

Université Catholique de Louvain
École Polytechnique de Louvain
Département d'Ingénierie Informatique



Mémoire

A survey of systems with multiple interacting feedback loops and their application to programming

Promoteur : Peter Van Roy

Lecteurs : Seif Haridi
Boriss Mejias

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques (120 crédits)
option Software engineering and programming systems
par
Alexandre Bultot

Louvain-la-Neuve
2008-2009

"Loving a baby is a circular business, a kind of feedback loop. The more you give the more you get and the more you get the more you feel like giving."

Dr. Penelope Leach

"A president on a feedback loop"

George W. Bush

Remerciements

Je tiens tout d'abord à remercier mon promoteur, Peter Van Roy, pour m'avoir guidé tout au long de la création de ce mémoire, pour ses remarques pertinentes qui m'ont permis de progresser et pour son envie qu'il m'a communiquée pour mener à bien ce projet.

I would firstly like to thank my supervisor, Peter Van Roy, for his guidance throughout the redaction of this dissertation, for his constructive comments and suggestions that helped me to move forward and for passing on to me his passion for this project.

Mes remerciements vont également à Boris Mejias et Seif Haridi pour la gentillesse et la patience d'avoir lu et commenté mon travail.

I would also like to thank Boris Mejias and Seif Haridi for their kindness and their patience whilst reading and commenting on my work.

Un tout grand merci à Yves Jaradin pour son aide très utile pour la conception du simulateur et pour l'interprétation des résultats.

A very big thank you goes to Yves Jaradin for his great help during the development of the simulator and the interpreting of the results.

Merci à ma Maman qui m'a permis de faire les études qui me tenaient à coeur, qui m'a donné la possibilité de réaliser tous mes projets, et qui m'a soutenu depuis que Papa nous a quitté il y a 20 ans.

Un gros bisou à Felicity pour m'avoir aimé, supporté et aidé moralement pendant déjà 6 ans et encore pour tant d'années à venir. Mais également pour m'avoir aidé à corriger mes fautes d'anglais.

Je tenais aussi remercier tous les SINF 22 et les autres SINF que j'ai connu pendant ces cinq dernières années pour tous les bons moments passés ensemble comme les pizza buffet, les Galilées, les Zanzibars du projet de génie logiciel et pour les rares mais intenses soirées SINF.

Contents

Remerciements

iii

1	Introduction	1
1.1	The SELFMAN project	2
1.2	Self-management	2
1.2.1	Self-configuration	2
1.2.2	Self-healing	3
1.2.3	Self-protection	4
1.2.4	Self-optimisation	4
1.3	Automatic elements	4
1.3.1	Definition	5
1.3.2	Coordinating multiple control loops	6
1.3.2.1	Stigmergy	6
1.3.2.2	Peer-to-Peer Management Interaction	6
1.3.2.3	Hierarchical Management	7
1.4	What's next?	8
2	System dynamics and systemic feedback	11
2.1	System dynamics	11
2.2	Systemic feedback or the system dynamics study	13
2.2.1	Importance of causal relations	13
2.2.2	Importance of circular causality (feedback causation) over time	13
2.2.3	Dynamic behaviour pattern orientation	14
2.2.4	Internal structure as the main cause of dynamic behaviour	14
2.2.5	Systems perspective	14
2.3	Feedback representation	15
2.3.1	Causal loop diagram	15
2.3.1.1	Causality	15
2.3.1.2	Positive and negative causal effects and feedback loops	16
2.3.1.3	Example	17
2.3.2	Stock-and-flow diagram	18
2.3.2.1	Diagram elements	19
2.3.2.2	Diagram construction	19
2.3.2.3	Example	20
2.3.3	Monotonic feedback loop	21
2.3.3.1	Hill equations	22

2.3.3.2	Example	22
2.4	Modelling methodology	23
2.4.1	Problem identification	24
2.4.2	Model conceptualisation and dynamic hypothesis	25
2.4.3	Formal model construction	25
2.4.4	Estimation of the parameter values	25
2.4.5	Model credibility testing	26
2.4.6	Sensitivity analyses	27
2.4.7	Impact of policies testing	27
2.4.8	Model validation	27
2.4.8.1	Validity or usefulness?	28
2.4.8.2	Concrete tests to build confidence	28
3	Feedback systems - building blocks	33
3.1	Single loop systems	34
3.1.1	Feedback loop	34
3.1.2	Reactive loop versus proactive loop	34
3.1.3	Positive feedback loop	35
3.1.4	Negative feedback loop	37
3.1.5	Time delays	40
3.1.6	Oscillation	40
3.2	Two-loop systems	41
3.2.1	Stigmergy	41
3.2.2	Management	42
3.2.3	Coupling two positive feedback loops	42
3.2.4	Coupling two negative feedback loops	43
3.2.5	Coupling a linear positive feedback loop with a linear negative feed- back loop	43
3.2.6	Loop dominance shift	44
3.2.7	Coupling fast and slow positive feedback loops	46
3.2.8	Complex components should be sandboxed	46
3.2.9	Use push-pull to improve regulation	47
3.2.10	Reversible phase transitions	47
3.2.11	$1 + 1 = 3?$	47
3.3	Example with several basic patterns	48
3.4	Decomposition of big systems	49
3.4.1	Reduction method to lower high-dimensional ODE	50
3.4.2	Strongly connected components	51
4	Feedback systems - global behaviour patterns	53
4.1	Success to the Successful	54
4.2	Limits to Growth	55
4.2.1	The S-shaped growth	55
4.2.2	Overshoot-and-decline	59
4.3	Tragedy of the Commons	64
4.4	The Attractiveness Principle	65
4.5	Growth & Under Investment	66

4.6	Balancing with Delay	67
4.7	Escalation	68
4.8	Indecision	70
4.9	Fixes That Backfire	71
4.10	Accidental Adversaries	72
4.11	Shifting the Burden	74
4.12	Addiction	75
4.13	Drifting Goals	75
4.14	Growth & Under Investment with a Drifting Standard	77
4.15	The Archetype Family Tree	78
5	Simulator - definition	81
5.1	Definition of agents	82
5.1.1	Base agent	82
5.1.2	Generic agents	84
5.1.3	Example	85
5.2	Definition of the graph	87
5.2.1	Agents	87
5.2.2	Graph	87
5.3	Simulator engines	88
5.3.1	Synchronous engine	88
5.3.2	Asynchronous engine	92
5.4	Graphical interface	95
5.4.1	Main panel	95
5.4.2	Graph representation	97
5.4.3	Chart representation	97
6	Simulator - results	99
6.1	Delta time	100
6.2	Results	100
6.2.1	Population	101
6.2.1.1	Synchronous engine	101
6.2.1.2	Asynchronous engine	103
6.2.2	Drifting goals	105
6.2.2.1	Synchronous engine	105
6.2.2.2	Asynchronous engine	107
6.2.3	Epidemic	108
6.2.3.1	Synchronous engine	109
6.2.3.2	Asynchronous engine	110
6.2.4	Crowding	112
6.2.4.1	Synchronous engine	112
6.2.4.2	Asynchronous engine	113
6.2.5	Flowers	115
6.2.5.1	Synchronous engine	115
6.2.5.2	Asynchronous engine	116
6.2.6	Balancing loop	118
6.2.6.1	Synchronous engine	119

6.2.6.2	Asynchronous engine	119
6.2.7	Knowledge diffusion	120
6.2.7.1	Synchronous engine	121
6.2.7.2	Asynchronous engine	122
6.2.8	Escalation	124
6.2.8.1	Synchronous engine	124
6.2.8.2	Asynchronous engine	125
6.2.9	Success to the successful	127
6.2.9.1	Synchronous engine	127
6.2.9.2	Asynchronous engine	128
7	Conclusion	131
7.1	Modelling a feedback system for software applications	132
7.2	Feedback structures and design rules	134
7.3	Further work	135
	Bibliography	137
	Appendix A – Pattern examples	141
	Appendix B – Other examples	195
	Appendix C – Simulator.oz	219
	Appendix D – Simulation results	225

In memory of my Dad.

Chapter 1

Introduction

The idea of *automatic computing* is not new. In 2001, an IBM manifesto described “a looming software complexity crisis” as the “main obstacle to further progress in the IT industry”. [1]

“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.” [2]

1.1 The SELFMAN project

The SELFMAN project[3] – Self Management for Large-Scale Distributed Systems based on Structured Overlay Networks and Components – is an IST project launched in June 2006 to build self-managing applications on the internet based on peer-to-peer technology, whose aim is to build internet applications that do not break and that do not need a team of specialists to keep them running. This project uses two technologies: structured overlay networks (SONs) and advanced components models. SONs are “peer-to-peer systems that provide robustness, scalability, communication guarantees, and efficiency”. [4] On the other hand, “component models provide a framework to extend the self-managing properties of SONs over the whole application”. [4]

1.2 Self-management

Self-management is the core of autonomic computing. That is, systems that can manage themselves following some rules defined by an administrator. Four main characteristics or aspects can be used to define self-management (see below). All four self-* characteristics have been approached in the SELFMAN project. In [5], guidelines for building self-managing applications have been drawn up. These guidelines include tips to make applications self-tuning (self-optimising), self-protecting, self-healing, and self-configuring.

1.2.1 Self-configuration

The basic idea is that adding a new component to an existing system can be extremely complex and error-prone. Self-configuration can allow a component to be aware of the existing environment and to adapt itself to it. The system must also become accustomed to this new presence. This incorporation of a new component follows high-level policies that can be related to business objectives, for example.

Figure 1.1, taken from [6], represents the self-configuration control loop used in YASS (see [6] for more details). In this case, YASS has to maintain its total storage capacity and total free space to meet functional requirements (policies). This control loop consists of three elements: a Component Load Watcher (CLW), a Storage Aggregator (SA), and a Storage Manager (SM).

CLW receives information from the sensors about the storage group in order to evaluate the free space available. If a change in this value is greater than a certain delta value, CLW triggers an event understandable by SA. SA also receives information from the sensors, that is,

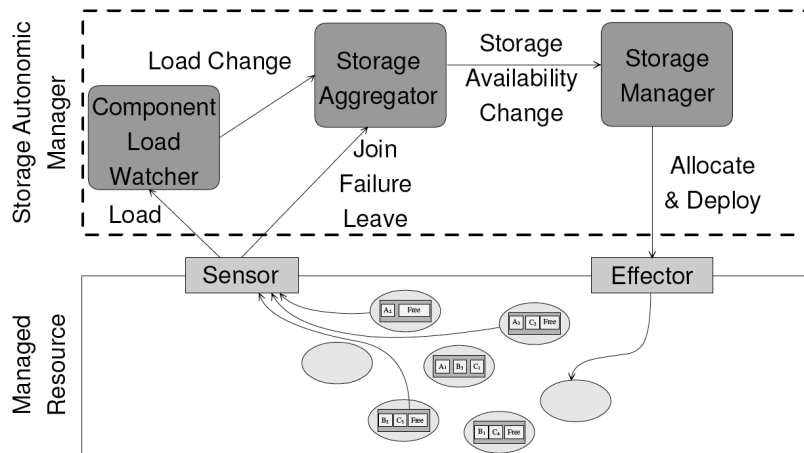


FIGURE 1.1: Self-configuration control loop

'fail', 'leave', and 'join' events. SA also has the responsibility to warn SM if the total capacity or the total free space drops below a predefined threshold. Whenever SM receives an event from SA about the problem, it allocates more resources and deploys storage components on these resources.

Self-configuration is thus an “automated configuration of components and systems that follows high-level policies. The rest of the system adjusts automatically and seamlessly”. [2]

1.2.2 Self-healing

Automatic computing systems use their knowledge to detect, diagnose, and repair problems from bugs or failures. The self-healing characteristic consists thus of agents that monitor log files or data from the system, compare it with their knowledge about problems, and if necessary, apply known solutions to the problem or warn an administrator if no solution exists for this problem.

Figure 1.2, taken from [6], depicts the self-healing configuration used in YASS. This self-management characteristic is used for maintaining a certain degree of replication of files. This self-healing control loop consists of two agents: a File Replica Aggregator (FRA) and a File Replica Manager (FRM).

FRA receives 'fail' or 'leave' events from a file group about its members. FRA then warns FRM about this problem. FRM will then find the corresponding replica of the file that 'fail' or 'leave' and will restore the replica.

Self-healing consists of “systems that automatically detect, diagnose, and repair localized software and hardware problems”. [2]

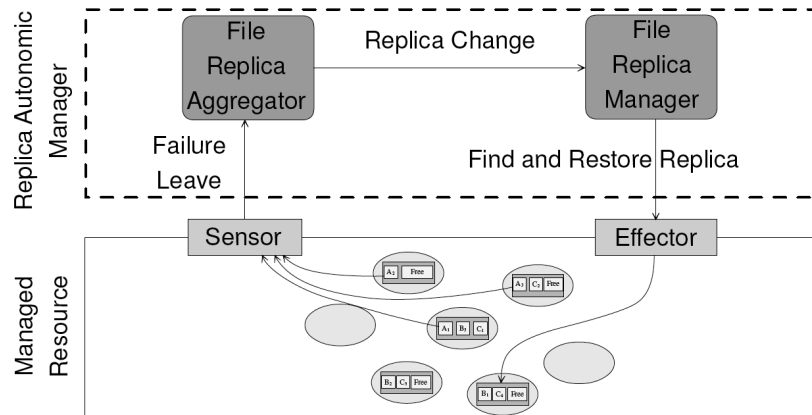


FIGURE 1.2: Self-healing control loop

1.2.3 Self-protection

[2] explains that automatic systems can be self-protective in two possible ways: “they will defend the system as a whole against large-scale, correlated problems arising from malicious attacks or cascading failures that remain uncorrected by self-healing measures. They will also anticipate problems based on early reports from sensors and take steps to avoid or mitigate them”.

1.2.4 Self-optimisation

Self-optimisation is also called self-tuning. This characteristic is only for software where a certain number of parameters exists each of these can have multiple values. The optimal choice for the value of parameters can be very complex and impossible to obtain by human tuning. By consequence, “automatic systems will continually seek ways to improve their operation, identifying and seizing opportunities to make themselves more efficient in performance or cost”. [2]

For example, [6] proposes a self-optimisation control loop used for a File Availability system. That is, popular files should have more replicas in order to increase their availability (see Figure 1.6 in Section 1.3.2.3).

1.3 Automatic elements

Automatic systems consists of *automatic elements* that interact with other automatic elements.

1.3.1 Definition

"[Automatic elements are] individual system constituents that contain resources and deliver services to humans and other automatic elements. Automatic elements will manage their internal behavior and their relationships with other automatic elements in accordance with policies that humans or other elements have established". [2]

Figure 1.3, taken from [2], represents the structure of an automatic element that consists of one or several managed elements coupled with an 'automatic manager'. An automatic element is also called a *control loop*.

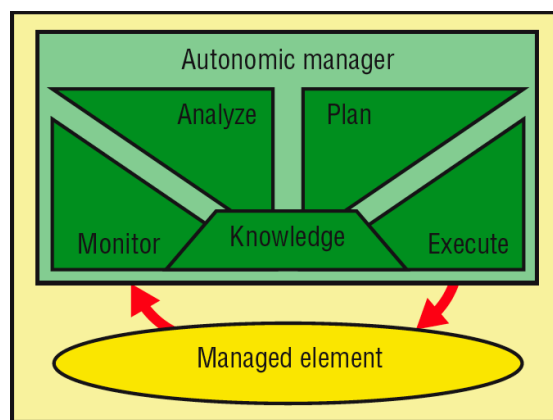


FIGURE 1.3: Structure of an automatic element

The automatic manager, also called the MAPE-loop[7], has five main components (another feedback structure closely related to the MAPE-loop will be defined and used throughout this dissertation: 'feedback loops'):

One that **monitors** the managed element and collects data via aggregation or filtering.

One that **analyses** the collected data regarding management policies.

One that **plans** if needed, a set of actions that can be executed and placed in a reaction plan.

One that **executes** the plan built in the previous component.

Knowledge that can be used to detect problems and that contains solutions to such problems.

1.3.2 Coordinating multiple control loops

An application is composed of multiple control loops that interact with each other. They are not independent due to the fact that they manage the same system. Interaction has three possible forms described in [6] and in the following points.

1.3.2.1 Stigmergy

In this case, both control loops explained in the self-healing and self-configuration characteristics are combined. Stigmergy is a kind of communication between agents. One agent applies a change in the system and this change is perceived by other agents via their sensors. This perception of change by the others implies more change to come.

For example, one can observe in Figure 1.4, taken from [2], that the Storage Manager plans to de-allocate (via a 'leave' command) some resources because the use of storage components drops (i.e. the total capacity is greater than initial requirements and free space is more than a predefined ratio). This action will be perceived by the File Replica Manager that will take further action to move the file components from the leaving resource to other resources.

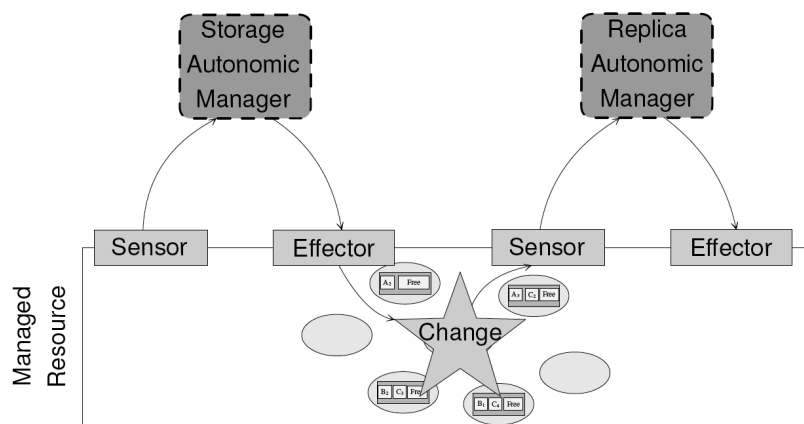


FIGURE 1.4: The stigmergy effect

1.3.2.2 Peer-to-Peer Management Interaction

This configuration is the same as in the previous point. When two control loops manage the same resource, non expected behaviours may occur. These kinds of behaviours may be avoided if a link exists between the two control loops (see Figure 1.5 taken from [2]). "P2P management interaction is a way for managers to cooperate to achieve self-management. It does not mean that a management controls the other".[6]

“For example, when a resource fails, the Storage Manager may detect that more storage is needed and start allocating resources and deploying storage components. Meanwhile the File Replica Manager will be restoring the files that were on the failed resource. It might fail in restoring the files due to space shortage since the Storage Manager did not have time to finish. This may also prevent the user temporary from storing files.

If the File Replica Manager waited for the Storage Manager to finish, this problem could be avoided. [...] Before restoring files, the File Replica Manager informs the Store Manager with the amount of storage it needs to restore files. The Storage Manager checks available storage and informs the File Replica Manager that it can proceed if enough space is available or ask to wait until more storage is allocated”.[6]

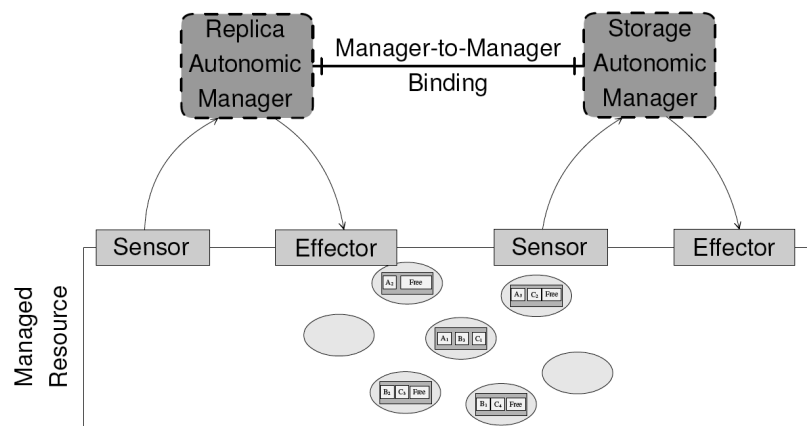


FIGURE 1.5: Peer-to-Peer management interaction

1.3.2.3 Hierarchical Management

In this case, control loops can be embedded in other control loops as a resource. For example, the control loop used for self-optimisation (Figure 1.6 taken from [2]) consists of a File Access Watcher (FAW) and a File Availability Manager (FAM). “FAW monitors the file access frequency. If a file is popular and accessed frequently then it issues a ‘frequency change’ event. The FAM may decide to increase the value of the replication degree parameter in the File Replica Manager that will start storing more replicas and then maintaining the new replication degree”.

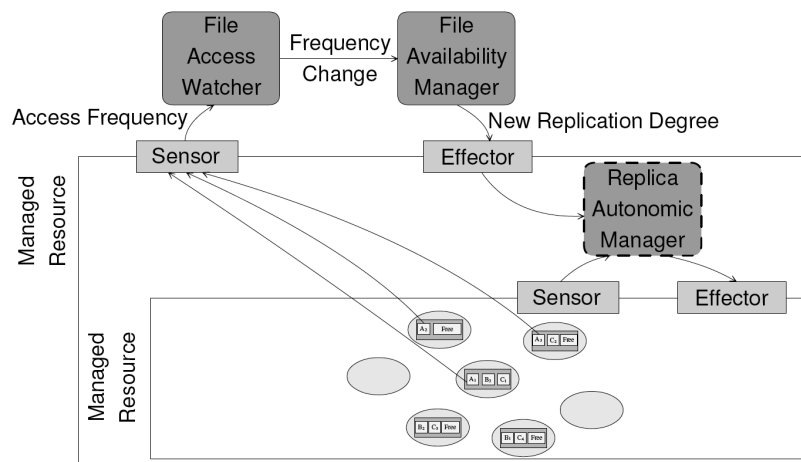


FIGURE 1.6: Hierarchical Management

1.4 What's next?

After this introduction to the state of the art in self-management applications, we need to introduce the content of this dissertation.

