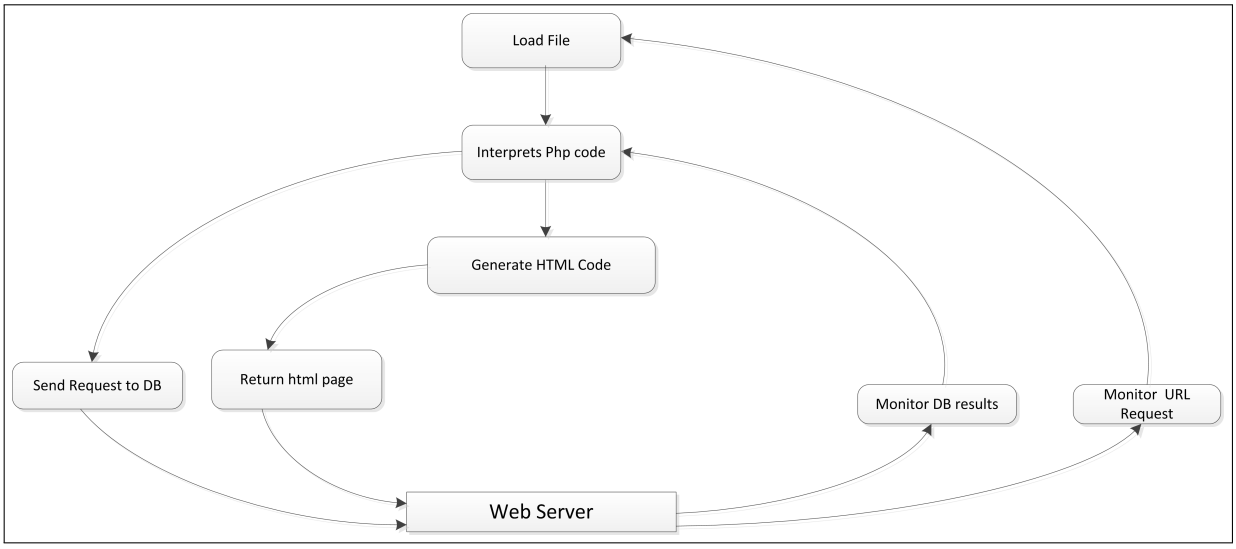
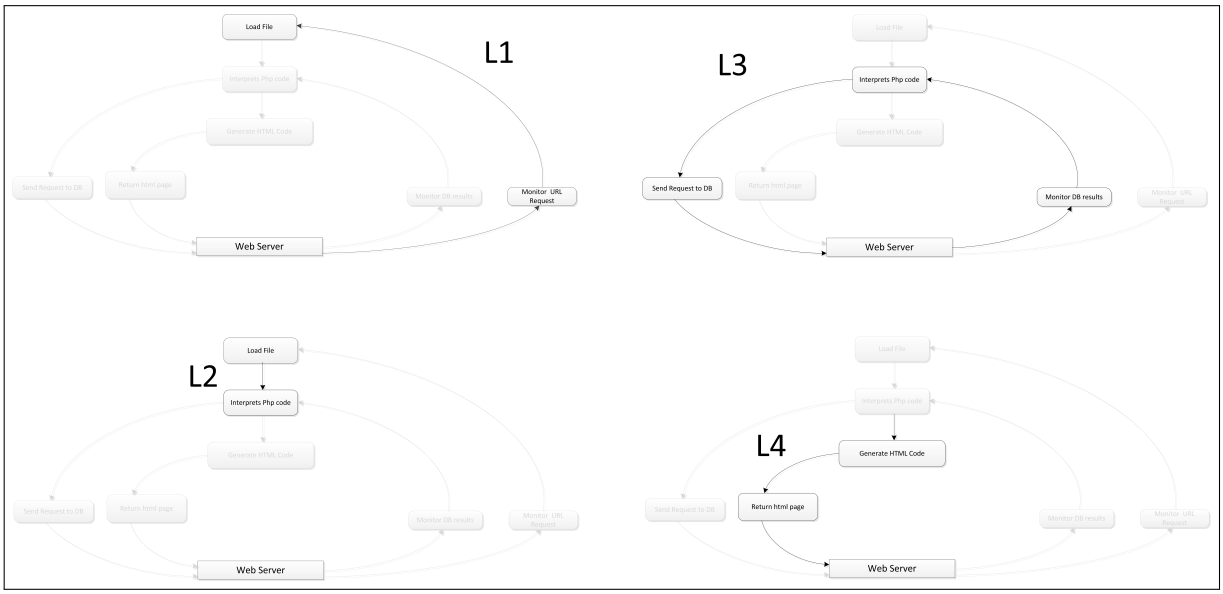


Figure 6.13: Graph of a typical *Apache-Php-MySQL* web server

6.3.5 Finding A

We first identify the different loops from the graph G:



6.3.6 Behavior: E

Current state (internal)	Event and/or condition	New state
Listening	*e1: Valid URL request*	*LoadingAndInterpretingCode*
LoadingAndInterpretingCode	*No Db request needed*	*SendingResponse*
LoadingAndInterpretingCode	*Db request needed*	*WaitingForData*
WaitingForData	*e2: Data received*	*SendingResponse*
SendingResponse	*Response Sent*	*Listening*

Chapter 7

Conclusion

Contents

This end chapter is divided in two parts. Firstly we will enumerate the contributions of this master thesis by establishing a brief review. Secondly we will launch new tracks to go further this work and to start new thesis. And finally, I will close this thesis with my own views and experiences during this work.

7.1 Contributions

Here Thompson's quote (see beginning of this work) is very relevant : *This book of mine has little need of preface, for indeed it is "all preface" from beginning to end.* [45]. Indeed, this master thesis has just made the first steps toward a methodology for WIFS.

We have starting this thesis by recalling some systems notions and the importance of modeling non-linear systems due to their presence if we represents our world in term of dynamical systems. We have then presenting Weakly Interacting Feedback Structures from [35] and introducing a formalism to represent qualitatively these feedback structures. One of the main property of these feedback structure is that their are **adaptives** given their environment. We have then proposed a first methodology, based on design rules, to build WIFS from a system description. This is the continuity of Alexandre Bultot's work [7]. After that, we have bond some existing quantitative methodologies and tools (System Dynamics, Control Theory and Model Checking) and our qualitative methodology and we have seen that if we use them together, it can empowering the **robustness** of designed system. Finally we have applied our methodology to concrete examples from different fields: biology, IT,...

Actually, the whole thesis is a big multidisciplinar introduction to a new way of modeling systems. Indeed, in addition to a new methodology, this thesis includes System Dynamics, Control Theory and Model Checking. I strongly think that it is important to make the links between these disciplines and use them together to design huge complex sytems. It is the key of adaptivity and robustness.

For me, one of the most important contributions is the introduction of WIFS formalism. Note that this is a *proposed* formalism and this is only the basis. We are open to any contribution or modification.

7.2 Further Work

There are a lot of new tracks to go further:

In our methodology, at the interraction level, we have mostly developed the interractions *inside* the feedback structure, between feedback loops. There is a lack of details and development about interractions between the feedback structures themselves, at the *orchestration* level.

In a general point of view, one big thing is to deepen the WIFS methodology. It means to go further in our design rules to guarantee more robust system and justify it. By this way, it avoids unexpected behavior of the system designed. Actually we are now able to build WIFS system qualitatively but not yet quantitatively. This can be considered as one of the most important tasks for software development. A good resource to start to work is the Strogatz book *Nonlinear Dynamics And Chaos* [43].

Despite taking huge time, applying our proposed methodology to design a big complex system can show the power and/or the limits of our methodology. As it takes a lot of time, I've decided to focus my *Rules Application* chapter on three smaller cases instead a bigger one.

It can be interesting to stress the designed sytems to test their robustness and by the way the quality of the methodology.

As previously said, if we can find a good methodology to split and part *System Dynamics* 's diagrams, we will be able to match each part of the partitioned diagram to Feedback Structures graphs which can interract weakly between themselves (so called *Weakly Interacting*)

We can build a verify tool that will be able to check the validity of a WIFS by applying Model Checking in the differents feedback structures.