"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".[5]

And this is where my dissertation brings its contribution to the self-management of applications. My work consists in analysing feedback structures from widely different disciplines,

such as management, biology, etc. This analysis tries to extract feedback patterns and design rules to help the design of self-managing applications. These patterns and rules may help developers to find solutions to unexpected behaviours and may give tips to designers who are building feedback structures.

The structure of this dissertation is as follows:

- Chapter 2 presents general information about system dynamics and systemic feedback. Applications evolve over time, therefore dynamic behaviours and diagramming representation of feedback structures must be introduced. A modelling methodology is also introduced to help modellers to create models of the system being studied.
- Chapter 3 gives the basis of feedback systems. One-loop and two-loop systems are presented alongside their behaviour. A proposition for the decomposition of large systems is also introduced. This chapter provides building blocks that can be combined to obtain global behaviour patterns presented in Chapter 4.
- Chapter 4 depicts more complex feedback structures that are mostly used in System Thinking to solve problems encountered in organisations. These patterns are explained in detail with structure diagrams and possible behaviours.
- Chapter 5 presents the simulator that was developed to simulate feedback structures presented in Chapters 3 and 4. This simulator uses two different approaches: one based on a synchronous engine and one based on an asynchronous engine.
- Chapter 6 gives some results from simulations produced by the simulator explained in Chapter 5. These results are interesting cases that are described in this chapter.
- Chapter 7 is the conclusion of this dissertation. Several applications of the feedback structures and design rules observed in a wider variety of domains than the field computer sciences domain are given.
- Appendix A presents examples related to the feedback patterns presented in Chapter 4.
- Appendix B gives examples of several feedback structures that are examples of the feedback building blocks presented in Chapter 3 but with more loops that intensify the effect of regulation for example.
- Appendix C lists the code used to build the engines of the simulator defined in Chapter 5.
- Appendix D contains all the simulations carried out by the simulator for different models described in Chapters 3 and 4.

Chapter 2

System dynamics and systemic feedback

2.1 System dynamics

“*System dynamics* is an approach to understanding the behaviour of complex systems over time. It deals with feedback loops and time delays that affect the behaviour of the entire system.” [8]

The definition given above contains important notions such as “complex systems”, “time”, “feedback loop”, “delay”, and “behaviour”. These notions will be explained in the following sections. They are very important and must be understood in order to follow the rest of this dissertation.

Dynamic: change over time

System: “collection of interrelated elements, forming a meaningful whole” [9]

Complex: dynamic systems are complex especially in the case of non-linear feedback systems involving human actors. But the complexity of a dynamic system comes from a list of reasons [9, 10] (see below).

Here is a non exhaustive list of reasons why a dynamic system can be complex:

Dynamics: as opposed to static systems, elements in a dynamic system change over time due to their interaction with other elements. These changes are not always predictable. Changes also occur in different time scales that may interact: for example, a star

evolves over billions of years burning its fuels but it explodes in seconds as a supernova. Finally, time delays between the cause and the effect and between actions and reactions exist.

Tight coupling: “everything is connected to everything else”. Actors in a system interact with other actors and with their environment.

Feedback: due to the tight coupling between actors of a system, actions from an actor influence the world, changing the nature and triggering other actors to react. Thus, multiple feedback loops may interact together and mental simulation of such configuration may be very difficult to predict intuitively.

Non-linearity: the effect of an action on a variable may not be proportional to the action taken. That is, the same action may not have the same effect on a range as it may on another range. For example, comparing the effect of doubling the amount of money spent on the advertising campaign on two products A and B: the effect of this increase may produce an increase of 20% for product A that costs 100 euros but produce an increase of only 5% for product B that costs 200 euros. Non-linearity is very hard to predict intuitively but also mathematically.

History-dependency: some actions may not be reversible and can have dramatic effects on a system.

Self-organisation: “The dynamics of systems arise spontaneously from their internal structure. Often, small, random perturbations are amplified and modelled by internal feedback structures, generating patterns in space and time”. [10]

Scale: non-linearity is not always associated with effect, this notion also co-exists with the notion of scale. The complexity of a system is non-linear with regards to the number of elements in a system.

Human dimension: this dimension makes the system more complex due to the nature of a human being. Unlike physical laws, the behaviour of human beings is not “constant”. There are no laws about how human reacts to a phenomenon. The modeller has to make assumptions about the human behaviour.

Trade-offs: time delays mean that the long-run response is different from the short-run response. For example, “high leverage policies often cause worse-before-better behaviour, while low leverage policies often generate transitory improvement before the problem grows worse”. [10]

Cause and effect separated in time and space: “in non-linear dynamic feedback models with several variables, the cause-effect relations become detached in time and space.

When an action is applied at point A in the model with an expected immediate result at point B, this result may never be obtained and furthermore, some unintended effect may be observed at a distant point C, after some significant time delay.” [9]

Counterintuitive: this separation in time and space is counterintuitive to the normal behaviour of human beings who try to look for causes near the variables that they want to observe.

2.2 Systemic feedback or the system dynamics study

In general, system dynamics studies are carried out to discover and understand the causes of a dynamic problem and to find alternatives in the system to try to eliminate the problem. To do so, modellers must follow a certain philosophy of modelling, analysis and design called *systemic feedback*. [9] This philosophy has a systems theory that combines cybernetics (developed by Nobert Wiener in 1948 [11]) and feedback control theories (developed by Gordon S. Brown in 1948 [12]).

This philosophy contains five principles that are explained in the following sub-sections. [9]

2.2.1 Importance of causal relations

This notion of causal relations is opposed to simple correlations. System dynamics studies differ from simple forecast based on variables that are statistically correlated (this correlation being non-causal). For example, a negative correlation between rainfall and skin cancer exists, that is, in regions where it often rains, there is less skin cancer than in regions with sunny weather. This does not mean that rain does not cause skin cancer.

A causal relation means that “an input variable has some *causal influence* on the output variable”. That is, if the cause variable changes, one can expect a change in the effect variable. But as already mentioned in the reasons why a dynamic system is complex, whether the change occurs or not, and the nature of the change if it does occur, depend on many other factors.

2.2.2 Importance of circular causality (feedback causation) over time

The causal relations described in the previous sub-section are one-way relations. The next step is to identify the dynamics of circular causalities over time, that is, *feedback loops*. For example, if it is true that births has a causal influence on population (births \rightarrow population), it is also true that, dynamically over time, the population has a causal influence on births.

In fact, these two relations have no meaning if they are taken statically, that is, at a precise moment in time. At this precise moment, “births cannot determine population *and* be determined by it”. [9] Thus, circular causality is only possible if the time is taken into account. The notion of “direct causality” is used to express the fact that $X \rightarrow Y$, meaning that they are directly causally related, given the other variables in the model.

2.2.3 Dynamic behaviour pattern orientation

The dynamic behaviour pattern orientation is the contrary of event-orientation. Modellers have to keep in mind that “dynamic problems are characterised by undesirable performance patterns, not by one or more isolated events”. [9] These problems need to be analysed and understood with regards to their past dynamics. If this is not the case, these events seem to be random and are meaningless.

2.2.4 Internal structure as the main cause of dynamic behaviour

Internal structure as the main cause of dynamic behaviour is also called the *endogenous perspective*. As mentioned before, a system is “a collection of interrelated elements, forming a meaningful whole”. How can this structure be represented? By a causal loop diagram (see next section). This diagram depicts the causal links and loops between variables. “The interaction of the feedback loops in a system is the main engine of change for the system: the structure causes the behaviour of the system”. [9] The importance of this sub-section is explained in the next section (“Causality”).

2.2.5 Systems perspective

The aim of the system is to have an internal structure with causal links and causal loops that are large enough so that it has an endogenous aspect. The more external forces influence the system the more this system is exogenous. But the more the system is endogenous, the less it is possible to apply correction in order to solve the problem. In general, dynamic problems arise from the fact that the system structure cannot cope with unfavourable external conditions. Thus, the dynamic problem is not the fault of one element in the system, but of the structure of the system itself. The model boundary determination is thus an issue in the system dynamics methodology.

2.3 Feedback representation

Two different types of feedback lead to two different representations of diagramming:

Information feedback: this kind of feedback, that is very important when building models, will lead to the *causal loop diagrams* where causality between variables is be represented.

Material feedback: this kind of feedback where “material” flows between stocks leads to the *stock-and-flow diagrams* where these stocks, flows and converters are be represented.

Three kinds of feedback loops are present in systems. The two first ones are described above, the third one is called *monotonic feedback loop* where there are quantities but where no conservation laws about these quantities like in the stock-and-flow systems exist.

In this section, the three kinds of feedback loops will be defined.

2.3.1 Causal loop diagram

Information feedback, causality, positive and negative causal effects, and an general example will be discussed in this subsection.

2.3.1.1 Causality

Since primary school, pupils are taught that every action has an effect, that is, one event or *cause* is related to another event or *effect* which is the direct consequence of the first event. This relationship is called *causality*.^[13]

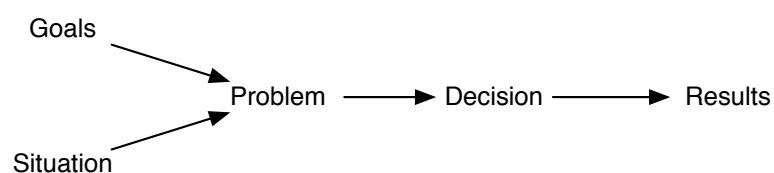


FIGURE 2.1: Event-oriented view of the World

For example, Figure 2.1 taken from [10] represents an “open-loop view of the world that leads to an event-oriented, reactionary approach to problem solving”. An organisation establishes goals and compares these goals with the current situation. The difference or gap between

these two variables creates the problem that the organisation will have to solve by taking decisions and obtaining results whether they are good or bad.

In real systems however, the actions an organisation takes will have an effect on its future, thus bringing new problems and new decisions. The left-hand side of Figure 2.2 taken from [10] shows that the decisions an organisation takes “alter its environment, leading to new decisions”.

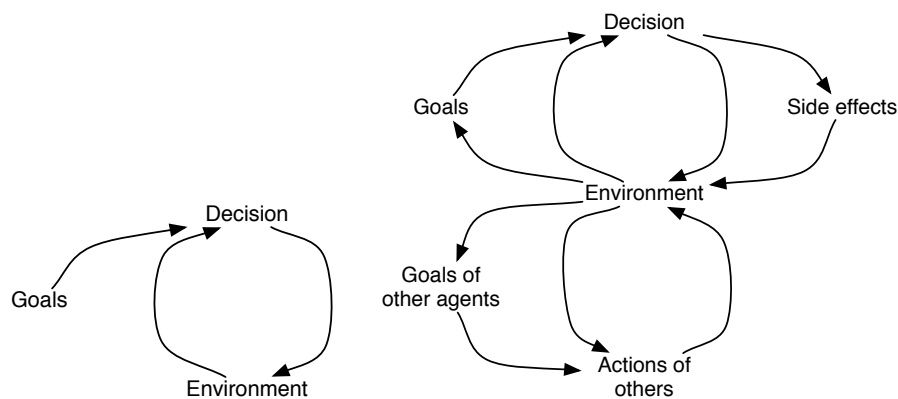


FIGURE 2.2: Feedback view of the World

A more realistic feedback view of the world is depicted in on the right-hand side of Figure 2.2, where side effects that are often unexpected and actions of others are represented. These actions and side effects influence the environment of an organisation. The latter has to take them into account when modelling its system.

2.3.1.2 Positive and negative causal effects and feedback loops

As mentioned in the previous section, $x \rightarrow y$ means that the variable x has a *causal effect* on the variable y . Also, a “feedback loop is a succession of cause-effect relations that start and end with the same variable”. [9] A feedback loop is positive or negative depending on the number of negative causal effect arrows in the cycle.

Positive causal effect “A change in x , ceteris paribus, causes y to change in the same direction” [9] For example Motivation \rightarrow^+ Productivity, therefore an increase in the motivation will influence the productivity in the same direction, which entails an increase in the productivity. Conversely, a decrease in the motivation will also influence the productivity in the same direction, which will lead to a decrease in the productivity.

Negative causal effect “A change in x , ceteris paribus, causes y to change in the opposite direction” [9] For example Frustration \rightarrow^- Studying, therefore an increase in a student’s

frustration will influence its number of study hours in the opposite direction, which will lead to a decrease in the number of study hours. Conversely, a decrease in the frustration will also influence the number of study hours in the opposite direction, which will entail an increase in the number of hours when the student studies.

Positive feedback loop Zero or an even number of negative causal effects in a cycle means that the loop is *positive* (see Figure 2.3).

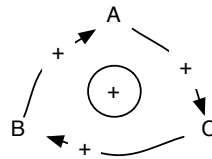


FIGURE 2.3: Positive feedback loop

Negative feedback loop An odd number of negative causal effects in a cycle means that the loop is *negative* (see Figure 2.4).

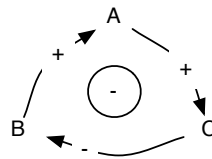


FIGURE 2.4: Negative feedback loop

Both positive and negative feedback structures will be described in more details in the next chapter.

2.3.1.3 Example

Here is another example to understand the difference between the traditional representation of cause and effect, and the system dynamics representation of causality: the student performance.

In Figure 2.5 taken from [14], the traditional linear representation of student performance is depicted. This example is the representation made by educational researchers interested in student performance in a classroom. This model assumes that the variable Student Performance does not influence the variable that causes effect on this variable.

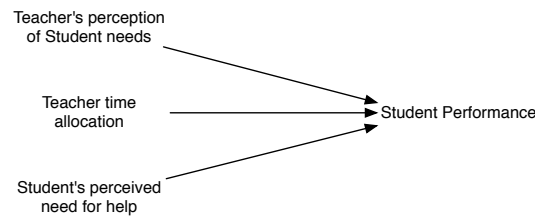


FIGURE 2.5: Traditional representation of student performance

A real world process is likely to have recursive relationships among the variables. System dynamics modellers will close the loop(s) (see Figure 2.6 also taken from [14]).

In this example, one can observe that the student performance is affected positively (in the same direction) by the help given by the teacher through the allocation of its time to help students. This allocation is affected by two variables: the teacher's perception of the student's needs and the student's need for help. These two variables are in turn influenced by the student's performance (if the student's performance increases, then the teacher's perception of this need will decrease and the student's need for help will also decrease).

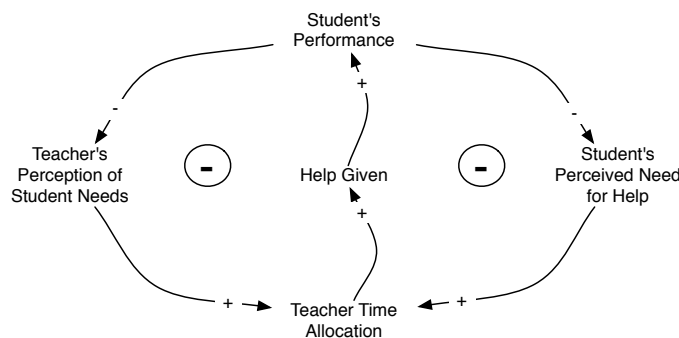


FIGURE 2.6: System dynamics feedback loops (non-linear) representation of student performance

The variables affecting the student's performance from Figure 2.5 are also present in Figure 2.6. The difference resides in the fact that the student's performance will in turn influence these variables.

2.3.2 Stock-and-flow diagram

Quantities in a system may be subject to conservation laws, that is, quantities flow from one stock to another (or from a source to a stock, or from a stock to a sink).

2.3.2.1 Diagram elements

In Table 2.1 taken from [15], the basic elements used in the stock-and-flow diagramming system are represented.



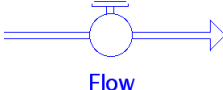


Element	Representation	Description
Stock		A stock is an accumulation over time. It can serve as a storage device for material, energy, or information. Contents move through stocks via inflow and outflow rates. They represent the state variable in a system and are a function of past accumulation of flows.
Source/Sink		Sources and sinks indicate that flows come from or go to somewhere external to the process. Their presence means that real-world accumulations occur outside the boundary of the modelled system. They represent infinite supplies or repositories that are not specified in the model.
Flow		Flows are the “actions” in a system. They affect the changes in stocks. Flows may represent decisions or policy statements. They are computed as a function of stocks, constants, and converters.
Converter		Converters help to elaborate the detail of stock and flow structures. A converter must lie in an information link that connects a stock to a flow.
Information link		Information linkages are used to represent information feedback. Flows, as control mechanisms, often require connectors from other variables (usually stocks or converters) for decision making. Links can represent closed-path feedback loops between elements.

TABLE 2.1: Stock-and-flow diagram elements

2.3.2.2 Diagram construction

It is important to start with stock as they are the key variables of the system modelled. “Stocks represent where accumulation or storage takes place in the system”. [16] To discover

what variable should be stocked, the modeller has to think of quantities that remain the same if the flows (in and out) are reduced to zero.

When the stock variables are chosen, the next step is to add flows that are the “actions” of the system. These flow will influence the stock for the next time step. The modeller also needs to double check the units used in the model. They should be the same as between flows and stocks modulo the unit of time. Flow directions are the following: one stock to another, one stock to a sink, and one source to a stock.

After that, the model will have strong bases but it will need more information. “If the stocks and flows are the nouns and verbs of a model, then the converters are the adverbs”. [16] They are, for example, the rates of flows, time, etc. They are information added to describe flows.

When a well-structured model is built, the final task is to write down equations for the model. These equations are easy in system dynamics due to the nature of the stock-and-flow structures (see Example).

2.3.2.3 Example

Here is an example taken from [16]. The stock-and-flow diagram (with the equations) will be depicted as well as the corresponding causal loop diagram.

Figure 2.7 shows the model for a plot of land where flowers grow and disappear. This area is limited to a certain number of acres of land that are suitable for flowers. Later in this dissertation, the behaviour of such a system will be described (see Section 4.2.1).

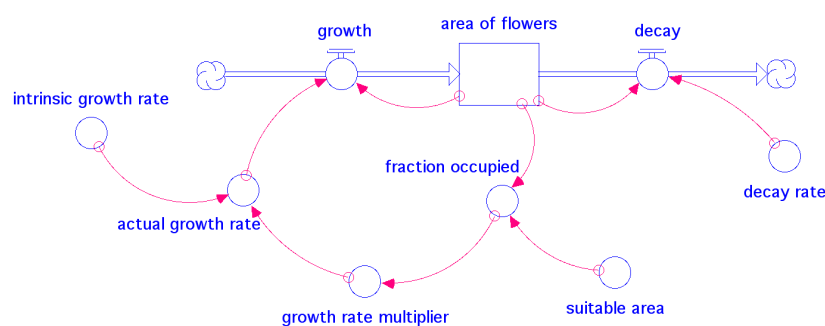


FIGURE 2.7: Flowered area: stock-and-flow diagram

A stock “area of flowers” represents the accumulation over time of flowered acres. This stock will be filled by the growth of new flowers and emptied by the decay of old flowers (these are the two flows). Further information is represented, such as the fact that the area is limited, etc.

Here is the list of equations related to the stock-and-flow diagram depicted in Figure 2.7 taken from [16]:

```

area_of_flowers(t) = area_of_flowers(t - dt) + (growth - decay) * dt
INIT area_of_flowers = 10.0
INFLOW: growth = area_of_flowers * actual_growth_rate
OUTFLOW: decay = area_of_flowers * decay_rate
actual_growth_rate = intrinsic_growth_rate * growth_rate_multiplier
growth_rate_multiplier = - fraction_occupied + 1.0
fraction_occupied = area_of_flowers / suitable_area
decay_rate = 0.2
intrinsic_growth_rate = 1.0
suitable_area = 1000.0

```

Figure 2.8 depicts the same model but represented with the causal loop diagramming system where only information feedback is represented. There are thus less variables than in Figure 2.7. In this case, one can observe the fact that if the area of flowers increases, then the growth and the decay increases as well but the observer does not know by how many.

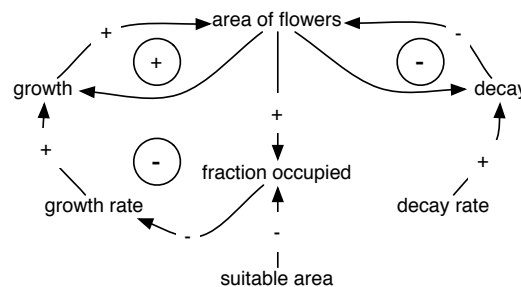


FIGURE 2.8: Flowered area: causal loop diagram

The difference between the causal loop diagramming (CLD) system and the stock-and-flow one (SFD) lies in the fact that the CLD's quantities can increase or decrease, this can be observed but the exact amount that is put or withdrawn from a variable is not visible. The SFD's represent this kind of information (material feedback) where one can observe the stock where material is added via inflows and withdrawn via outflows.