And finally, the ultimate idea is to create a framework or a tool to build software with WIFS and facilitate the design process.

7.3 Personal View

When Peter Van Roy suggested me this subject, I had no idea of what it was but I had the feeling that WIFS methodology could be a great challenge and very interesting. In my opinion, it exceeds my expectations: it was more interesting that I thought (!) but the challenge was also harder that I thought, sometimes I was desperate because my researchs didn't predict expected outcomes. In the end, I keep a great feeling of this experience, very interesting!

I think this topic is still relatively unexplored but also that there are real opportunities in research and business, especially in terms of *autonomic computing*. Imagine a company that sell softwares that can do the work of maintenance that a real person do and this without making mistakes. That would reduce significantly the maintenance costs of any company.

At the level of thesis development, the fact that Peter Van Roy tells me to start from a few limited resources helps me a lot to understand *gradually* how a qualitative methodology like WIFS works without being influenced by other existing methodologies. It was a very good approach.

Appendix A

Scalaris

Here is the work that has been done regarding the Scalaris system. We decided to include it because this thesis refers it a lot and it simplify the reading. This extract is from the article *Software design with interacting feedback structures and its application to large-scale distributed systems* written by Alexander Reinefeld, Peter Van Roy and Seif Haridi. [34]

Scalaris is an open-source library providing a self-managing data management service for Web 2.0 applications [42] [11] [46]. Web 2.0 enabled the Internet commerce revolution. It is no longer a convenience, but a necessity. Customers rely on its continuous availability, regardless of time and space. Even the shortest interruption, caused by system downtime or network partitioning, may cause huge losses in reputation and revenue. In addition to 24/7 availability, providers face another challenge: they must, for a good user experience, be able to respond within milliseconds to incoming requests, regardless whether thousands or millions of concurrent requests are currently being served. Continuous availability, high performance, and scalability were key requirements in the design of Scalaris. To satisfy these requirements, we designed Scalaris to be self managing.

As a challenging benchmark for Scalaris, Figure A.1 shows how we implemented the core of Wikipedia, the *free encyclopedia, that anyone can edit.* Wikipedia is among the ten most frequently accessed websites. It handles about 50,000 requests per second, of which 48,000 are cache hits in the proxy server layer and 2,000 are processed by ten servers in the master/slave MySQL database layer[48].The proxy and web server layers are embarrassingly parallel and therefore trivial to scale. From a scalability point of view, only the database layer is challenging. Because our implementation uses Scalaris to replace the database layer, it inherits all the favorable properties of Scalaris such as scalability and self management. Instead of using a relational database, we map the Wikipedia content to the Scalaris key/value store. On a page update, a transaction across all affected keys (content, backlinks, categories, etc.) and their replicas is done. With a synthetic benchmark, Scalaris achieves 14,000

read+write transactions per second on 15 servers, increasing almost linearly with the number of servers [11]. This number cannot be directly compared to the Wikipedia number since the work and the processors are not the same, but it does show that Scalaris is a credible implementation.

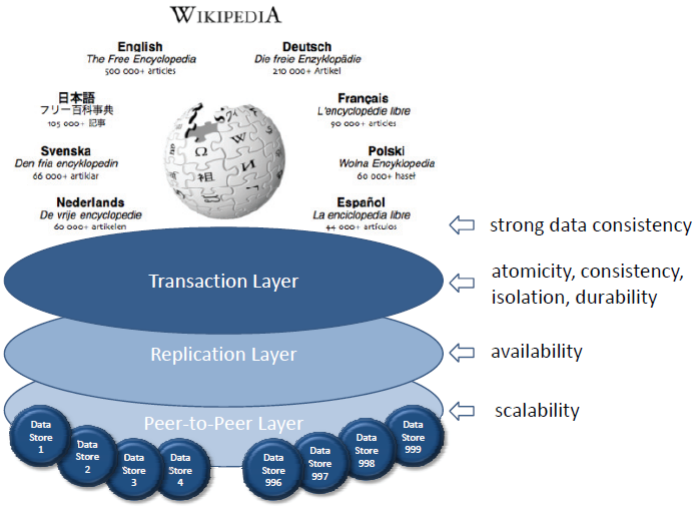


Figure A.1: Distributed Wikipedia built on top of Scalaris

We have built a second library, Beernet that differs from Scalaris in some important points. Whereas Scalaris is based on a Chord # overlay network, Beernet uses a relaxed ring structure [28]. We relax the connectivity condition, requiring only that a node be in the same ring as its successor (instead of both its successor and predecessor). Ring maintenance then does not need periodic stabilization and does not rely on transitive connectivity. The relaxed ring has a *bushy* structure that converges with local operations to a perfectly connected ring. We also modify the transaction manager to request locks quickly and to notify all nodes of modified state. We need these modifications for our collaborative drawing application, DeTransDraw, which uses transactions to overcome network delays while maintaining a coherent global drawing.

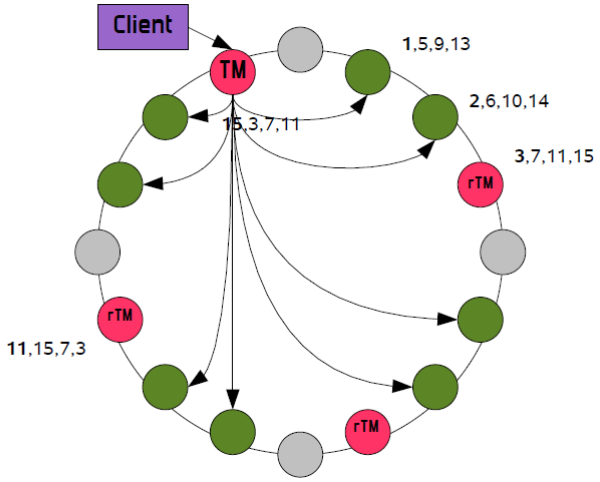


Figure A.2: The Scalaris transaction protocol

A.1 Transactions on an overlay network

We first present the architecture of Scalaris using a traditional layered approach. Scalaris is a structured overlay network extended with a transaction layer using a replicated key/value storage. Its architecture provides the traditional ACID properties of transactions in a scalable decentralized setting. It does not attempt to replace current database management systems with their general, full-edged SQL interfaces. Instead our target is to support transactional Web 2.0 services like those needed for Internet shopping, banking, or multiplayer online games. Figure A.1 shows the three layers of the system:

1. At the bottom, an enhanced structured peer-to-peer network, with logarithmic routing performance, provides the basis for storing and retrieving keys and their corresponding values. In contrast to many other overlays, our implementation stores the keys in lexicographical order. Lexicographical ordering instead of random hashing enables control of data placement which is necessary for low latency access in multi data center environments.

2. The middle layer implements data replication. It enhances the availability of data even under harsh conditions such as node crashes and physical network failures. We use symmetric replication, in which the data is replicated symmetrically around the ring.

3. The top layer provides transactional support for strong data consistency in the face of concurrent data operations. It uses a fast consensus protocol with low communication overhead that has been optimally embedded into the peer-to-peer network.

We explain briefly how the transaction protocol works, since this will help when we explain Scalaris with feedback structures. Figure A.2 shows an example with a structured peer-topeer network that has 16 nodes. A client initiates a transaction by asking its nearest node, which becomes a transaction manager. Other nodes that store data are transaction participants. Given symmetric replication with degree f (4 in the figure), we have f transaction managers (TM and rTM in the figure) and f replicas for the other participating nodes. A modified version of Lamport's Paxos uniform consensus algorithm is used for node agreement [11] [24] [12]: each replicated transaction manager (rTM) collects votes from a majority of participants and locally decides on abort or commit. The transaction manager (TM) then collects a majority from the replicated transaction managers and sends its decision to all participants. This algorithm achieves commitment if more than $f = 2$ nodes of each replica group are alive. The algorithm's operation seems simple; things are actually more subtle because it is correct even if nodes can at any time be falsely suspected of having failed. All we know is that after some unknown finite time, the failure suspicions are correct (eventually perfect failure detection). In our experience, this failure detector is adequate for an Internet setting, where nodes may crash and communication may be interrupted.