2.3.3 Monotonic feedback loop

Some systems have loops where quantities are represented. These quantities are not subject to conservation laws like the stock-and-flow systems. But they can be affected by saturation effects.

2.3.3.1 Hill equations

Researchers have analysed biological systems consisting of two interacting feedback loops.[17] One way of modelling non-linear monotonic interactions with saturation effect is to use the Hill equations. These equations are first-order non-linear differential equations capable of “modelling time evolution and mutual interaction of molecular concentrations”. [4] Here is an example of the Hill equations using two molecular concentrations X and Y (taken from [17]):

$$\begin{aligned}\frac{dY}{dt} &= \frac{V_X(X/K_{XY})^H}{1+(X/K_{XY})^H} - K_{dY}Y + K_{bY} \\ \frac{dX}{dt} &= \frac{V_Y}{1+(Y/K_{YX})^H} - K_{dX}X + K_{bX}\end{aligned}$$

A description of the above equations and of the following example are taken from [4]: “Here we assume that X activates Y and that Y inhibits X . The equations model saturation (the concentration of a molecule has an upper limit) and activation/inhibition with saturation (one molecule can affect another, up to a certain point). We see that X and Y , when left to their own, will each asymptotically approach a limit value with a exponentially decaying difference. Figure 2.9 shows a simplified system where X activates Y but Y does not affect X . X has a discrete step decrease at t_0 and a continuous step increase at t_1 . Y follows these changes with a delay and eventually saturates. The constants K_{dY} and K_{bY} model saturation of Y (and similarly for X). The constants V_X , K_{XY} and H model the activation effect of X on Y . We see that activation and inhibition have upper and lower limits.”

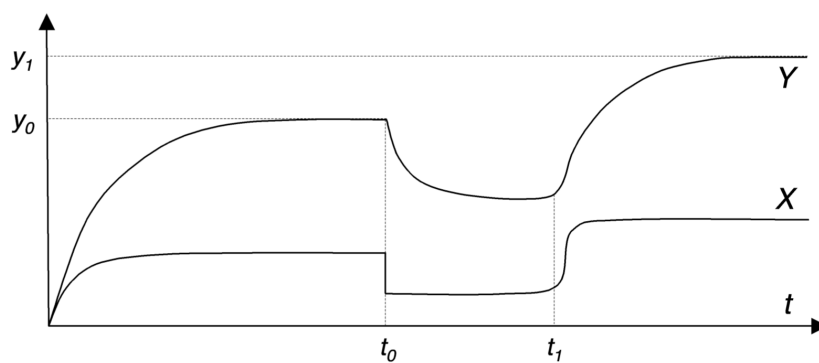


FIGURE 2.9: Biological system where X activates Y

2.3.3.2 Example

In this example, taken from [4], molecule concentrations are subject to saturation. One can observe in Figure 2.10 that the diagramming convention is the same as for the causal loop diagrams because there is no need to represent stocks and flows.

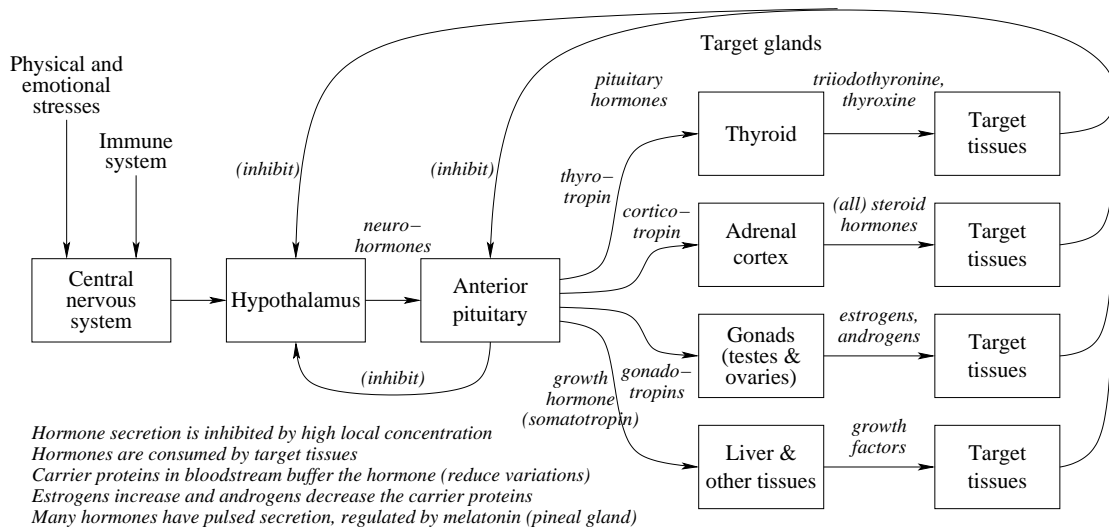


FIGURE 2.10: The hypothalamus-pituitarytarget organ axis

“The system contains two superimposed groups of negative feedback loops going through the target tissues and back to the hypothalamus and anterior pituitary. A third short negative loop from the anterior pituitary to the hypothalamus, and a fourth loop from the central nervous system.

The hypothalamus and anterior pituitary act as master controls for a large set of other regulatory loops. Furthermore, the nervous system affects these loops through the hypothalamus. This gives a time scale effect since the hormonal loops are slow and the nervous system is fast. Letting it affect the hormonal loops helps to react quickly to external events.” [4]

2.4 Modelling methodology

Most of the time, models are created to first simulate an existing system and then to observe what the behaviour of the system is if a change is applied to it (if the study is about a dynamic problem, the aim of studying the change in the behaviour of the system is to try to solve the problem). In this section, an eight-step modelling approach will be described. This approach comes from three articles [9, 15, 16] that have been put together to build it. The contribution of each article is mentioned in the corresponding points.

This methodology is partially done in group sessions where experts from the domain, systemic experts, and facilitators work together in order to develop the model. These are standard steps, variations may appear due to the nature of the problem. The modelling process is also iterative: a step might sometimes not be completely finished when the system modeller goes

to the next one. Thanks to the new ideas that emerge in the following steps, the modeller may go back to previous steps to change certain things, and so on.

This process is also specific for models where a stock-and-flow diagram can be built, that is the quantitative representation of a given system. Other kinds of models exist (described in previous sections) where quantities are neither preserved through the process nor is it possible to monitor them over time. Thus, some of the steps have to be adapted to the kind of system that the modeller wants to study.

2.4.1 Problem identification

The first step to be taken by a system modeller is to get acquainted with the system [16]. For example, a company asks a system modeller to model the system that is working in this company but that is not performing well, in order to find a solution to their problems. The model expert will first be introduced to the company and will then observe the behaviour of the surrounding environment. This person will analyse how people communicate with each other, how the system performs and why the company is performing poorly. The expert will also study the proposals that were made to solve the problem and why they did not work. If some dynamic data about the company exists, it might be interesting to try to plot the data to observe the behaviour.

Being specific about the dynamic problem is very important. Once the modeller has observed the system, he/she can draw what is called a *reference mode* [9, 16] where the time is represented on the horizontal axis and where an important variable is represented on the vertical axis. The time horizon should be long enough to represent the dynamic behaviour that summarises the problem. This graph should be recognised by the members of the organisation. This behaviour will probably correspond to one pattern described in the next chapters or maybe to the combination of two or more of these patterns.

Here is a summary of some sub-steps taken from [9]:

- Plot all the available dynamic data and examine the dynamic behaviours.
- Determine the time horizon (into the future and into the past) and basic time unit of the problem.
- Determine the *reference* dynamic behaviour: What are the basic patterns of key variables? What is suggested by data and what is hypothesised if there is no data? What is expected in the future?

- Write down a specific, precise statement of what the dynamic problem is and how the study is expected to contribute to the analysis and solution of the problem. Keep in mind that this purpose statement will guide all the other steps that will follow.

2.4.2 Model conceptualisation and dynamic hypothesis

The second step consists in developing hypotheses to explain the causes behind the problematic dynamics also called the *conceptual model*[9]:

- Examine the real problem and/or the relevant theoretical information in the literature.
- List all variables playing a potential role in the creation of the dynamics of concern.
- Identify the major causal effects and feedback loops between these variables.
- Construct an initial causal loop diagram and explore alternative hypotheses.
- Add and drop variables as necessary and fine-tune the causal loop diagram.
- Identify the main *stock* and *flows* variables.
- Finalise a dynamic hypothesis as a concrete basis for formal model construction.

2.4.3 Formal model construction

Once the causal loop diagram is built with sufficient confidence, and that the main stocks and flows are spotted, then the modeller can construct the stock-and-flow diagram and find the equations describing the cause-effect relations for all variables.

Depending on the modeller, the two previous steps can be interchanged. Some people have difficulties in describing a system with words and arrows like in the causal loop diagrams but work more easily with stocks and flows. The modelling methodology described in [9] recommends starting with the causal loop diagram and deriving the stock-and-flow diagram from the first one. On the other hand, [16] starts with the stock-and-flow diagram and explains in another chapter how to translate a stock-and-flow diagram into a causal loop diagram.

2.4.4 Estimation of the parameter values

Once the modeller has established the equation of the model, he/she has to estimate the values of the parameters and the initial value of stocks. In order to do so, the expert can

look into the existing data to find some coherent values, but sometimes this data does not exist and the modeller has to estimate it.

Generally, if the data does not exist, the expert will test the model with some initial values and try to find the ones that correspond the best to the hypotheses made earlier. In fact, what is important here is to move step-by-step in the estimation via the information spectrum [16] (see Table 2.2):

physical laws	controlled physical experiments	uncontrolled physical experiments	social system data	social system cases	expert judgement	personal intuition
------------------	---------------------------------------	---	--------------------------	---------------------------	---------------------	-----------------------

TABLE 2.2: The information spectrum

2.4.5 Model credibility testing

This step is an important one in order to confirm that model is a coherent representation of the real problem with respect to the study purpose. In [9], two kinds of credibility tests are described:

Structural: Is the structure of the model a meaningful description of the real relations that exist in the problem of interest? For example, the modeller can ask experts to evaluate the model structure, the dimensional consistency with realistic parameter definitions and the robustness of each equation under extreme conditions.

Behavioural: Are the dynamic patterns generated by the model close enough to the real dynamic patterns of interest? These tests are designed to compare the major pattern components in the model behaviour with the pattern components in the real behaviour. For example, one can examine the slopes, maxima, minima, periods and amplitudes of oscillations, inflection points, etc.

One has to keep in mind that if the structure of the model is not validated, then the behaviour validity is meaningless and the comparison between the behaviour of the model and the behaviour of the reality is not a point-to-point comparison but a pattern comparison.

If the results of the simulation do not match the reality, it is necessary to go back to previous steps to redefine some parts of the diagrams.

2.4.6 Sensitivity analyses

The tests that are run in this step try to make the modeller understand the important dynamic properties of the system. Tests with different values for the main parameters are conducted in order to learn whether the basic pattern of results is “sensitive” to changes in the uncertain parameters. The aim is to obtain the reference mode after each test, in this case the model can be called *robust*, that is, “when it generates the same general pattern despite the great uncertainty in parameter values”. [16]

But if the model is not robust, the modeller needs to go back to the estimation of values for the parameters of the system.

2.4.7 Impact of policies testing

Now that the model is fully tested and the properties are understood, the modeller may try alternative policies in order to possibly improve the dynamics of the system. “A *policy* is a decision rule, a general way of making decisions”. [9] Thus the modeller can test alternative policies, and if they produce interesting results, then the modeller might go to the previous step and run sensitivity tests on the policies. These policies can, in a certain way, improve the design of the model.

“How can experts know when a simulation is good or bad?” This question can be answered by [16]:

“A system dynamics model may be used to simulate changes in dynamic behaviour due to a change in policy. The question of whether the simulated changes are *good* or *bad* is not amenable to system dynamics analysis. Such questions are best addressed with evaluation tools from the field of decision analysis. Formal evaluation tools are designed to reveal trade-offs between competing goals and the possibility that different groups may assign entirely different priorities to the goals.”

2.4.8 Model validation

One may wonder whether the model that he/she has built is valid and may want to prove that it is valid. This sub-section will discuss model validation and propose concrete tests to build confidence in the model. [16]

2.4.8.1 Validity or usefulness?

Once a modeller has gone through all the steps described in the previous sub-sections, and the model has generated the expected behaviour, as depicted in the reference model, and that he/she has also found policies that can improve the system, then the expert is faced with one remaining question: "Can you prove the model is valid?".

The fact that the modeller uses formal mathematical methods with sophisticated software and fast computers might give the impression that the model is the perfect match with the reality. But the problem is always the same: "models are simplifications of the system under study".[16] The question should be "Is the model useful?". What can the expert do? Specify the purpose of the model and alternative ways to achieve this purpose. This is therefore no longer about validation but rather about building confidence in the new model.

2.4.8.2 Concrete tests to build confidence

This point describes five tests that deserve some attention[16] and then three tables that are presented with a non exhaustive summary of all possible validation tests. They consider the model's suitability for purpose, consistency with reality, utility and effectiveness from both structural and behavioural perspectives.[15]

Verification: "A model may be *verified* when it is run in an independent manner to learn if the results match the published results. The goal is simply to learn if the computer model *runs as intended*".[16] What the modeller can do is to give his/her model to another modelling group and ask them to run it. If the results correspond, then the model can be flagged as *verified*. But the task can sometimes be complicated by the fact that there is no documentation or because the model is too large.

Face validity: This test is the simplest one to carry out. It consists in verifying the structure and the parameter values. For example, an arrow pointing in the wrong direction or a parameter that is negative and must be positive, etc. This is common sense but in large organisations, models can be so large that no one carries out these face validity tests.

Historical behaviour: Sometimes when a modeller models a system where historical data is available, a test to build confidence in the model he/she is building is to set inputs of the model to their historical values and observe whether the output matches the historical behaviour.

Extreme behaviour: This test consists in making major changes in model parameters and observing whether the response is plausible.

Detailed model check: “If the modeller is working on an important topic within a large organisation, it is quite possible that there are several models describing different aspects of the system. In some cases, the other models may provide a more detailed and accurate representation of some aspect of the system. If it is true, the modeller should take advantage of these models and provide benchmark simulations that may be used to check his/her own model.” [16]

TABLE 2.3: Model validation tests - suitability for purpose

Focus	Test	Passing Criteria
Structure	Dimensional consistency	Variable dimensions agree with the computation using right units, ensuring that the model is properly balanced
	Extreme conditions in equations	Model equations make sense using extreme values
	Boundary adequacy Important variables Policy levers	Model structure contains variables and feedback effects for purpose of study
Behaviour	Parameter (in)sensitivity Behaviour characteristics	Model behaviour sensitive to reasonable variations in parameters
	Policy conclusions	Policy conclusions sensitive to reasonable variations in parameters
	Structural (in)sensitivity Behaviour characteristics	Model behaviour sensitive to reasonable alternative structures
	Policy conclusions	Policy conclusions sensitive to reasonable alternative structures

TABLE 2.4: Model validation tests - utility and effectiveness of a suitable model

Focus	Test	Passing Criteria
Structure	Appropriateness of model characteristics for audience Size Simplicity / complexity Aggregation / detail	Model simplicity, complexity and size is appropriate for audience
Behaviour	Counterintuitive behaviour	Model exhibits seemingly counterintuitive behaviour in response to some policies but is eventually seen as implication of real system structure
	Generation of insights	Model is capable of generating new insights about system

TABLE 2.5: Model validation tests - consistency with reality

Focus	Test	Passing Criteria
Structure	Face validity Rates and levels Information feedback Delays	Model structure resembles real system to persons familiar with system
	Parameter values Conceptual fit Numerical fit	Parameters recognisable in real system and values are consistent with best available information about real system
Behaviour	Replication of reference modes (boundary adequacy for behaviour) Problem behaviour Past policies Anticipated behaviour	Model endogenously reproduces reference behaviour modes that initially defined, the study including problematic behaviour, observed responses to past policies and conceptually anticipated behaviour
	Surprise behaviour	Model produces unexpected behaviour under certain test conditions: (1) model identifies possible behaviour, (2) model is incorrect and must be revised
	Extreme conditions simulation	Model behaves well under extreme conditions or policies, showing that formulation is sensible
	Statistical tests Time series analyses Correlation and regression	Model output behaves statistically with real system data; shows same characteristics

Chapter 3

Feedback systems - building blocks

Introduction

Whenever a system changes, inputs and outputs are always present. The inputs reflect the influence of the environment on the system, and the outputs reflect the influence of the system on the environment. Inputs and outputs are separated by time, as in before and after.

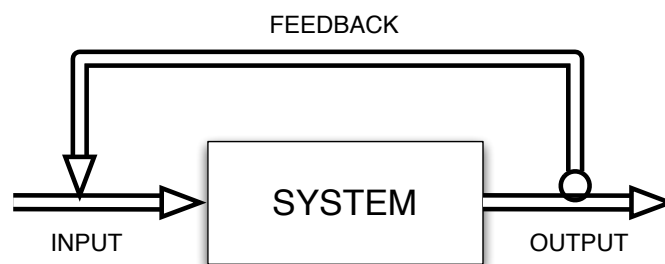


FIGURE 3.1: A system with feedback

But, if the output (the information about the change of the system) is sent back to the system as an input, we can call this kind of loop a *feedback loop*.

In this chapter, basic ideas about feedback loops such as negative and positive feedback loops, coupling, management, stigmergy, dominant loop shift, etc. will be discussed.

3.1 Single loop systems

In this section, three types of single feedback loops, and some features like delays and oscillation will be described.

3.1.1 Feedback loop

A feedback loop consists of three elements that interact together with a subsystem (see Figure 3.2): [18]

- **Monitoring agent:** monitors the state of the subsystem and sends this 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.

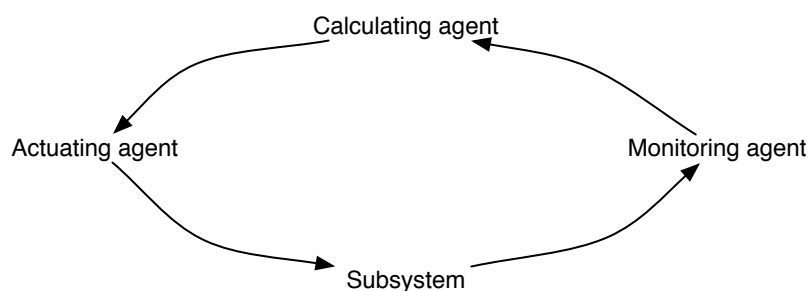


FIGURE 3.2: Basic structure of a feedback loop

Not all feedback loops that can be observed have the same structure as this one, especially in system dynamics where we only have variables.

3.1.2 Reactive loop versus proactive loop

Two ways of 'triggering' a loop exist: a loop can respond to an event (stimulus) or a loop can anticipate this event (stimulus) and triggers itself.

A *reactive loop* reacts to an event triggered by the system. Examples of such configurations have been defined in Section 1.2. For example, in the self-healing case, the sensors detect 'failure' or 'leave' events from a file group in the resource (events are triggered). The File Replica Aggregator receives these events and creates another event that will be

perceived by the File Replica Manager. This component will take actions according to the event. Thus, the 'failure' or the 'leave' event from a file group triggers a chain reaction.

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. Figure 3.3, taken from [6], depicts an example where a proactive manager is described. In this example, the Load Balancing Manager queries the Storage Aggregator every x minutes to know the most and least loaded storage components. When the manager receives the answer from the other agent, it can act and move files if this is deemed necessary (i.e. from the most to the least loaded component).

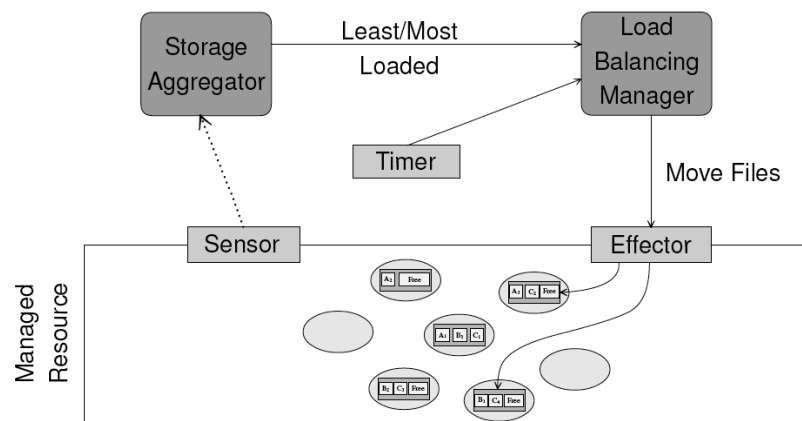


FIGURE 3.3: A proactive monitor

3.1.3 Positive feedback loop

When the feedback is reintroduced into the system as new data, if this data facilitates and accelerates the transformation in the *same* direction as the previous change, then the loop is called a *positive feedback loop*.

In system dynamics, if the algebraic product of all signs of all the cause-effect relations that start and end with the same variable is positive, then the loop is *positive* or *compounding* or *reinforcing*.

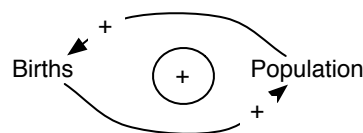


FIGURE 3.4: Births: a single positive feedback loop (CLD)

In Figure 3.4 taken from [9], one can observe that more births means a higher population, which in turn means more births and then an increase in the population, and so on. This loop creates an exponentially growing population. This example is not realistic on its own, the stable state of this system would be an infinite population. We will discuss the same example later, coupled with other examples to build a more realistic system.