$$S_{Scalaris} = S_{key\text{-}value} \wedge S_{connectivity} \wedge S_{routing} \wedge S_{load} \wedge S_{replica} \wedge S_{transaction}$$

Scalaris specification is a conjunction of system properties.
Each non-functional property is implemented by a feedback structure.

$$S_{connectivity} \rightarrow S_{routing} \qquad S_{routing} \rightarrow S_{load}$$
$$S_{connectivity} \rightarrow S_{replica} \qquad S_{replica} \rightarrow S_{transaction}$$
$$S_{routing} \rightarrow S_{replica}$$

Dependencies between the feedback structures (except for covert stigmergy)

Figure A.3: Scalaris specification with feedback structures

A.2 Feedback structures in Scalaris

As a complement to the layered presentation of the previous section, we can present the architecture of Scalaris as a set of five feedback structures and their interactions:

1. *Connectivity management.* This feedback structure maintains the connectivity of the ring using periodic successor list stabilization.
2. *Routing management.* This feedback structure maintains efficient routing tables using periodic finger stabilization.
3. *Load balancing.* This feedback structure balances load by monitoring each node and moving nodes when necessary to distribute load evenly.

4. *Replica management.* This feedback structure maintains the invariant that there will always eventually be f replicas of each data item. Whenever there is a potential new replica, it uses consensus to propose a new replica set.

5. *Transaction management.* This feedback structure uses consensus among replicated transaction managers and storage nodes to perform atomic commit, as explained in the previous section. If the transaction manager TM fails, then one of its replicas rTM takes over. Multiple takeovers are tolerated by consensus.

The Scalaris specification then consists of the conjunction of the five properties implemented by these feedback structures, together with the functionality of the key-value store itself. Interactions between the feedback structures are possible when the perceived set of correct nodes changes, due to nodes joining, leaving, failing, or suspected of failing. This gives a dependency graph between feedback structures. Figure A.3 shows the form of the specification and the dependency graph. [...] For Scalaris we handle the interactions as follows:

- Connectivity management, routing management, and replica management interact when the set of nodes changes. This does not affect correctness because each manager always converges towards its ideal solution. Oscillations do not occur because there are no cyclic dependencies (connectivity management is not affected by the other two). We choose the time delays of the different managers to improve effciency.

- Routing management can influence the load balancing. This has an effect on the effciency of the load balancing algorithm.

- Replica management can influence the transaction management because the number of replicas can change temporarily. This can cause data inconsistency if there are temporarily more than f replicas, which can occur if there is a false failure suspicion. This is trickly to handle correctly. One solution is to require more than a simple majority in the consensus algorithm of the transaction management. This reduces the probability of inconsistency. This is not satisfactory because it does not eliminate the problem and because it reduces overall performance just to handle a rare situation. Another solution, which we are working on, is to use consensus in the replica management itself to ensure that all nodes agree on the f replicas. The transaction manager then takes a majority only from an agreed set of replicas.

- Covert stigmergy between feedback structures may occur because the network is a shared resource. Connectivity management is the most important property and so it must be done faster than the other managers. Otherwise the overlay network may become disconnected at high loads. To minimize other bad effects due to stigmergy, the management load on the network should be kept as constant as possible. If connectivity management does less work, then routing management takes up the slack.

Because these five feedback structures act at all layers of the system, we can say that the Scalaris implementation is self managing *in depth*. This has important consequences for system administration. For many Web 2.0 services, the total cost-of-ownership is dominated by the costs needed for personnel to maintain and optimize the service. In traditional database systems, changing system size and tuning require human interference which is error prone and costly. In both these situations, the same number of administrators in Scalaris can operate much larger installations.

Bibliography

[1] Per Brand Ahmad Al-Shishtawy, Vladimir Vlassov and Seif Haridi. *A Design Methodology for Self-Management in Distributed Environments.* Royal Institute of Technologyand Swedish Institute of Computer Science, Stockholm, Sweden, 2009.