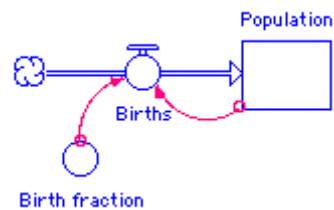


FIGURE 3.5: Births: a single positive feedback loop (SFD)

The similarity between the two kinds of representations is very clear. Figure 3.4 represents the qualitative aspect of the system whilst Figure 3.5 depicts the quantitative aspect. The only difference between these two diagrams is the presence of Birth fraction in the stock-and-flow diagram. In fact this notion of “what is the percentage of new arrivals in the population at time $t+1$ ” is implicit in the the causal loop diagram.

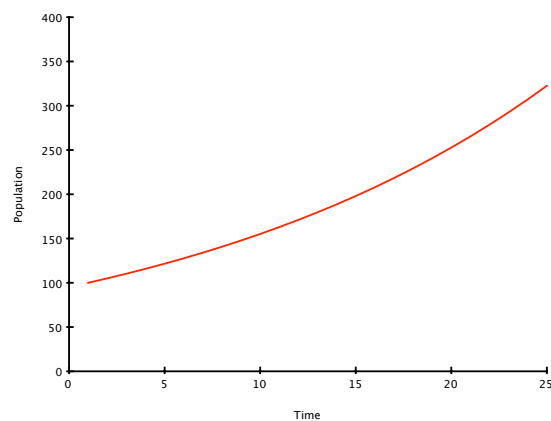


FIGURE 3.6: Births: a single positive feedback loop (Behaviour)

This model was tested with an initial population of 100 people and a percentage of newborns equal to 6% over a period of 25 cycles. The results are plotted in Figure 3.6 where the real behaviour of a positive linear feedback loop can be observed, that is, an *exponential growth*. But not all positive feedback loops have the same behaviour, some can create exponential decay. For example, if we consider the value of a stock in a stock market and the fact that a critical value for this stock exists (below this value, people sell this stock). So, the more the value drops, the more people sell, the more it drops, and so on. This behaviour is called *crash* or *collapse* or *vicious loop*. [9]

Some studies in cellular networks [17] have shown that a positive feedback loop tends to slow down the response to a stimulus. That is, it does not help to make instantaneous decisions for critical or lethal decisions but it helps to avoid careless decisions. Thus, one other possible utility of a positive feedback loop is to *make good decisions*.

3.1.4 Negative feedback loop

When the feedback is reintroduced into the system as new data, if this data produces a result in the *opposite* direction to the previous change, then the loop is called a *negative feedback loop*.

In system dynamics, if the algebraic product of all the signs of all the cause-effect relations that start and end with the same variable is negative, then the loop is *negative* or *balancing* or *goal-seeking*.

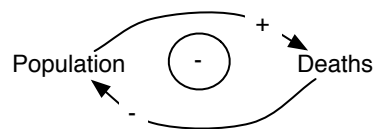


FIGURE 3.7: Deaths: a single negative feedback loop (CLD)

In Figure 3.7 taken from [9], one can observe that more deaths means a lower population, which means more deaths and then even less population, and so on. This loop creates an exponential decay of the population. This example is also not realistic as the stable point would be a zero-population. A more realistic example will be discussed later.

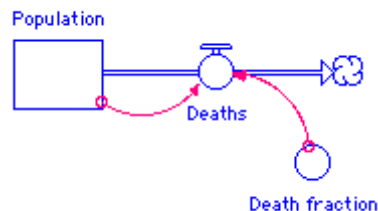


FIGURE 3.8: Deaths: a single negative feedback loop (SFD)

As has already been seen in the previous point, there is no great difference between the causal loop (Figure 3.7) diagram and the stock-and-flow (Figure 3.8) diagram, except for the *Death fraction*, which is once more implicit in the CLD.

This model was tested with an initial population of 300 people and a percentage of deaths of 6% over a period of 25 cycles. The results are plotted in Figure 3.9 where the real behaviour of a negative linear feedback loop can be observed, that is, an *exponential decay*.

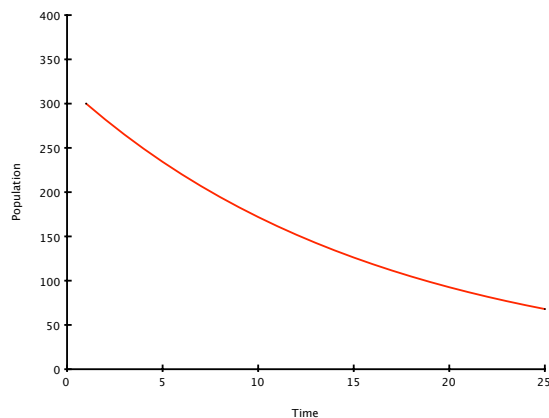


FIGURE 3.9: Deaths: a single negative feedback loop (Behaviour)

Another typical example of a negative feedback loop is the *regulation* of the temperature in a room with a thermostat.[9] The desired temperature is entered into the thermostat. If it is cooler in the room, then the thermostat will activate the heating system. If it is hotter, then it will activate the cooling system. The resulting behaviour is a goal-seeking one, where the aim is to reach the desired temperature in the room.

In Figure 3.10, one can observe the causal loop diagram for the system described above. Some causal effects can be explained: Room temperature \rightarrow Discrepancy is a negative effect because the equation of discrepancy is the difference between the desired temperature and the actual temperature in the room. Thus, if the room temperature increases, then the discrepancy will decrease. The adjustment time has a negative impact on the temperature change, due to the fact that this change is defined by the discrepancy divided by the time. If the time is increased, then the change will be smaller for two same discrepancies.

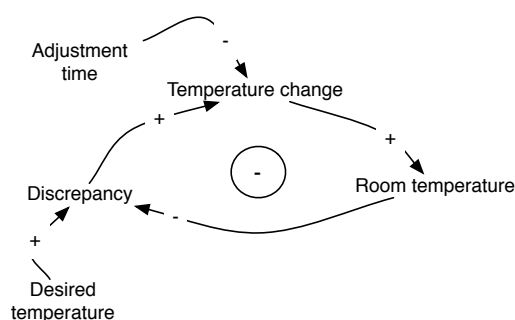


FIGURE 3.10: Room temperature regulation via a thermostat (CLD)

The corresponding stock-and-flow diagram depicted in Figure 3.11 is the direct translation of the CLD. The temperature in the room is represented as a stock, and temperature change is represented as the inflow, due to the definition of “change” which can be positive or negative.

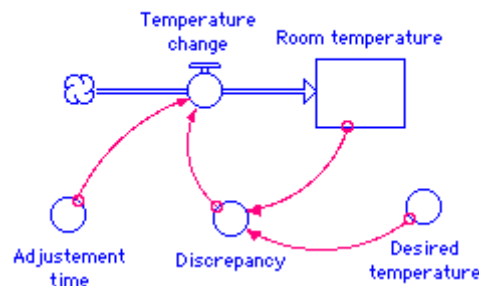


FIGURE 3.11: Room temperature regulation via a thermostat (SFD)

Tests were carried out using different values for the parameters of the system. The desired temperature was set at 20.0 (degrees celcius) and the adjustment time at 5.0 (units of time). Only the value for the initial temperature was changed (to 30.0 and to 10.0). In fact, had the adjustment time been changed, then the two curves would have been steeper at the beginning (for a shorter time) or less steep (for a longer time). Results are plotted in Figure 3.12

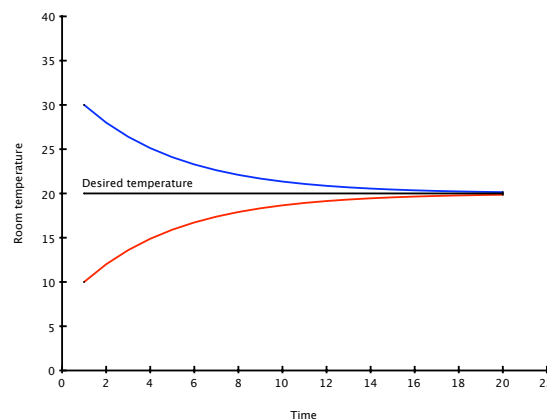


FIGURE 3.12: Room temperature regulation via a thermostat (Behaviour)

The behaviour pattern is indeed the decay. One can observe such a behaviour if the initial temperature is higher than the desired one. But this observation is less obvious if the opposite happens. In this case, what decays exponentially is the discrepancy.

3.1.5 Time delays

Time delays often play an important role in the dynamic of systems. Significant time lags may intervene between causes and their effects.[9] Their influence on the system may have enormous consequences, frequently accentuating the impact of other forces.

"This happens because delays are subtle: usually taken for granted, often ignored altogether, always under-estimated. In reinforcing loop [positive feedback loop], delays can shake our confidence, because growth doesn't come as quickly as expected. In balancing loop [negative feedback loop], delays can dramatically change the behaviour of the system. When acknowledged delays occur, people tend to react impatiently, usually redoubling their efforts to get what they want. This results in unnecessarily violent oscillations." [19]

In a regulation system where negative feedback loops and time delays are present, oscillations in the behaviour are more likely to appear. For example, "long after sufficient corrective actions have been taken to restore a system to equilibrium, decision makers often continue to intervene to correct apparent discrepancies between the desired and actual state of the system".[10]

3.1.6 Oscillation

Growth and decay may be associated with some oscillation patterns, generally due to time delays. Oscillations can be characterized by their amplitude (the height of the variable at its peak) and their period (length of time before the oscillation repeats itself). Four types of oscillations exist[19] :

Sustained oscillations: oscillations where neither the amplitude nor the period change.

For example, if a pendulum is in an environment where no frictional forces exist, its movements will draw "perfect" oscillations in its dynamics.

Damped oscillations: oscillations where the period is fixed but where the amplitude decreases continuously to eventually make the system come to a stop. One can for example consider a pendulum in an environment where friction forces exist: the amplitude of the pendulum will progressively decrease until the oscillations stop.

Growing oscillations: oscillations where the period is fixed but where the amplitude grows from one oscillation to another. This kind of oscillation can be found in an unstable system. Due to its behaviour, this type of oscillation, that tends to lead the system to its self-destruction, is not normally observed in natural systems.

Limit cycles: when a limit is encountered, the oscillations can grow into a limit cycle and may remain stable, like the rhythmic beating of a human heart.

3.2 Two-loop systems

How do feedback loops interact? How can feedback loop systems be coupled? What are the dangers? What types of coupling exist? All these questions will be answered in this section.

3.2.1 Stigmergy

The first kind of interaction between two loops is *stigmergy*: two loops monitor and affect a common subsystem.

In an example taken from Norbert Wiener [11] and shown in Figure 3.13, 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.

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

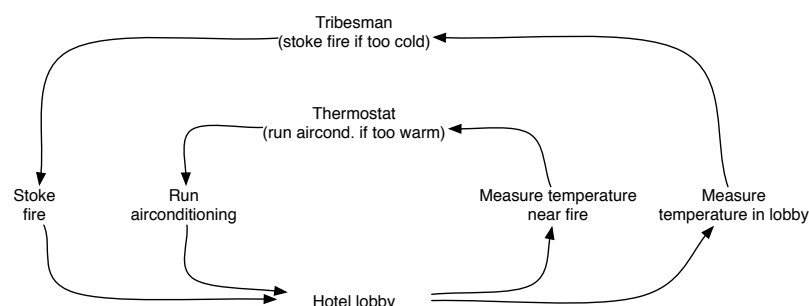


FIGURE 3.13: Tribesman: two feedback loops interacting through stigmergy

In this example, the loops taken separately are negative feedback loops and are stable. But, if the whole system is taken into account, the loops are unstable. Even if a negative loop is added to an existing system with the aim of ensuring stability, this is insufficient. The new result could be unstable because of the new interaction of the loop with the old system.

“Stigmergy should then be used with care” [5]. The example depicted above 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).

3.2.2 Management

The second kind of interaction between two loops is called *management*: one loop directly controls another loop.

The correct solution to the example depicted in Figure 3.13 is presented in Figure 3.14. 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. The outer loop is now managing the inner one and the system tends to a stable state. This is the illustration of a design rule: “to modify a system’s behaviour, the right way is to work with the system and not to try to bypass it”. [18] This example with the tribesman shows that sometimes management is preferable to stigmergy.

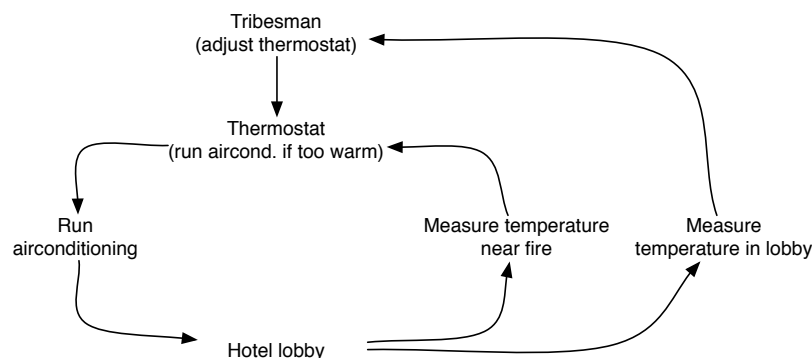


FIGURE 3.14: Tribesman: two feedback loops interacting through management

Management can be seen as a ‘data abstraction’ [5] where a complex component can manage an inner loop without having to know or to understand the details of this inner loop. It just has to know some of the parameters to interact with it.

Management should also control a ‘natural parameter’ [5]. The outer loop of the management pattern should adjust a simple parameter of the inner loop and no other interaction should be needed.

3.2.3 Coupling two positive feedback loops

Studies have shown that in cellular networks where signals are sent all the time, positive feedback loops amplify signals but elongate the time needed to make a decision. [17] Such

results may impair the decision making process for critical or lethal stimuli, but it avoids careless decisions being made. The main role of a positive feedback loop is thus to make important decisions.

Coupling two positive feedback loops induces a slower but amplified signal response and enhances bistability.

3.2.4 Coupling two negative feedback loops

Also in cellular networks, negative feedback loops maintain the homeostasis (the property of a system that regulates its internal environment and tends to maintain a stable, constant condition) of cellular systems.[17] Because of the signal reduction, they are also considered as noise filters. They help to make a prompt decision for strong and critical stimuli.

If two negative feedback loops are coupled, their behaviours are further enhanced. The response time is also accelerated. These two coupled negative feedback loops enhance sustained oscillations and homeostasis.

3.2.5 Coupling a linear positive feedback loop with a linear negative feedback loop

In the description of single positive and single negative feedback loops, it was concluded that both examples of births and deaths are worthless because their stable state is infinite or zero-population respectively. No real quantity can grow forever, there will always be natural limits to such growth. The nature of the negative feedback loop is to counteract change. If we couple the two kinds of loops (as linear feedback loops), the result is more interesting but still useless (see Figure 3.15 taken from [9]). In this case, the two fractions (birth and death) are explicitly added to the causal loop diagram.

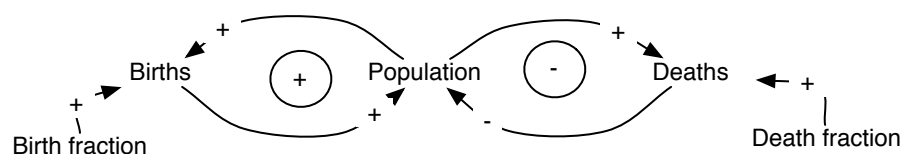


FIGURE 3.15: Population model: simple case (CLD)

Let us suppose that the population is calculated every year, the resulting behaviour will depend on which loop is dominant. A loop is dominant in this example if its fraction is higher. The fractions are constant in this example. If the births loop is dominant (birth

fraction $>$ death fraction), then the population will increase exponentially but if the deaths loop is stronger, then the population will decrease. If both loops have the same effect on the population, then the latter will not change. An example with non-linear coupling of population loops will be discussed in the next chapter.

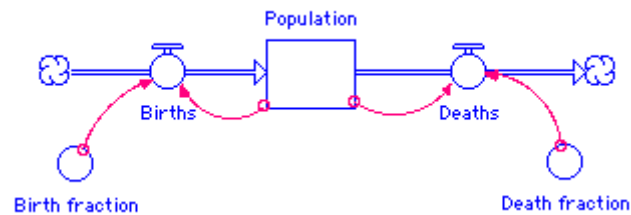


FIGURE 3.16: Population model: simple case (SFD)

Figure 3.16 is the direct translation of Figure 3.15 in a stock-and-flow diagram. This model was tested with different values for the birth and death fractions. The initial population was fixed at 1.000 people and the following values were set for the fractions (all behaviours are plotted in Figure 3.17:

- *Birthfraction* = 0.6 and *Deathfraction* = 0.6: in this case neither of the loops is dominant. The population will not change because at any given time t , the exact same number of people will be added and withdrawn. (red line)
- *Birthfraction* = 0.6 and *Deathfraction* = 0.3: the positive feedback loop is now the dominant one and an exponential growth of the population can be observed. The stable point is an infinite population. (blue line)
- *Birthfraction* = 0.3 and *Deathfraction* = 0.6: the negative feedback loop is now dominant causing the population to decrease progressively until it reaches the limit of a zero-population. (black line)

In cellular networks, studies have shown that coupling a positive and a negative feedback loop results in obtaining the advantages of both kind of loops, and that it is the most common regulatory motif.[17]

3.2.6 Loop dominance shift

In the example depicted in Figure 3.15, tests were carried out on the model and different behaviours were extracted. The birth and death rates were fixed. For example, an initial population of 200 was chosen. The linearity of the loops and the fact that the fractions are fixed means that the dominant loop will be known before the test.

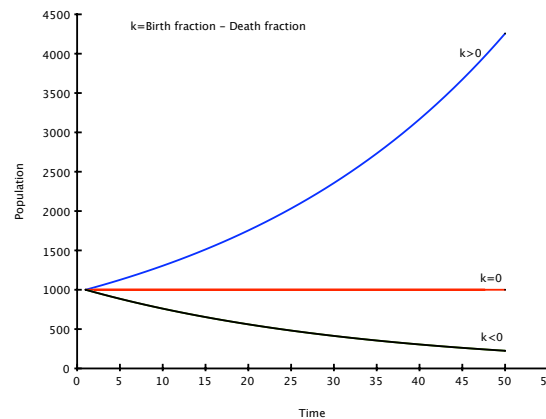


FIGURE 3.17: Population model: simple case (Behaviour)

Here, a more realistic example is analyzed, taken from [10], where the loop dominance shift can be observed. In this case, a firm that is about to launch a new product that creates an entirely new category with substantial market potential, but for which no market yet exists. The managers want to know how the market will develop, how they can stimulate adoption, when and how the market will saturate, etc. All these questions can be answered with a causal loop diagram and the corresponding stock-and-flow diagram that can be run through simulation.

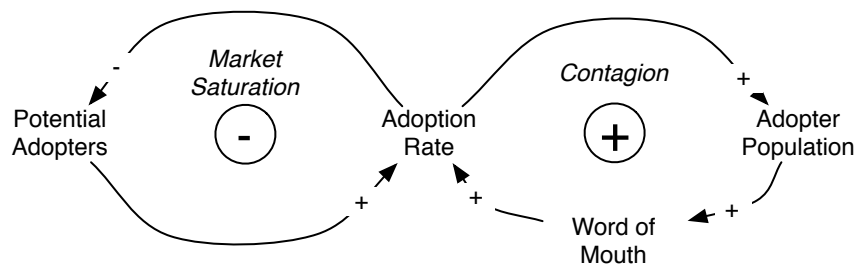


FIGURE 3.18: New product on a market: loop dominance shift

Here is the conclusion taken from [10]: “The overall dynamics of the system depend on which feedback loops are dominant. [...] For a sufficiently attractive innovation, the self-reinforcing word of mouth loop dominates initially, and the adoption rate and adopter population grow exponentially. The growing rate of adoption, however, drains the stock of potential adopters, eventually constraining the adoption rate due to market saturation. The dominant feedback loop shifts from the positive contagion loop to the negative saturation loop. The shift in loop dominance is a fundamentally nonlinear process, which arises in this case because adoption requires a word of mouth encounter between an adopter and a potential adopter. The shift in loop dominance occurs at the point where the adoption rate peaks. The behaviour of

the system shifts from acceleration to deceleration, and the system gradually approaches equilibrium.”

3.2.7 Coupling fast and slow positive feedback loops

In the cell signaling domain, some researchers have found the way to create an optimal bistable switch.[20] The two most important aspects when evaluating the performance of a bistable switch are the sensitiveness and the robustness.

Their research showed that for a single-loop switch, the “one-fast-loop” switch is sensitive to stimuli but unstable against noise, but the “one-slow-loop” switch is more robust to noise but slowly inducible. They also demonstrated that for two-loop switches

- the “two-fast-loop” switch is rapidly inducible but may undergo frequent transitions between two states driven by noise.
- the “two-slow-loop” switch is resistant to noise but is slowly inducible.
- the “dual-time” switch exhibits a sensitive robustness, capable of yielding a fast yet robust response.

They concluded that the dual-time switch is the optimal one. The fast loop is responsible for the speed of switching between two states, while the slow loop is crucial for the stability of steady states.

Modellers should take advantage of different time scales.[5] To illustrate this sentence, an example from [11] will be explained. A man drives his car on a slippery road but he does not know how slippery it is. The man will then ‘test’ the road by small and quick braking attempts. The reaction of the car will provide information for the human to adjust his braking attempts. The process of gathering information can be seen as a quick feedback loop that provides information to a slower regulation loop. In this case, the fast loop manages the slower one.

3.2.8 Complex components should be sandboxed

This idea comes from [5] where an example is explained about the human respiratory system (see Section 3.3). They conclude that complex components are introduced to stabilise unstable systems but they can also introduce instability. This is why the introduction of complex components triggers the need for a monitoring system (an outer feedback loop) that can take action if the system becomes unstable.

3.2.9 Use push-pull to improve regulation

[5] gives two examples from biology where hormones are used to regulate a substance. For example, the glucose level in the blood stream: this substance is regulated by glucagon and insulin. The pancreas produces the two hormones (α cells produce glucagon and β cells produce insulin). If the level of glucose increases then the glucagon level decreases and the insulin level increases, and vice versa. Both hormones act on the liver to release glucose in the blood stream.

The two hormones act in opposite directions in order to regulate a substance. Thus, in order to improve the regulation usually done with one negative feedback loop, the idea is to add another feedback loop that works in the opposite direction to the first one. This pattern allows a parameter to change quickly in both directions. It is either 'pushed' or 'pulled' in one direction.

3.2.10 Reversible phase transitions

The behaviours of feedback structures can show abrupt changes that can be seen as *phase transition*. [5] It is important that these phase transitions are reversible. If this is not the case, the system could stay in an unstable state forever. This concern has been studied in the SELFMAN project where structured overlay networks could be in three different phase (liquid, solid or gaseous).

3.2.11 $1 + 1 = 3?$

In general, feedback loops are associated with a policy in lots of domain. Sometimes, one might think of combining these policies because, taken into account separately, they are very efficient. But at times, the result of this combination is not the expected one and to find the source of the problem might be interesting. More specifically, in large software systems where adaptive components are more frequent than ever, the performance is one of the most important criteria when developing a big application. This criterium calls for adaptive capabilities to reduce the need for manual tuning or maintenance.

Researchers suggest a way to discover incompatibilities between adaptation policies through the use of what they call *adaptation graphs*. [21] In fact, their adaptation graphs are synonymous with causal loop diagrams because nodes also represent key variables, in the same way that directed arcs represent the direction of causality. These arcs are labelled with their policy name. What is innovative here is their check for potential incompatibility.

Sometimes, when two policies are combined, a new loop may emerge from nowhere and this loop is unexpected but may also have dramatic effects on the system. It is better to avoid this kind of loop, and, in order to do so, researchers suggest carrying out these two checks on the new directed graph. A key requirement for them is that all cycles in the adaptation graph must be negative as it stems directly from stability conditions in control theory.

Positive feedback: as mentioned above, only negative feedback loops are allowed in an adaptation graph. Thus any positive loop can combine two policies that are potentially incompatible. Positive loops are unsafe by nature because a change reinforces itself by causing more changes in the same direction.

Unstable negative feedback: even if a negative loop in a single policy tends to be well-tuned and hence stable, emergent loops raised by the combination of two or more policies must be analyzed for stability. In fact, every loop where not all the arcs have the same policy label, are flagged as potentially unintended interactions.

After this check, some loops may be flagged. They must be analyzed to establish whether their behaviour is intended. If not, the two or more policies are incompatible.

3.3 Example with several basic patterns

In Figure 3.19 taken from [4], one can observe four feedback loops: three of them are interacting through management and the last one interacts through stigmergy. This model represents the human respiratory system. Some design rules can be extracted from observing the whole system.

Fail-safe: this pattern contains two feedback loops: a negative loop and a complex loop (neither positive nor negative) - the conscious control of body and breathing. These two loops interact through management: the negative loop manages the complex one. In fact, if the complex loop makes a mistake, the negative loop takes control and makes the person faint, for example.

Data abstraction: these are also two feedback loops that 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. In this example, a negative loop is responsible for the breathing movements and the complex loop controls the breathing speed.

Management tower: this pattern is the combination of the two patterns described above.

The previous patterns are both kinds of management interactions, so the combination gives a tower of management which is observable on Figure 3.19 (the three outer loops). In this example, the conscious control is secured by the fail-safe system and implements the breathing system via an abstraction pattern.

Event inhibitor: this pattern is the combination of two patterns described earlier in this chapter: management and slow and fast combination. In this case, a slow negative loop controls a fast positive loop. When something bad occurs, the fast positive loop is triggered but after a few seconds the negative loop deactivates the positive one. In this case, laryngospasms (positive loop) is triggered if something obstructs the airway and after a few seconds the breathing reflex (negative loop) takes over and re-establishes a normal breathing.

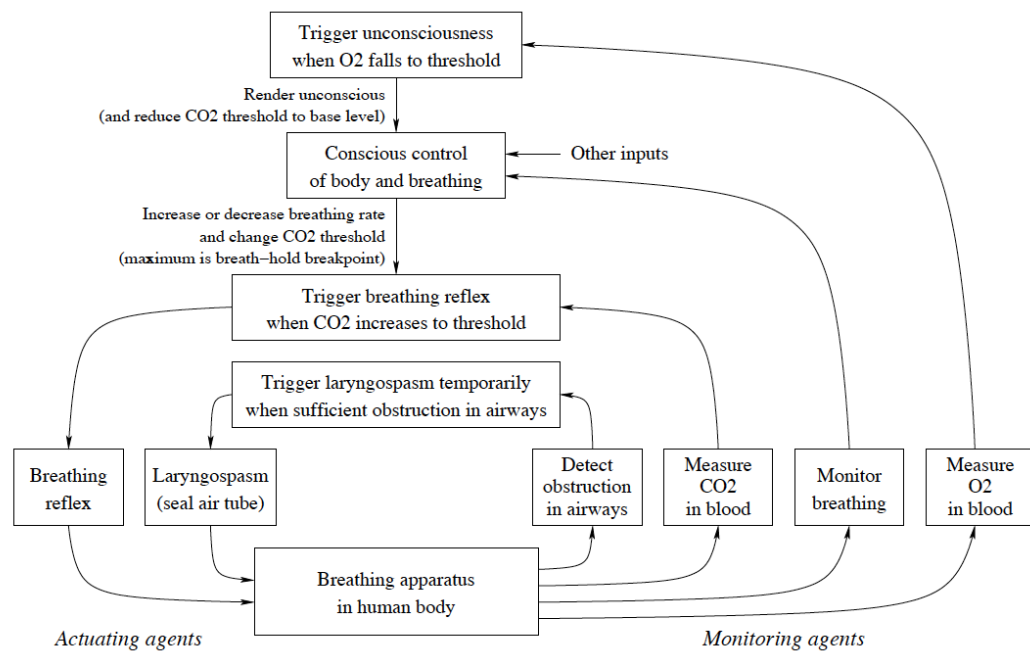


FIGURE 3.19: The human respiratory system: a feedback loop system

3.4 Decomposition of big systems

Sometimes systems can be very large and complex to understand. In this section, two ways of simplification are presented.

3.4.1 Reduction method to lower high-dimensional ODE

This first method is used in the analysis of genetic switches [22]. Researchers in this domain have proved that “networks with only positive feedback loops have no stable oscillations but stable equilibria whose stability is independent of the time delays. In other words, such systems have ideal properties for switch networks and can be designed without consideration of time delays”. [22]

To do so, they propose a method to simplify complex systems. First they had to prove that genetic networks with only positive feedback loop converge to stable equilibria and do not have any dynamic attractors that modify their expected behaviour. Then, they proved that equilibria are stable in these special networks. Finally, they gave a reduction method to simplify high-dimensional ordinary differential equations (ODE) to lower-dimensional ones.

The basic idea of the reduction procedure is to obtain a graph with only self-feedback edges, that is, edges coming and going from the same node. Figure 3.20 shows how to remove a non self-feedbacked node from the graph. Two possible situations exists. In both case, the node 2 is the node to be removed.

“Then for all nodes from which an edge goes out to the target node, we create new edges from the nodes to all nodes to which an edge goes in from the target node. The sign of each new edge is the same as that of the path from the start node of the new edge to the end node of the new edge through the target node in the original graph.” [22]

In Figures 3.20(A) and (B) from [22], one can observe two edges *going to* the target node from nodes 1 and 5 and three edges *coming from* the target node to the nodes 1, 6, and 7. The new edges will be from node 1 to nodes 1, 6, and 7 and from node 5 to nodes 1, 6, and 7. In both case the edge from 1 to 1 will be a positive self-feedback loop.

In Figure 3.20(A), the edge between nodes 1 and 2 was positive, therefore the new edges are $1 \rightarrow^+ 1$, $1 \rightarrow^- 6$, $1 \rightarrow^+ 7$. On the other hand, in Figure 3.20(B), the sign of the new edges will be changed because the edge between nodes 1 and 2 was negative.

Figure 3.21 from [22] depicts an example where a four-node network can be reduced to a one-node network with the same reduction procedure. “The original network with four components is reduced step by step to one network with one component and two feedback loops. First, the 4th node is removed and edges e_{43} and e_{14} are merged. Then, the 2nd and the 3rd nodes are removed in turn. Finally, we obtain a network with only the 1st node and two positive self-feedback loops.

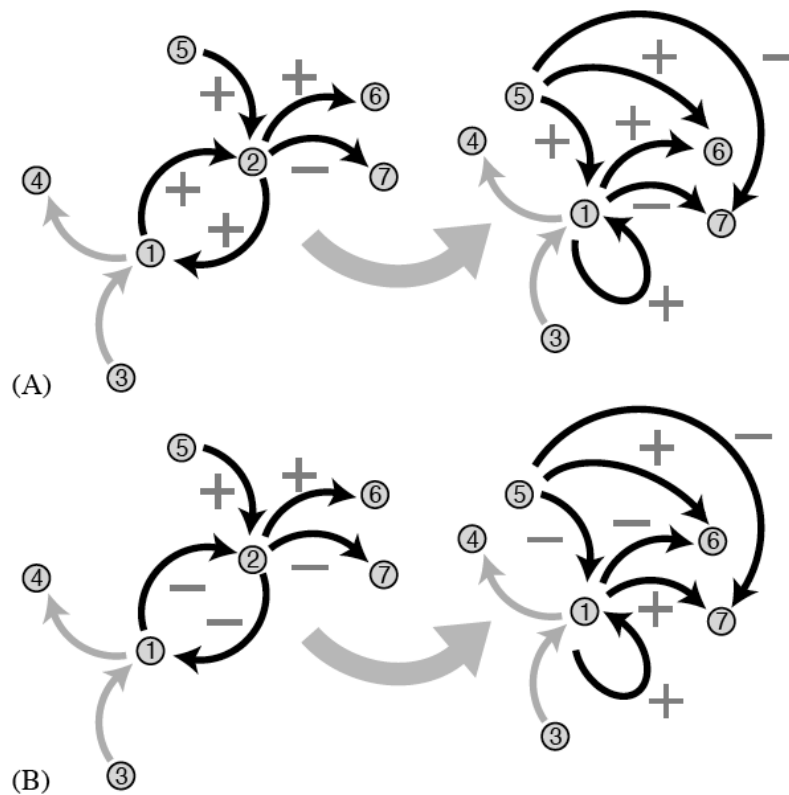


FIGURE 3.20: A reduction procedure (A) positive loop with two positive edges (B) positive loop with two negative edges

3.4.2 Strongly connected components

Another technique consists in representing a feedback loop by a node and the influence of a loop on another by an edge. The system with n loops becomes an acyclic graph where each node is a SCC.

"A directed graph is called strongly connected if there is a path from each vertex in the graph to every other vertex. In particular, this means paths in each direction; a path from a to b and also a path from b to a .

The strongly connected components (SCC) of a directed graph G are its maximal strongly connected subgraphs. If each strongly connected component is contracted to a single vertex, the resulting graph is a directed acyclic graph, the condensation of G ".[23]

In most cases each feedback system can be reduced to one node. A need is to relax the hypothesis is therefore perceived: several SCCs may exist when systems are quasi independent.

Here is a list of examples with multiple SCCs:

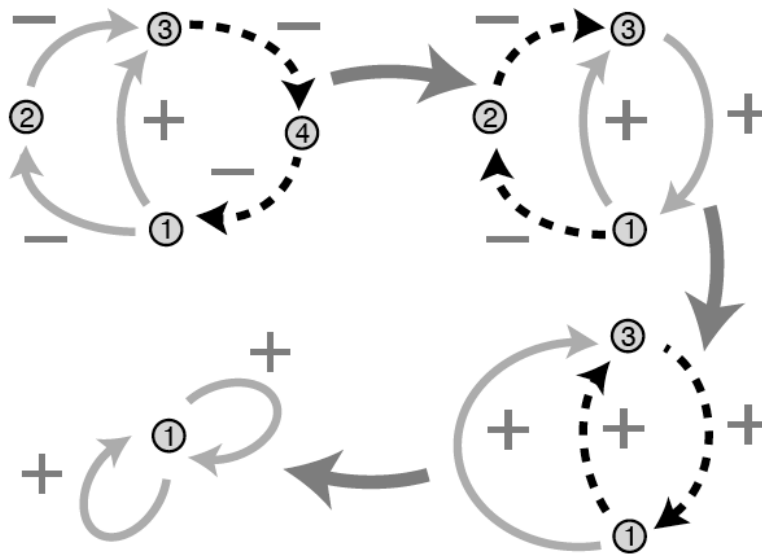


FIGURE 3.21: Example of reduction from a four-component network to a single-component network

- $S1 \leftrightarrow S2$: a driver in his/her car.
- $S1 \rightarrow S2$: the atmospheric system that acts on a human being.
- $S1 \rightarrow S2$: the sun that acts on the Earth.
- $S1 \rightarrow S2 \rightarrow S3$: the sun that acts on the Earth that acts on a human being.
- $S1 \rightarrow S2 \leftrightarrow S3$: the atmospheric system that acts on a car that contains a human being.
- $S1 \rightarrow S2 \leftrightarrow S3$: the atmospheric system that acts on a building that contains a human being.
- $S1 \rightarrow S2 \leftrightarrow \{S3, S4\}$: the atmospheric system that acts on a building that contains two humans beings in the same room.
- ...

Chapter 4

Feedback systems - global behaviour patterns

Introduction

This chapter will introduce a certain number of examples of *feedback patterns*, which also known as *feedback archetypes*. These archetypes are used in *System thinking* to solve problems in organisations. That is, system experts use them to find problems in a organisation and to create strategies to solve these problems. They use these archetypes as a kind of debugging system. Several patterns presented in this chapter come from a book called “The Fifth Discipline, Fieldbook” [19] written by System Thinkers who present strategies and tools for building a learning organisation.

4.1 Success to the Successful

When two positive feedback loops act like a bigger reinforcing loop, the “Success to the Successful” pattern takes place (see Figure 4.1). The Allocation to A instead of B results in more Resources to A. More Resources to A enhances the Success of A which in turn enhances the perception that there should be an Allocation to A Instead of B. With the increase of Allocation to A instead of B, fewer Resources to B are allocated. Fewer Resources to B impedes the Success of B which further reinforces the perception that there should be an Allocation to A instead of B.

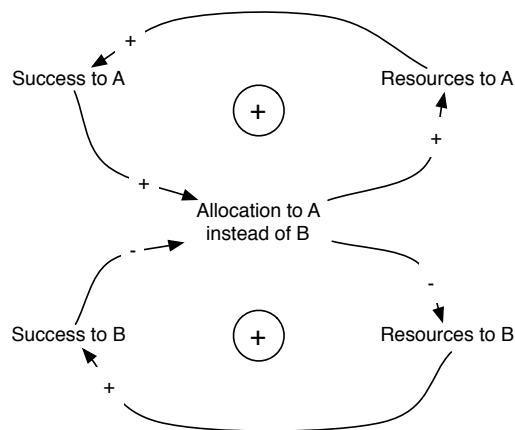


FIGURE 4.1: Success to the Successful: template (CLD)

On comparing the two successes, one can observe that as resources are allocated to A instead of B, the success for A is increased and as resources are diverted from B, the success of B is decreased (see Figure 4.2).

For example, in Figure 4.3 adapted from [24], the success at work and the success with family are compared. At the beginning of the simulation, there is an equal allocation of time to both work and family. But progressively, as more and more time is spent at work instead of with the family, the success at work will grow exponentially at the expense of the success with family.

An application of this pattern as mentioned in [25] is to “avoid competency traps”. It is suggested that the success or the failure of an organisation is more probably due to the initial conditions than intrinsic merits. Organisations should unlearn what they know about what they are best and to look for alternatives rather than always focusing on their knowledge.

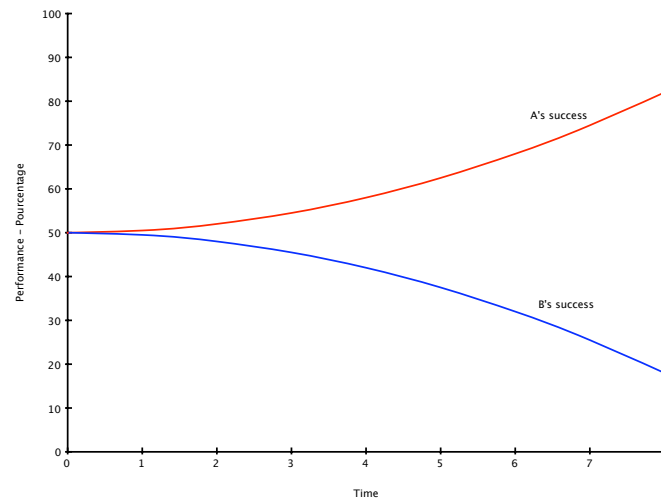


FIGURE 4.2: Success to the Successful: behaviour

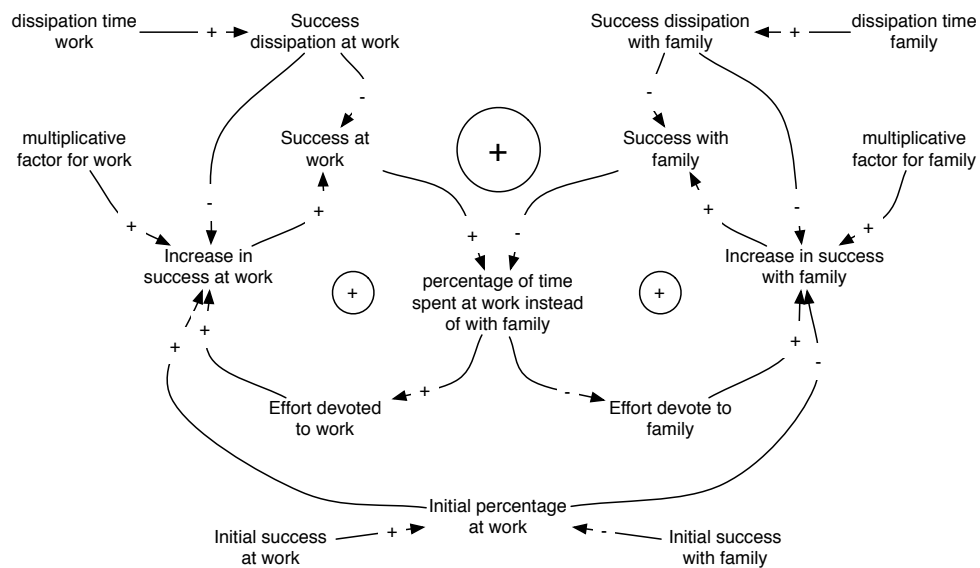


FIGURE 4.3: Success to the Successful: self-fulfilling prophecies (CLD)

4.2 Limits to Growth

"There is growth (sometimes dramatic growth), levelling off or falling into decline." [19]

4.2.1 The S-shaped growth

The S-shaped growth pattern is the first of two patterns that appears in a limited growth environment. After an exponential growth, a certain limit is reached where the growth stabilises to a certain equilibrium forming a plateau in the behaviour picture.

The limited population growth

In the previous chapter, several examples about population were developed. The conclusion was reached that these examples were not very realistic because their stable points were either infinity or zero-population. All growth has to eventually have a limit of some sort. In the population case, we could introduce a capacity limit. Imagine for example, an island with an initial population of 200 people. Food and water are unlimited (not very realistic but keeps the model simple). There exists a time in the future when the population will not grow anymore because of the size of the island (maximum capacity).

The notion of “crowding” can now be defined: population divided by the maximum capacity. Let’s see what happens in this new model depicted in Figure 4.4 taken from [9]. Capacity and crowding are now variables of the model and they form a negative feedback loop with the population and the births. In this example, the death fraction is still a constant but the birth fraction is now a function of crowding (the higher the crowding, the lower the birth fraction). What happens now is that a higher population means a higher crowding which decreases the birth fraction, which in turn has a negative effect on the population via births. The effect on population is indeed negative because of the two positive effects $\text{Birth fraction} \rightarrow^+ \text{Births} \rightarrow^+ \text{Population}$. The birth fraction is decreased, the same effect will therefore be applied on births and then population.