[2] Ahmad Al-Shishtawy. *Enabling and Achieving Self-Management for Large Scale Distributed Systems.* "KTH - Royal Institute of Technology, Stockholm, Sweden, 2010.

[3] K. Pyragas V. Pyragas H. Benner. *Delayed feedback control of dynamical systems at a subcritical Hopf bifurcation.* Semiconductor Physics Institute, 2004.

[4] H. K. D. H. Bhadeshia. *Neural Networks and Information in Materials Science.* Wiley InterScience, 2008.

[5] Olivier Bonaventure. *Computer Networking : Principles, Protocols and Practice.* Addison Wesley, 2010.

[6] Holford T. et Fish D. Brownstein, J.S. *A climate-based model predicts the spatial distribution of the Lyme disease vector Ixodes scapularis in the United States.* Environmental Health Perspectives 111(9), 2003.

[7] Alexandre Bultot. *A survey of systems with multiple interacting feedback loops and their application to programming.* Memoire, 2009.

[8] Christos G. Cassandras and Stephane Lafortune. *Introduction to Discrete Event Systems.* 2008.

[9] Jack Cohen. *The Collapse of Chaos: Discovering Simplicity in a Complex World.* Perguin, 2000.

[10] D. A. Peled E. M. Clarke, O. Grumberg. *Model Checking.* MIT Press, 1999.

[11] Seif Haridi Florian Schintke, Alexander Reinefeld and Thorsten SchÃijtt. *Enhanced Paxos Commit for Transactions on DHTs.* 10th IEEE/ACM International Symposium on Cluster, 2010.

[12] Rachid Guerraoui and Luis Rodrigues. *Introduction to Reliable Distributed Programming.* Springer-Verlag, 2006.

[13] Bruce Hanon and Matthias Ruth. *Dynamic Modelling of Diseases and Pests.* Humana Press, 2009.

[14] Konstantin Popov Joel Hoglund, Per Brand Ahmad Al-Shishtawy, and Vladimir Vlassov Nikos Parlavantzas. *Design of a self-application using P2P based management infrastructure.*

[15] IBM. *An architectural blueprint for autonomic computing, 4th edition.* 2006.

[16] Absolute Motion INSTITUTE. *The Joules of the Universe.*

[17] SociÃl'rÃl' internationale de sauvetage du LÃl'man. *La survie en eau froide.* Les dossiers techniques de la SISL, 2006.

[18] Allen Tannenbaum John Doyle, Bruce Francis. *Feedback Control Theory.* Macmillan Publishing Co., 1990.

[19] Sujay Parekh Joseph L. Hellerstein, Yixin Diao and Dawn M. Tilbury. *Feedback Control of Computing Systems.* 2004.

[20] JPFWiki. *Testing vs. Model Checking.* NASA Official: Sonie Lau, 2009.

[21] J. O. Kephart and D. M. Chess. *The Vision of Autonomic Computing.* 2003.

[22] S. El-Ansary E. Aurell Krishnamurthy, S. and S. Haridi. *A statistical theory of Chord under churn.* 2005.

[23] Supriya Krishnamurthy and John Ardelius. *An Analytical Framework for the Performance Evaluation of Proximity-Aware Overlay Networks.* Tech. Report TR-2008-01, Swedish Institute of Computer Science, 2008.

[24] Leslie Lamport. *The part-time parliament.* ACM Trans. Comput. Syst. 16(2), 1998.

[25] Stephane Molotchnikoff Lauralee Sherwood and Alain Lockhart. *Physiologie humaine.* De Boeck, 2006.

[26] Boris J. Lurie and Paul J. Enright. *Classical Feedback Control with MATLAB.* Marcel Dekker Inc., 2000.

[27] Mario Salvadori Mathys Levy. *Why Buildings Fall Down: How Structures Fail.* W. W. Norton & Company, 1994.

[28] Boris Mejias and Peter Van Roy. *The Relaxed-Ring: A Fault-Tolerant Topology for Structured Overlay Networks.* Parallel Processing Letters 18(3), 2008.