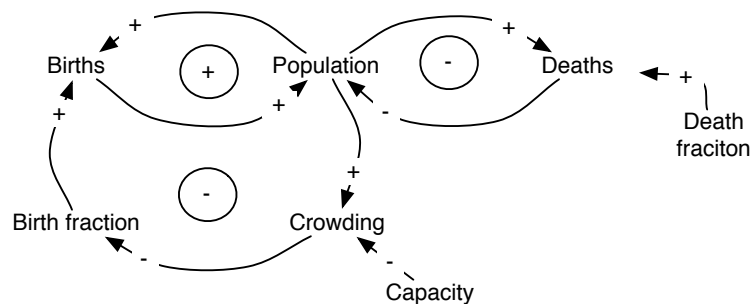


FIGURE 4.4: Population model: density-dependent growth (CLD)

The translation between the causal loop (Figure 4.4) and the stock-and-flow (Figure 4.5) diagram is direct. We can see that the population acts like a stock and that births and deaths are in and out flows respectively.

The model was tested with different configurations. A maximum capacity of 200 people with a death fraction of 0.6. One can observe three possible cases in the dynamics of this system (see Figure 4.6).

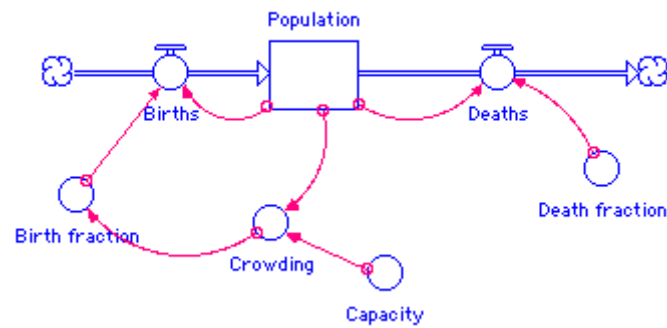


FIGURE 4.5: Population model: density-dependent growth (SFD)

1. Initial population greater than 200: the crowding fraction is greater than 1 meaning that the birth fraction is lower than the death fraction, which in turn means that the population will decay progressively to 200. Here, the initial population is of 400 people (red line).
2. Initial population of 100: the crowding fraction is lower than 1 meaning that the birth fraction is greater than the death fraction. The population will progressively grow to 200 - goal-seeking phase (blue line).
3. Initial population of 10: the crowding fraction is near 0 meaning that the population will grow exponentially. But after a certain time, it will grow progressively and no longer exponentially, until the 200-people limit is reached. This is the effect of the non-linear negative loop (green line).

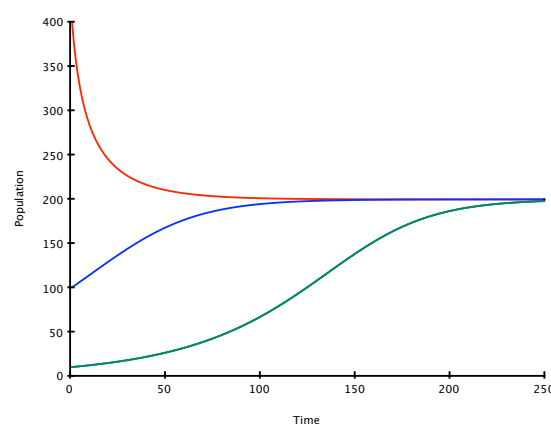


FIGURE 4.6: Population model: density-dependent growth (Behaviour)

This last case is thus the more interesting of the three because the dynamics of the population growth look like an S, which indicates that there are two phases. The first phase is an

exponential growth phase following by a goal-seeking phase as a certain limit is approached. This behaviour is caused by the shift in loop dominance from the simple positive feedback loop to the negative density dependence loop.

Word of mouth

In the previous chapter, the loop dominance shift was explained with an example of the introduction of a new product on a market[10]. It was concluded that, depending on the dominant loop in the system, the growth is more or less fast. This is exactly the behaviour we observed in the S-shaped growth.

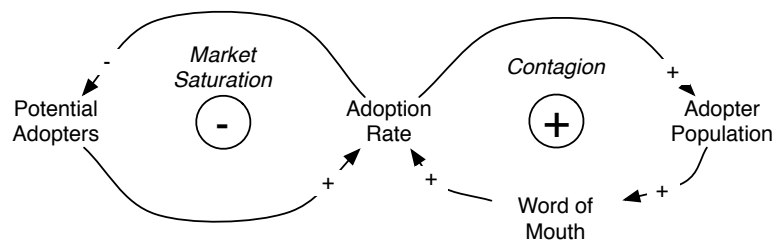


FIGURE 4.7: New product on a market: S-shaped growth (CLD)

First the “Contagion” loop – positive – is dominant and creates an exponential growth until a certain point is reached, when the second phase starts. This second phase consists in a goal-seeking growth where the “market saturation” loop – negative – leads the growth progressively to its limit.

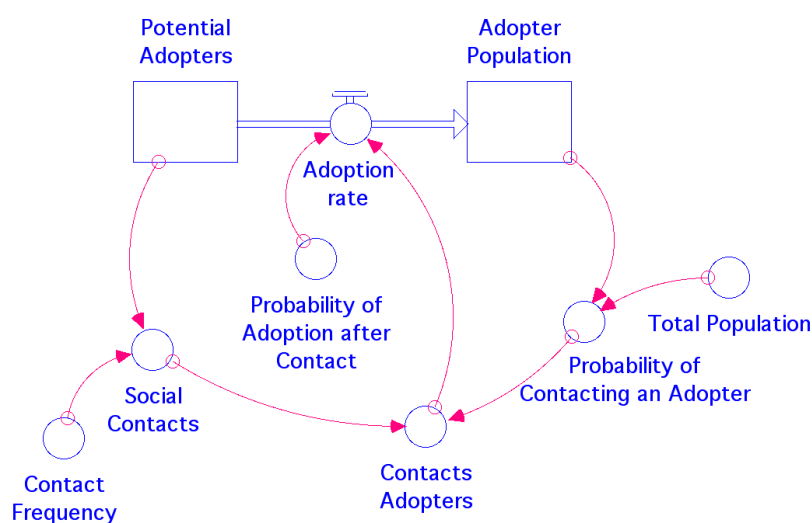


FIGURE 4.8: New product on a market: S-shaped growth (SFD)

Figure 4.8 depicts the stock-and-flow diagram where two stocks are represented: one for the potential adopters and one for the adopter population. In this case, there is no source nor sink. So when all of the initial potential adopters flow from their stock to the one for the adopter population, the system has no longer a reason to exist. The limit of the system lies in the level of the potential adopters.

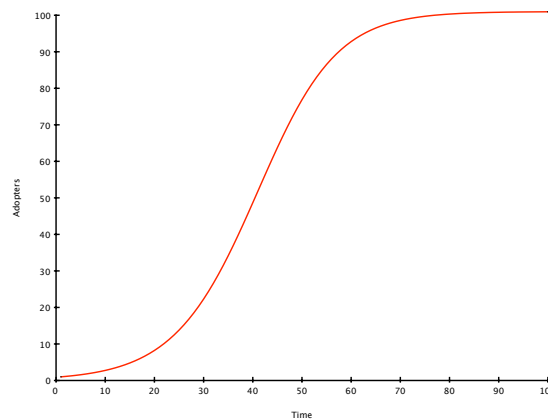


FIGURE 4.9: New product on a market: S-shaped growth (Behaviour)

Figure 4.9 shows that the adopter population grows like the S-shaped pattern. If the state of the potential adopters had been plotted for the corresponding situation, we would have observed the opposite behaviour. As already mentioned before, there are no sources or sinks for the stocks. Thus, all the potential adopters flow to the adopter population.

4.2.2 Overshoot-and-decline

The overshoot-and-decline pattern is the second pattern encountered in the limited growth environment. After the exponential growth, variables have gone over their natural constraint and have crashed completely.

Population limited growth

In the previous section, an example where the limiting variable was actually a constant was described. The interesting case was the S-shaped growth where the population increases in two phases: the first one is an exponential growth and the second one was a goal-seeking growth until the capacity is reached. Now, a system where the limiting factor is itself a variable depleted by the population, will be analysed. Also in this example, the limiting factor is not on the positive feedback loop but on the negative one. The population will not be limited in the growth but will decrease more or less rapidly depending on the food.

The *overshoot* pattern is as follows: the population can grow so rapidly that it shoots past its limit and its size is no longer sustainable. The decrease in population is thus a simple adjustment that is necessary to drive the system to a stable state. One has to keep in mind here that the population is not limited by a maximum capacity but by the food factor.

One consequence of this overshoot can be observed on the food level: the population grew so rapidly that it consumed all the available resources, which in turn creates a delayed reaction on the population. People that were over the limit disappear but also cause more death due to the extra consumption of food. That is the *decline* behaviour.

The loop Population \rightarrow^- Food per capita \rightarrow^- Death Fraction \rightarrow^+ Deaths \rightarrow^- Population plays the role of the density-dependent limit. So what will happen? Firstly, the population will grow exponentially until it reaches a certain point where the consumption rate is so high that the stock of food collapses. Secondly, due to this high consumption, starvation will appear and the death rate will increase causing the population to decay until it reaches the complete annihilation of the human specie.

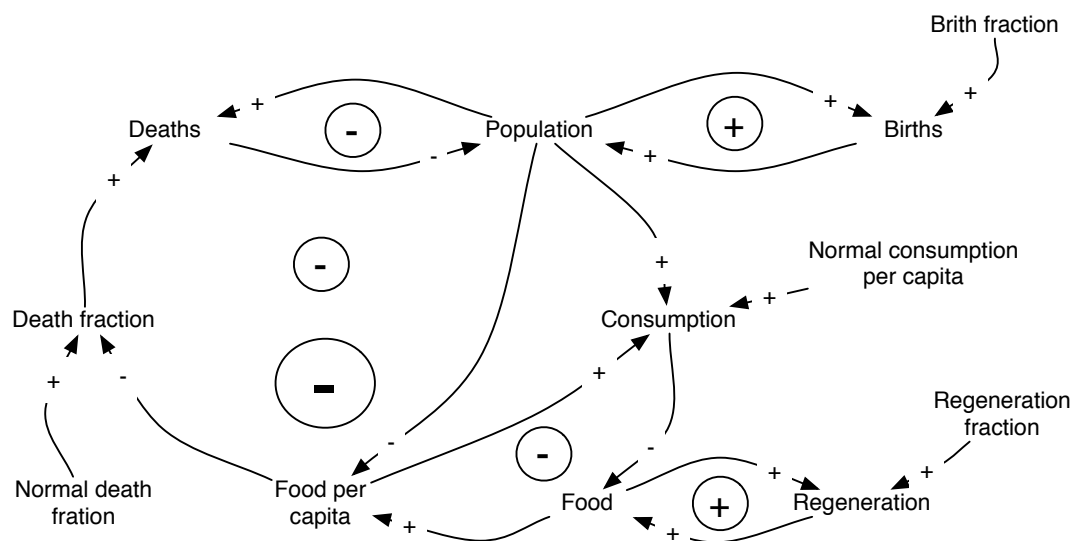


FIGURE 4.10: Population model: a population-food interaction model (CLD)

The transition between the qualitative and the quantitative diagram is not straightforward. Two new variables appear in Figure 4.11 compared to Figure 4.10: Effect of food on consumption and Effect of food on df. These variables are implicit in the arrow joining the Food per capita and Consumption on one side, and Food per capita & Death fraction on the other side. These are also the non-linear factors of the system.

In Figure 4.12 taken from [9], one can observe the evolution of the two stock variables that are Population and Food. Thanks to a certain level of food, the population can begin its

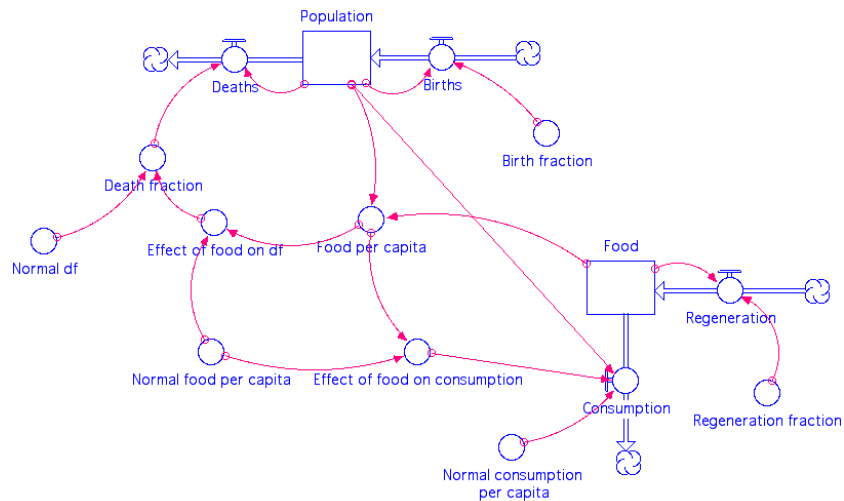


FIGURE 4.11: Population model: a population-food interaction model (SFD)

exponential growth and create more food. But, after a certain point, the food available is not enough and the consumption is much more important than the regeneration, causing the food to fall down to zero. Just after this point, starvation begins to appear and more and more people die until the population reaches its minimal capacity of zero.

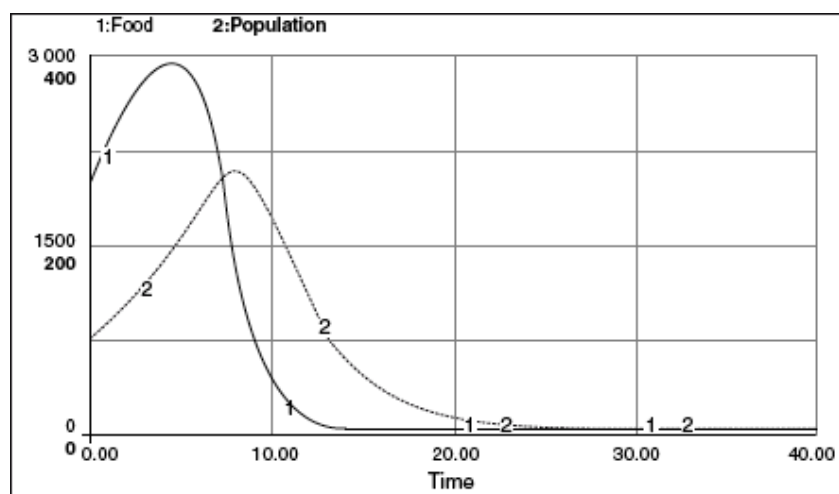


FIGURE 4.12: Population model: a population-food interaction model (Behaviour)

Epidemic dynamics

Here is another example of the overshoot-and-decline pattern: the epidemic dynamics.[9] Let us suppose that there is a contagious disease which means that when two people meet (one susceptible and one infected), there is a chance that the infected person will infect the susceptible person.

In this model, a constant flow of incoming people is used. That is, in the global population, each month, a constant number of people are susceptible to contract the disease. Also, the removal factor contains the notion of infected people who have died and infected people who have been cured. Let us also suppose that people who have recovered from the disease are now permanently immunized against this disease and will not return directly into the group of susceptible people.

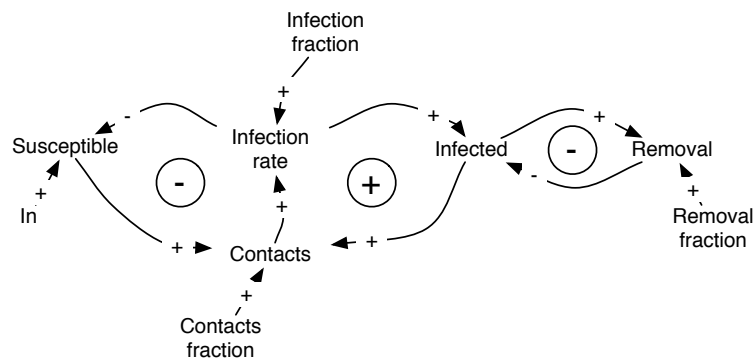


FIGURE 4.13: Epidemic dynamics (CLD)

Figure 4.14 is the direct translation of Figure 4.13. With an initial set of 500 susceptible people and 10 infected ones, and a contact fraction of 1 %, an infection rate of 1 %, a removal fraction of 1 % and a constant inflow of 1 person per month in the pool of susceptible people, the following behaviour depicted in Figure 4.15 was observed.

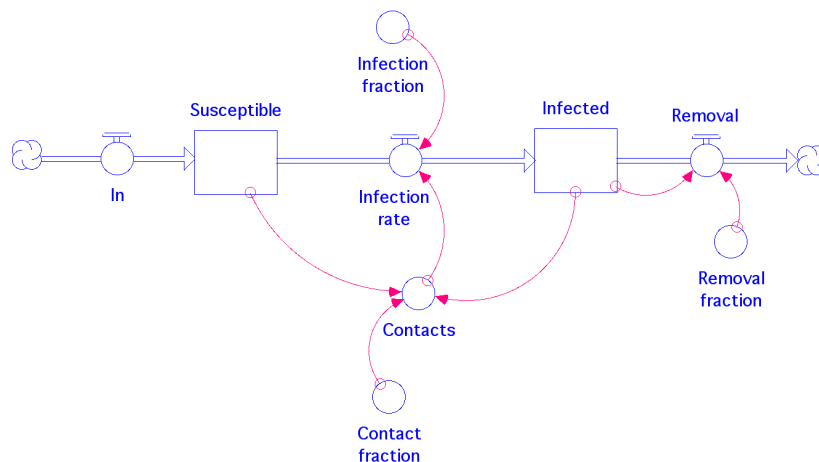


FIGURE 4.14: Epidemic dynamics (SFD)

We observe that the pool of susceptible people is flowing into the pool of infected people. But, because the resource for the disease is decreasing dramatically, the number of infected people grows exponentially at first. But when the susceptible people are reduced to a handful

of individuals, then the disease disappears progressively either because the infected people recover or because they die.

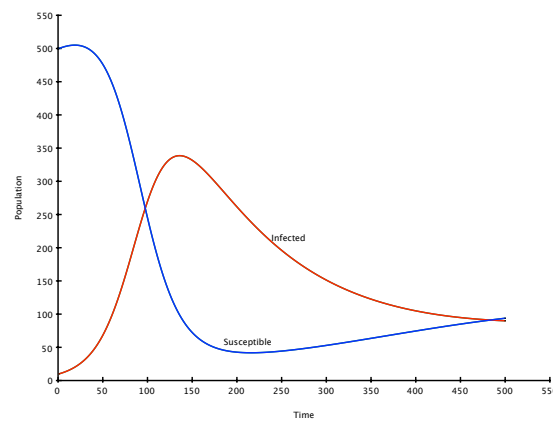


FIGURE 4.15: Epidemic dynamics (Behaviour)

Something interesting emerged from this model when different values are assigned to variables. For example, if we change the infection rate to 2%, the contact fraction and the removal fraction to 10%, oscillations in the dynamics of epidemics were observed (see Figure 4.16).

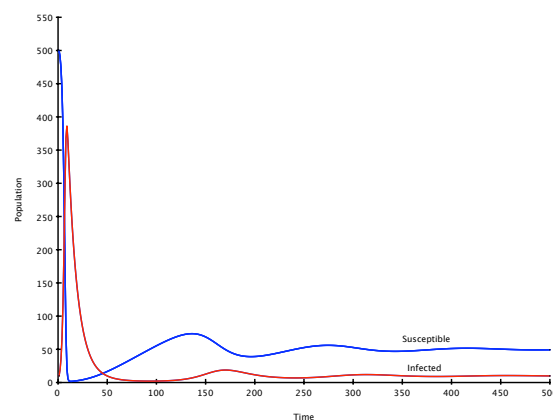


FIGURE 4.16: Epidemic dynamics - oscillation

In this case, the growth of the infected population is steeper than in the other case because the contact percentage is greater. But when the susceptible population dramatically falls down to zero, which leads the infected population in the same direction. But during the disappearance of ill people, the constant flow of new susceptible people makes this population grow slowly. With a certain delay, the infected population will grow progressively but not as high as the susceptible one. This oscillation movement will vanish after a certain lapse of time when the two pools will stabilize.

4.3 Tragedy of the Commons

'The Tragedy of the Commons always opens with people benefiting individually by sharing a common resource. But at some point, the amount of activity grows too large for the *commons* to support. In many cases, the commons seems immeasurably large and bountiful at first, but is either non renewable or takes a great deal of time and effort to replenish.'" [19]

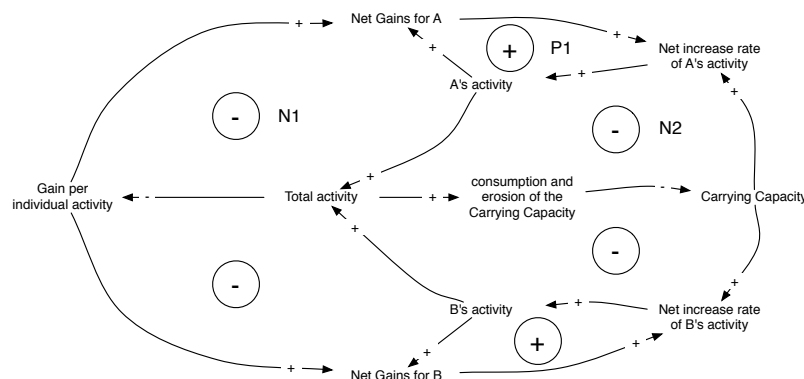


FIGURE 4.17: Tragedy of the Commons: All for One & None for All (CLD)

For example, in Figure 4.17, the Carrying Capacity is the common resource between the two populations (A and B). The loop P1 for A is a positive feedback loop that generates growth thanks to the common resource. The loop N1 is a negative feedback loop that limits the gain of each individual due to the natural limit of the shared resource. The behaviour of the pair (P1,N1) was described in the "The S-shaped growth" pattern where the population A would stabilise to an equilibrium. Unfortunately for them, a third loop N2, also negative, provokes the erosion of the common resource. The tragedy point of view comes from the *crash dynamic*, that is, the erosion of the shared resource means that it becomes unable to regenerate itself.