[29] Grafman J.-Pietrini P. Alway D. Carton J. C. Miletich R. Nichelli, P. *Brain activity during chess playing.* Nature, 1994.

[30] Wiener Norbert. *Cybernetics, or Control and Communication in the Animal and the Machine.* MIT Press, 1948.

[31] European Sixth Framework Programme. *Self Management for Large-Scale Distributed Systems based on Structured Overlay Networks and Components*. The Adventures of Selfman - Year Three, 2009.

[32] SELFMAN project final review. *WP 5 Application Evaluation*. 2009.

[33] Randolph. *General Framework for Comparative Quantitative Studies on Transmission of Tick-Borne Diseases Using Lyme Borreliosis in Europe as an Example*. J. Med. Entom, 1995.

[34] Peter Van Roy Seif Harid Alexander Reinefeld. *Software design with interacting feedback structures and its application to large-scale distributed systems*. 2010.

[35] Peter Van Roy Seif Harid Alexander Reinefeld. *Designing Robust and Adaptive Distributed Systems with Weakly Interacting Feedback Structures*. January 2011.

[36] Peter Van Roy. *SELFMAN Project. D5.7: Guidelines for bulding self-managing applications.*

[37] Peter Van Roy. *Self Management and the Future of Software Design*. 2006.

[38] Peter Van Roy. *Presentation: Scale and Design for Peer-to-Peer and Cloud*. February 2012.

[39] Wilson J. Rugh. *Nonlinear System Theory: The Volterra/Wiener Approach*. The Johns Hopkins University Press, 1981.

[40] Peter M. Senge. *The Fifth Discipline, the art and practise of the learning organization*. Currency Doubleday, 1990.

[41] Michael Sipser. *Introduction to the Theory of Computation*. 1997.

[42] Alexander Reinefeld Stefan Plantikow and Florian Schintke. *Transactions for Distributed Wikis on Structured Overlays*. DSOM, Springer LNCS, 2007.

[43] Steven Henry Strogatz. *Nonlinear Dynamics And Chaos: With Applications To Physics, Biology, Chemistry, And Engineering*. Westview Press, 1994.

[44] Steven Henry Strogatz. *Sync: The Emerging Science of Spontaneous Order*. Hyperion, 2003.

[45] D'Arcy Wentworth Thompson. *On Growth And Form*. Cambridge University Press, 1917.

[46] Alexander Reinefeld Monika Moser Christian Hennig Thorsten SchÃijtt, Florian Schintke and Nico Kruber. *Scalaris: A Scalable Transactional Data Store for Web 2.0 Services*. Zuse Institute Berlin, 2008.

[47] Axel van Lamsweerde. *Software Engineering: Development Methods : course slides*.

[48] the free encyclopedia Wikipedia. *Entry wikipedia*. 2009.

[49] the free encyclopedia Wikipedia. *Hyperthermia*. 2013.

[50] the free encyclopedia Wikipedia. *Hypothermia*. 2013.

[51] the free encyclopedia Wikipedia. *Thermoregulation*. 2013.

[52] Michel Jobin William-F Ganong. *Physiologie medicale*. De Boeck (2e edition), 2005.

[53] P. Wolper. *An Introduction to Model Checking*. 1995.

[54] Peter Van Roy Ali Ghodsi Seif Haridi Jean-Bernard Stefani Thierry Coupaye Alexander Reinefeld Ehrhard Winter Roland Yap. *CoreGRID Technical Report Number TR-0018 : Self Management of Large-Scale Distributed Systems by Combining Peer-to-Peer Networks and Components*. Institute on System Architecture, 2005.

[55] Peter Van Roy Seif Haridi Jean-Bernard Stefani Thierry Coupaye Alexander Reinefeld Roland Yap. *Self Management for Large-Scale Distributed Systems: An Overview of the SELFMAN Project*.

[56] W.T. Freeman Yedidia, J.S. and Y. Weiss. *Understanding Belief Propagation and Its Generalizations*. 2003.