In the case studied here, even if the two demands on the common resource are increasing (see Figure 4.18), a certain declining behaviour may appear. The populations will increase their demand, because they do not understand why their previous demands are not being met, until the common good collapses.

There is a link between this pattern and the "Limits to Growth" pattern: the "Tragedy of the Commons" consists of multiple "Limits to Growth" patterns sharing a common constraint or finite limit. In fact, the collapse in this pattern is steeper than the one in the overshoot-and-decline pattern due to the fact that there are more than one growth loops – positive feedback loops.[19]

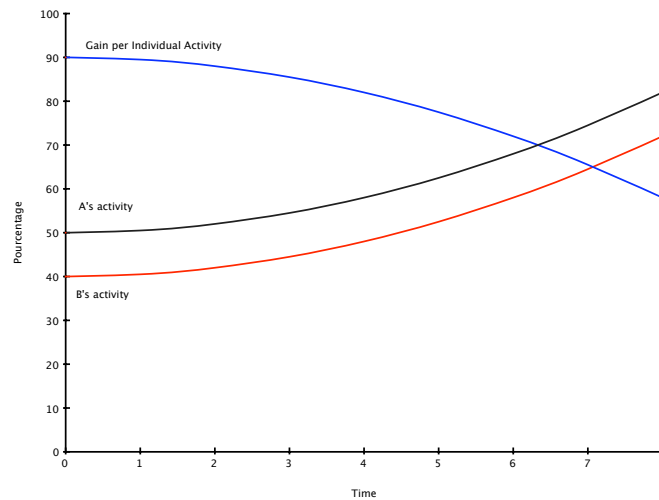


FIGURE 4.18: Tragedy of the Commons: behaviour

4.4 The Attractiveness Principle

The “Attractiveness Principle” consists in making choices. This pattern is essentially the “Limits to Growth” pattern with multiple limits that can not be addressed equally.

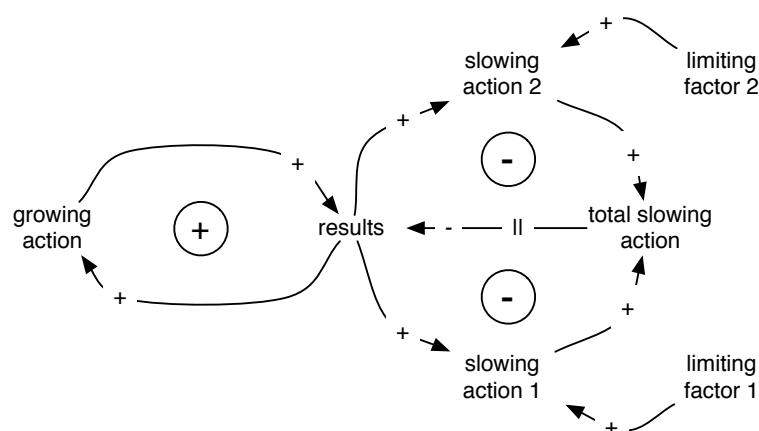


FIGURE 4.19: Attraction Principle: template (CLD)

In Figure 4.19 taken from [26], one can observe a positive feedback loop that represents the growth of the system and produces results that tend to create more growth actions. These results are in two negative feedback loops that limit the growth action by slowing down the growth process. The two limiting factors are different but their corresponding slowing actions are combined before withdrawing any results from the results variables.

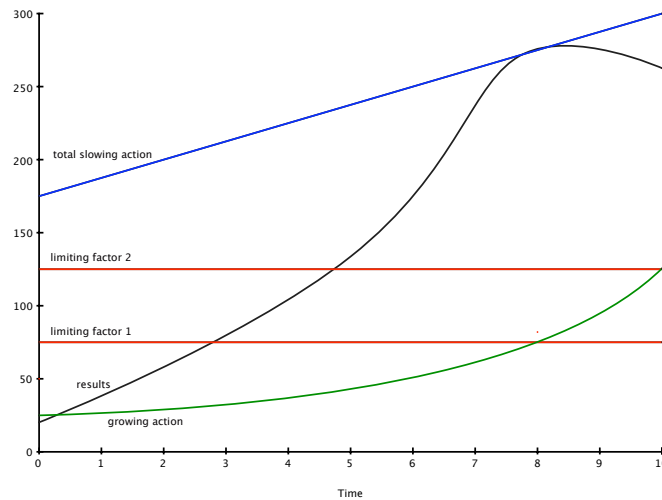


FIGURE 4.20: Attraction Principle: behaviour

One can observe in Figure 4.20 that the performance (results in the CLD) is increased with a small amount of effort (growth action) but, due to the slowing action which is the accumulation of the two slowing actions, the performance will stop its growth and start declining even if the efforts are still increasing.

4.5 Growth & Under Investment

The pattern “Growth & Under Investment” is a variation of the “Limits to Growth” pattern, but in this case the limiting factor is in a negative feedback loop that contains a delay and an external standard. In fact, the two negative feedback loops form a positive feedback loop which inhibits the growth (see Figure 4.21 taken from [26]).

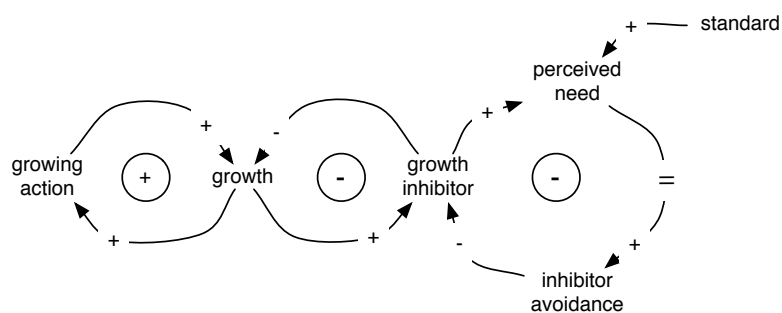


FIGURE 4.21: Growth & Under Investment: template (CLD)

The growth action increases the growth that simply influences positively the same growing action producing the characteristic exponential growth. But, as it was said in the “Limits

to Growth”, nothing can grow forever. The increase in the growth influences the growth inhibitor that decreases the growth.

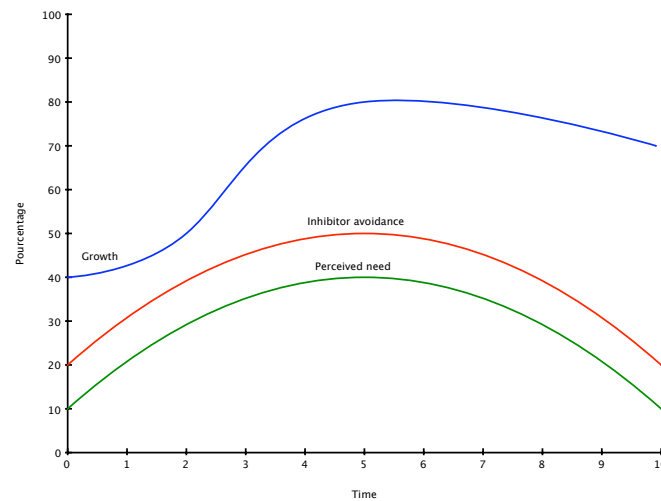


FIGURE 4.22: Growth & Under Investment: behaviour

If the growth inhibitor is reduced then the growth can increase. As this inhibitor interacts with a standard, a perceived need emerges and a movement to avoid this inhibitor is created. This inhibitor avoidance will decrease the inhibitor which in turn will increase the growth (see Figure 4.22). But the delay in this inhibitor avoidance loop is disturbing the need to avoid the inhibitor: the inhibitor slows down the growth which in turn decreases the inhibitor. This decrease in the growth inhibitor will also decrease the perceived need. In this configuration, the growth is limited because the perceived need is undermined by the system’s own actions and, by the time the system realises that the inhibitor avoidance is needed, the need is already decreasing and the growth levels off at a lower level than it could have reached.

4.6 Balancing with Delay

The pattern “Balancing with Delay” is a derivation of a negative feedback loop where delay is added. This delay is responsible for oscillation in the behaviour of this pattern.

When someone wants to have a shower, he/she has a desired water temperature. The first drops of water coming out of the shower handle are therefore either too hot (if the hot water is turned on first) or too cold (if the cold water is turned on first). Depending on the desired temperature, this person will increase the hot water or the cold water. But, in general, the change is too important and, for example, if the water was too hot, it is now slightly cold. So the water will be progressively adjusted until the desired temperature is reached. The behaviour of the temperature will be characterised by damped oscillations (see Figure 4.24).

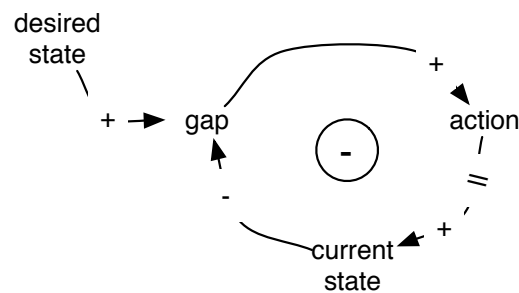


FIGURE 4.23: Balancing with Delay: template (CLD)

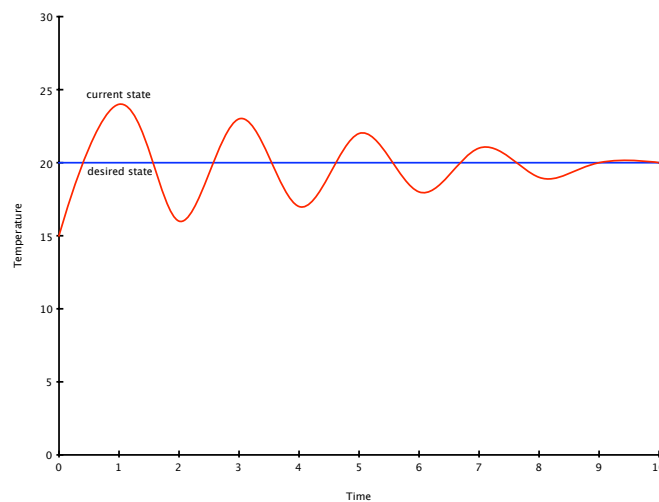


FIGURE 4.24: Balancing with Delay: behaviour

4.7 Escalation

When two balancing loops (negative feedback loops) act like one big reinforcing loop (positive feedback loop), the “Escalation” pattern takes place (see Figure 4.25). An increase in the Results of A relative to B influences more Action by B. An increase in Action by B enhances B’s Results. As B’s Results increase, it tends to reduce the Results of A relative to B. This reduction implies more Action by A. Additional Action by A increases A’s Results. The increase in A’s Results then increases the Results of A relative to B, and the cycle then repeats itself.

In Figure 4.26, one can observe the simple behaviour of the Escalation pattern where the two growths are parallel. Because of the increase of the actions taken by A and interpreted as a threat for B, B takes the same actions. One thing that is not expressed by the behaviour analysis is the possibility of collapse if the escalation goes to high.

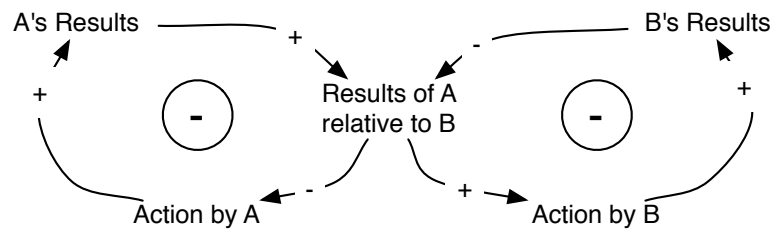


FIGURE 4.25: Escalation: template (CLD)

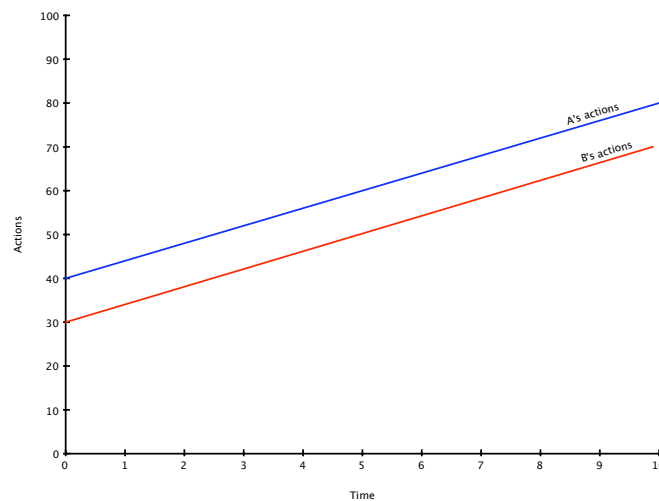


FIGURE 4.26: Escalation: behaviour

For example, in Figure 4.27 adapted from [24], x and y are compared. When x takes actions, buying guns or missiles for examples, y perceives these actions as a threat and takes the same actions. The combined infinity-shaped loops form an exponential tension between the two parties (see Figure 4.27).

De-escalation is also possible, if one of the two parties decides to step back. In this case, one can observe a slightly delayed reaction where the other party will also step back and stop the escalation of tension (see Figure 4.28 taken from [24]).

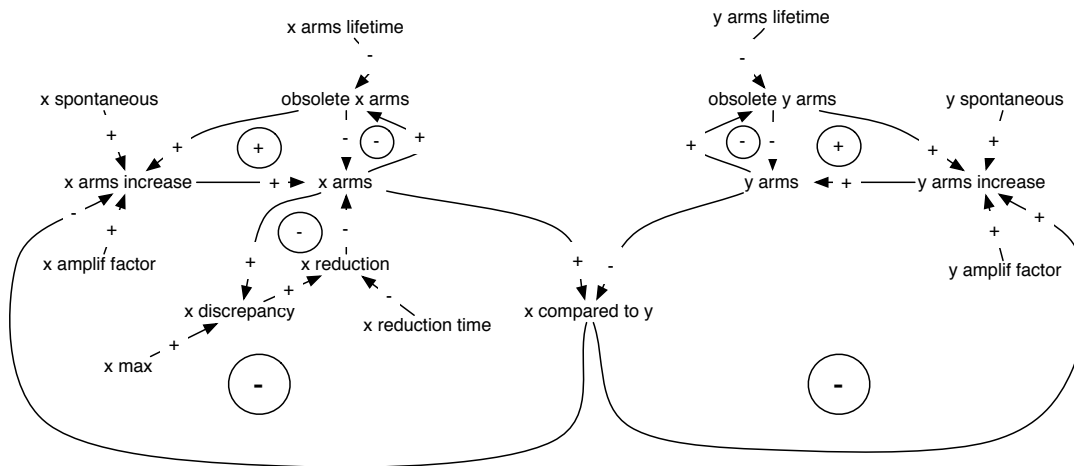


FIGURE 4.27: Escalation: the arms race (CLD)

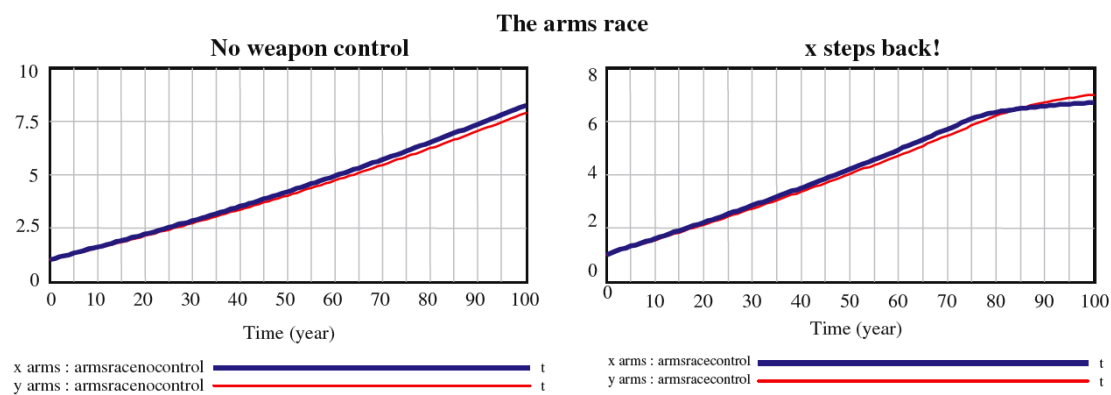


FIGURE 4.28: Escalation: the arms race (behaviour)

4.8 Indecision

The “Indecision” pattern consists of two negative feedback loops with delays (see Figure 4.29). Due to the delays, when a value arrives at its destination, this will already have changed therefore the value changes again. Here is an example taken from [26]:

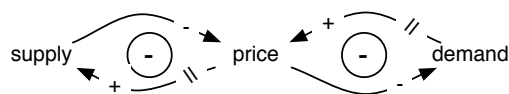


FIGURE 4.29: Indecision: template (CLD)

In a market, the price is fixed by the supply and the demand. Therefore, if the demand increases, the price will rise, this rise will have a direct effect on the demand and a long-term

effect on supply. That is, the demand will slightly decrease and the supply will increase (the companies will produce more according to the increase in demand).

But, as the supply increases, the price will decrease due to the direct effect that supply has on price. But at the same time the demand that went through its delay, adds less to the price. The supply being more than appropriate for the demand, the price decreases. This decrease will increase the demand and after some time will decrease the supply.

In this way, by the time the effect of the increase on the demand reaches the price, the supply will have decreased. The supply and the demand will once more be out of sync.

Because of the delays, the supply and the demand will never be synchronised and the price will oscillate endlessly.

4.9 Fixes That Backfire

The pattern “Fixes That Backfire” can be explained by a simple example: take the case of a person who knows next to nothing anything about mechanics and hears a strange noise coming from the wheel of his/her bike. This person, who has been told that, to repair a squeaky wheel he/she needs oil, will pick up a can of water instead and put it on the wheel. The noise will disappear but only for a brief lapse of time. It will then return more loudly than before, air and water combining to rust the joint of the wheel. Thus the person grabs another can of water and splashes it on the wheel again, making the noise disappear once again for a short time. Eventually, the noise will come up again more loudly and will quickly be fixed by water until the wheel stops turning.

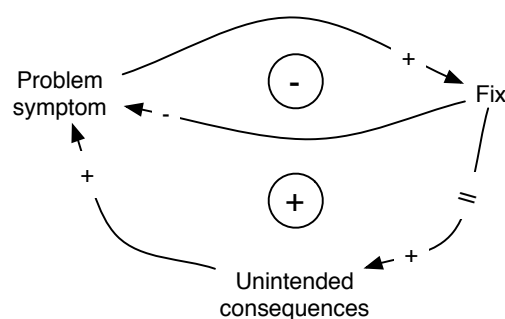


FIGURE 4.30: Fixes that backfire: template (CLD)

"The central theme of this archetype is that almost any decision carries long-term *and* short-term consequences, and the two are often diametrically opposed. As shown in Figure 4.30, the problem symptom cries out (squeaks) for resolution. A solution is quickly implemented (the fix - water) which alleviates the symptom (in the balancing loop). *But the*

unintended consequences of the fix (the vicious cycle of the reinforcing loop) actually worsen the performance or condition which we are attempting to correct.” [19]

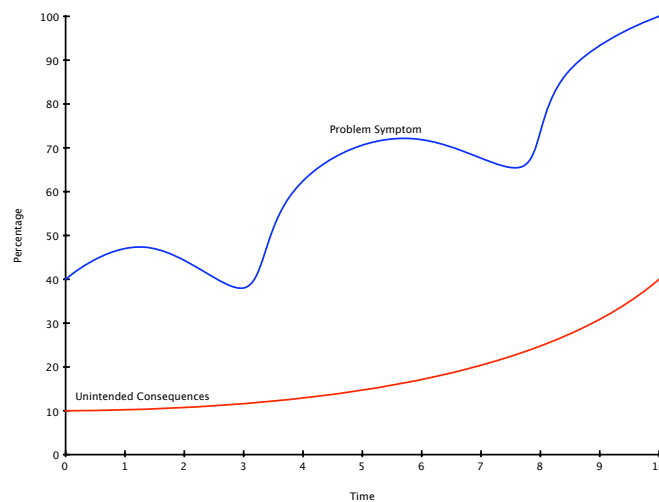


FIGURE 4.31: Fixes that backfire: behaviour

In Figure 4.31, the general behaviour of this pattern is depicted. The problem symptom increases continuously while the fixes that are applied reduce the problem for a short time, thus creating gaps in the growth of the problem. "A problem symptom alternatively improves (the problem variable goes down) and deteriorates (the problem goes up, worse than before)." [19]

4.10 Accidental Adversaries

When groups of people decide to work together on a project that they want to build together, the end result may not be as nice as they might have thought. The local activity used to improve the partnership, even with the best intentions, can limit the overall development of the alliance and this will in turn decrease the local activity as well.

The "Accidental Adversaries" pattern consist of four positive feedback loops and two negative feedback loops [26]:

- $A's\ success \rightarrow^+ A's\ activity\ toward\ B \rightarrow^+ B's\ success \rightarrow^+ B's\ activity\ toward\ A \rightarrow^+$
 $A's\ success$: represents a co-operative reinforcing loop between A and B.
- $A's\ success \rightarrow^+ A's\ activity\ toward\ A \rightarrow^+ A's\ success$: A taking actions to enhance its success.
- $B's\ success \rightarrow^+ B's\ activity\ toward\ B \rightarrow^+ B's\ success$: B taking actions to enhance its success.

- $A's\ success \rightarrow^+ A's\ activity\ toward\ B \rightarrow^+ B's\ success \rightarrow^+ B's\ activity\ toward\ B \rightarrow^- A's\ success \rightarrow^+ A's\ activity\ toward\ A \rightarrow^- B's\ success \rightarrow^+ B's\ activity\ toward\ A \rightarrow^+ A's\ success$: the overall system growth driven by this global reinforcing loop.
- $A's\ activity \rightarrow^+ A's\ activity\ toward\ A \rightarrow^- B's\ success \rightarrow^+ B's\ activity\ toward\ A \rightarrow^+ A's\ success$: an increase in A's activity means a decrease in B's activity which in turn mean a decrease in A's activity.
- $B's\ activity \rightarrow^+ B's\ activity\ toward\ B \rightarrow^- A's\ success \rightarrow^+ A's\ activity\ toward\ B \rightarrow^+ B's\ success$: an increase in B's activity means a decrease in A's activity which in turn means a decrease in B's activity.

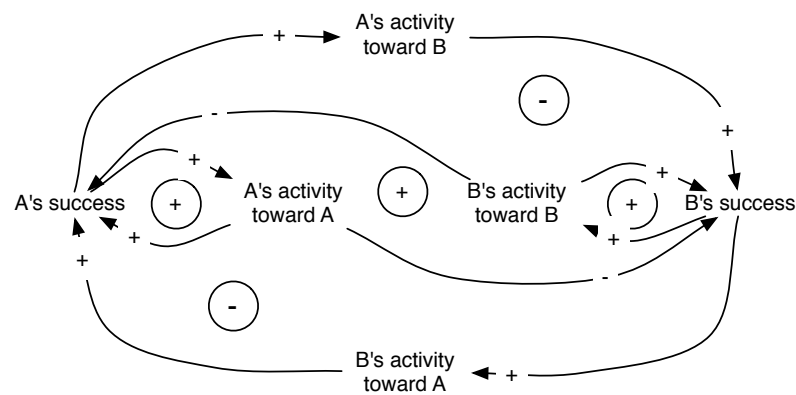


FIGURE 4.32: Accidental Adversaries: template (CLD)

This typical collaboration pattern shows in Figure 4.33 that the partnership which is normally beneficial for both parties due to the synergy of their actions, can also be detrimental to them. If one of the two parties misunderstands the action of the other party, suspicion or mistrust may erode the relationship. If none of the parties takes hindsight and thinks about the reason of the actions taken by the other partner, they may lose the benefit of their "contract". Thus, one will drag its new adversary into the spiral of losing impact on each other.

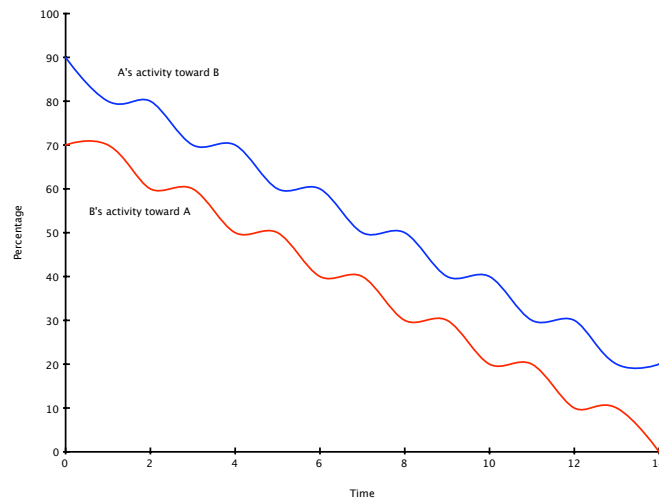


FIGURE 4.33: Accidental Adversaries: behaviour

4.11 Shifting the Burden

The pattern “Shifting the Burden” begins like the pattern “Fix That Backfire” with a problem symptom that needs to be solved. Easy and immediate solutions are rapidly found and applied to the problem, thus shifting the attention away from the real roots of the problem.

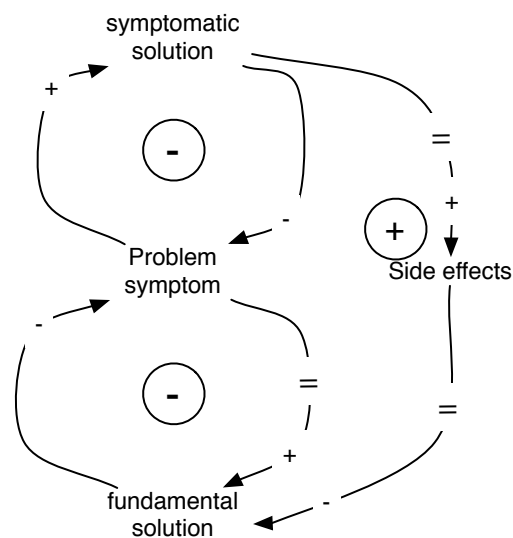


FIGURE 4.34: Shifting the Burden: template (CLD)

“The basic Shifting the Burden template has two reinforcing loops (positive feedback loops). Each represents a different type of fix for the problem symptom. The upper loop [in Figure 4.34] is a symptomatic quick fix; the bottom loop represents measures which take longer (hence the delay) and are often more difficult, but ultimately address the real problem.” [19]

Figure 4.35 depicts the three behaviour patterns that coexist in “Shifting the Burden”:

Symptomatic solution: grows due to the addiction loops.

Problem symptoms: oscillates like in the “Fix That Backfire” pattern.

Fundamental solutions: declines also due to the addiction loops.

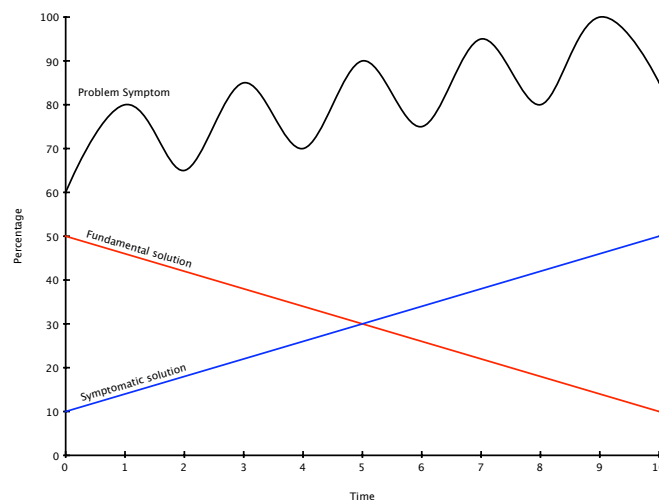


FIGURE 4.35: Shifting the Burden: behaviour

4.12 Addiction

When a problem reoccurs repeatedly and when the “easy” solution is chosen to solve this problem again and again, the “Shifting the Burden” pattern becomes the “Addiction” pattern where the side effects of the symptomatic solutions promotes some sort of addiction which inhibits the fundamental solution.

4.13 Drifting Goals

The Drifting Goals pattern is identified when two balancing loops interact in such a way that the activity of one loop actually influences the intended goal of the other one in a negative way (see Figure 4.36). The Desired State interacts with the Current State to produce a Gap. This Gap influences Action intended to move the Current State in the direction of the Desired State. At the same time, the Gap influences Action and creates a Pressure to Adjust Desire. This pressure essentially acts as an influence to reduce the Desired State. As the Desired State is undermined, it works to reduce the Gap, thus lessening the influence

toward Action. The final result of this structure is that it reaches an equilibrium other than what was the initial Desired State.

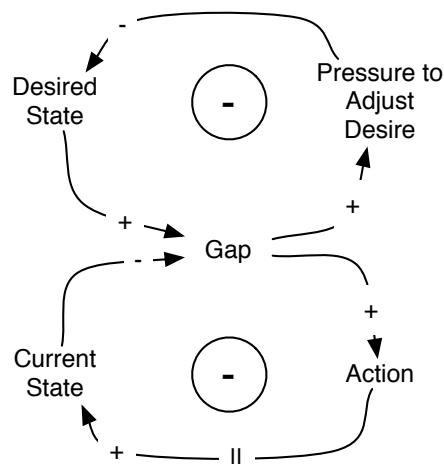


FIGURE 4.36: Drifting goals: template (CLD)

For example, in Figure 4.37 taken from [26], a company situation is depicted where the company fixes a revenue goal at the beginning of year for the end of the year. There is a gap between the revenue goal and the current revenue that will be reduced during the year via the sales. But the revenue at the end of the year appears to be less than the goal revenue. For the following year, the company will decide to establish a lower revenue goal (via the pressure to adjust the goal) but the revenue at the end of the year will still be less than the goal one. The company will continually decrease the revenue. One can observe that one loop is drifting away from the goal of the other loop.

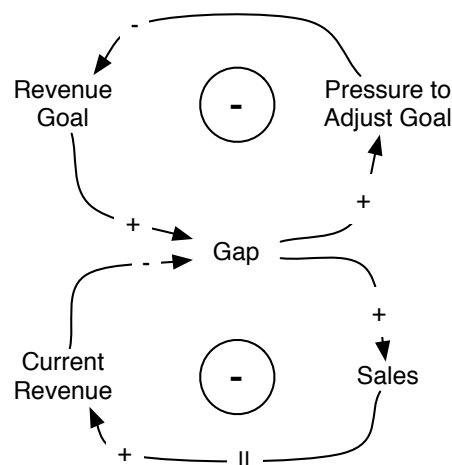


FIGURE 4.37: Drifting goals: Declining Sales Goals (CLD)

Figure 4.38 depicts the behaviour of this pattern Drifting Goals also called Eroding Goals due to the fact that in general, organisations prefer to lower the goal that is not achieved rather than taking corrective actions without changing the target goal. One can observe that the goal is lowered progressively and that the current state (actual) is oscillating downwards below the goal that is never met. The gap between the desired state and the goal is also oscillating.

The reason why the actual state is oscillating downwards below the desired goal is as follows: as the gap decreases between the desired and the actual goals, the actions taken to improve the current state are diminished and the actual state will also decrease. Then, when the desired goal decreases, the gap will also decrease and the actual state will increase but, due to the delay between the actions taken and the actual state, the behaviour of this state is an oscillatory one.

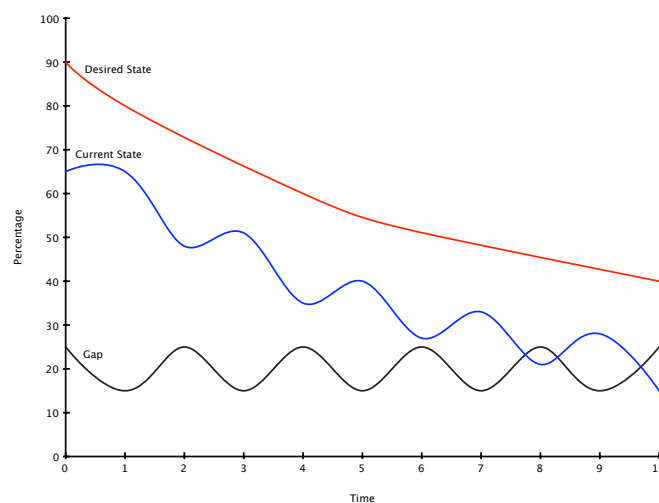


FIGURE 4.38: Drifting goals: behaviour

4.14 Growth & Under Investment with a Drifting Standard

The “Growth& Under Investment with a Drifting Standard” pattern is the same as the “Growth & Under Investment” pattern, the only difference being that in the first case the growth inhibitor induces a decline of the standard over time (see Figure 4.39 taken from [26]).

The explanation is exactly the same as for the “Growth & Under Investment” pattern, plus the additional influence where the inhibit growth subtracts from the standard implies an even less perceived need for the inhibitor avoidance.

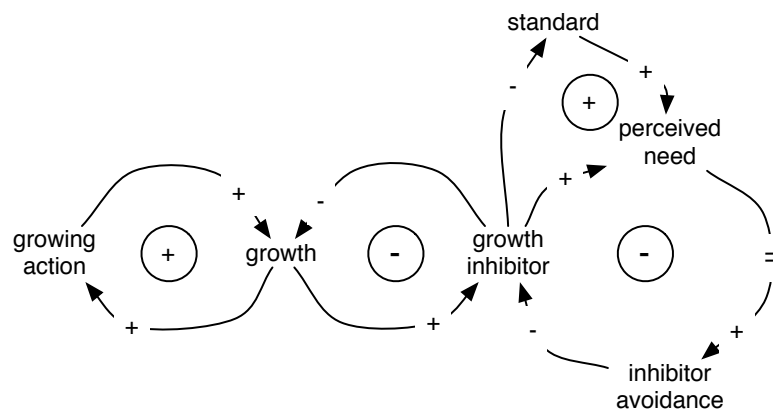


FIGURE 4.39: Growth & Under Investment with a Drifting Standard: template (CLD)

4.15 The Archetype Family Tree

All the patterns or archetypes that are described in this chapter are in fact related to each other. In general, moving down the tree is adding a loop or combining two patterns. Figure 4.40 summarises the situation where each link between patterns is described by a sentence. This tree was first developed in [19] but was modified to take into account new patterns also defined in this chapter.

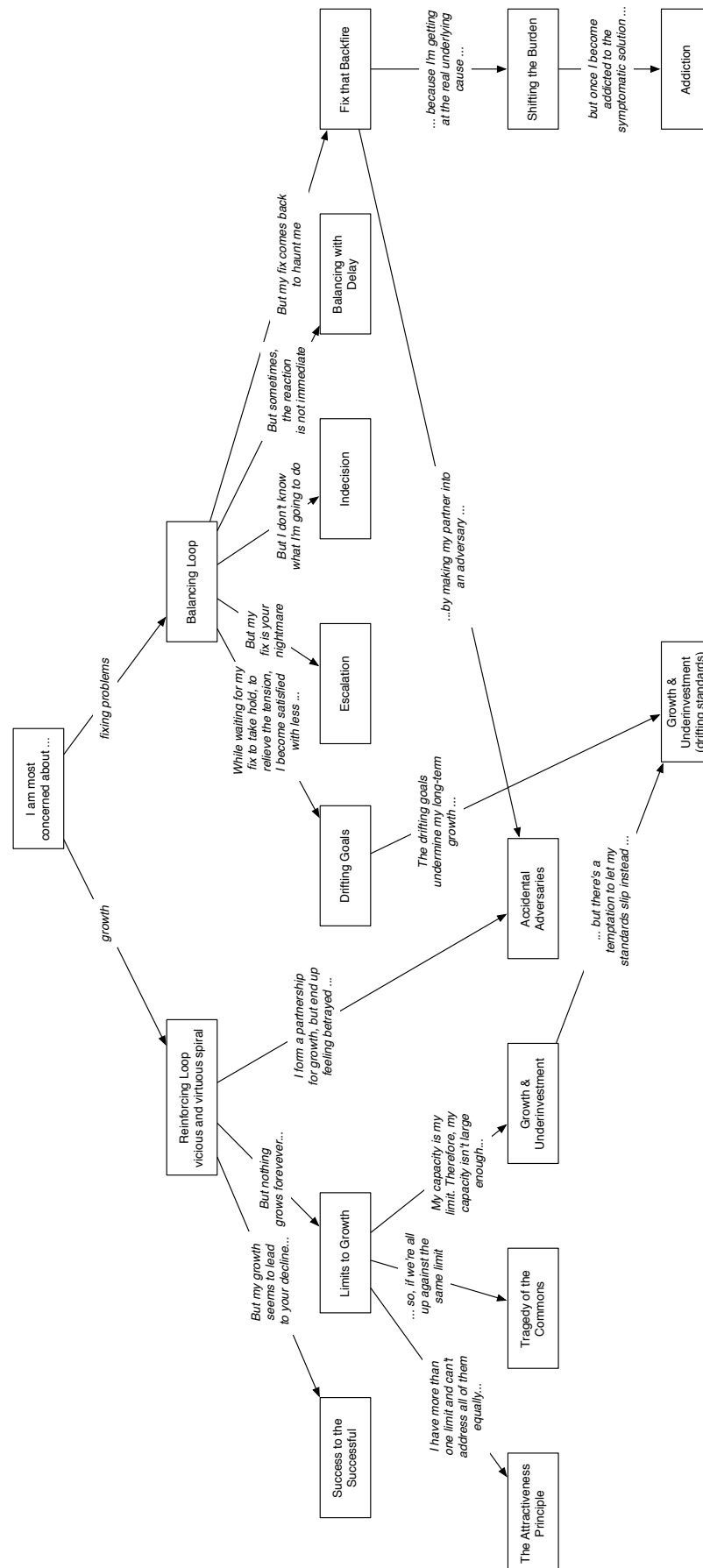


FIGURE 4.40: The Archetype Family Tree

Chapter 5

Simulator - definition

Introduction

To help the analysis of feedback systems, a framework prototype has been developed in order to better understand the behaviour of such systems. The basic idea is to define agents, create links between them, and then to simulate the behaviour with one of the two available simulator engines. Also, the feedback systems that will be simulated are all systems with quantities.

The engines, agents and graphs were implemented in Oz[27, 28] and a graphical interface was built in Java in order to help the user to define the agents and the graphs and to run tests on them.

In this chapter, the definition of agents, of the graphs and of the two simulator engines will be explained.

5.1 Definition of agents

An *agent* is an event-driven piece of code. That means that an agent has several input links coming from other agents as well as several output links going to other agents. An agent is thus typically a variable in a feedback loop.

One can observe in the equations of a model that there are two kinds of equations: ordinary differential equations (ODE) and simple equations. These ODEs represent variables that need a notion of variation over time, that is, for the population example, the population in time t is equal to the population in time $t - 1$ plus the difference between the births and the deaths. In a stock-and-flow system, ODEs describe stocks and simple equations give the computation to be done in flows and auxiliaries/converters.

Both kinds of equations are very important and must be taken into account when creating correct simulations. These simulations will be carried out using the discretization of the differential equations.

In order to model an agent in an event-driven software, a base agent with several features needs to be built.

5.1.1 Base agent

An agent will be represented as a `class` in Oz. Thus the *base agent* will also be a class with several methods as basic features. The code of a base agent is shown in Listing 5.1

The description of the attributes and the methods are explained just below:

- `name`: an atom that represents the name of the agent.
- `ins`: a list of variables that represents input agents.
- `outs`: a list of variables that represents output agents.
- `out_stream`: a list of all output values created by the agent.
- `monitored`: true if the `out_stream` needs to be printed in the console to be used by the graphical interface; false otherwise.
- `stock`: true if the agent is dependant on time.
- `delta`: a value of the delta used in the simulation.
- `init(Id)`: initialises the base agent and sets the attribute `name` to `Id`.
- `is.stock($)`: returns true if the agent is dependent on time.
- `ins($)`: returns the list of input agents.

- `name($)`: returns the name of the agent.
- `monitored(Boolean)`: sets the attribute `monitored` to `Boolean`; true if the agent needs to be monitored - false otherwise.
- `set_ins(Agents)`: sets the attribute `ins` to `Agents`, the list of incoming agents.
- `set_outs(Agents)`: sets the attribute `outs` to `Agents`, the list of outgoing agents.
- `otherwise(Msg)`: prints a message to the user to warn him that `Msg` is not understood by the agent.
- `stop(?End)`: stops the agent, if it is monitored then it prints the output stream and binds `End` to `unit`.

```

1  class BaseAgent
2      attr name ins outs out_stream monitored stock delta
3      meth init(Id)
4          name := Id
5          stock := false
6          delta := 1.0
7      end
8      meth is_stock($)
9          @stock
10     end
11     meth ins($)
12         @ins
13     end
14     meth name($)
15         @name
16     end
17     meth monitored(Boolean)
18         monitored := Boolean
19     end
20     meth set_ins(Agents)
21         ins := Agents
22     end
23     meth set_outs(Agents)
24         outs := Agents
25     end
26     meth otherwise(Msg)
27         {System.show 'Message unknown' #Msg}
28     end
29     meth stop(?End)
30         if @monitored then
31             Stream in Stream = {Reverse @out_stream}
32             for X in Stream do
33                 {System.show @name#X}
34             end
35         end
36         End = unit
37     end
38 end

```

LISTING 5.1: Base Agent

5.1.2 Generic agents

The *generic agent* is the agent that will be modified by the user to fit the system he/she wants to create. This agent extends the class `BaseAgent` and has only one method called `meth m(In ?Out)` that will be invoked during the simulation. This method `m` takes as input the record `In` with features corresponding to the name of the incoming agents. An agent has the possibility to have an attribute that can be a stock value or a change rate for a flow (converters), etc.

If a variable in a causal loop diagram is linked to another that is part of a loop but that the first variable is not part of any loop (like the `birth_fraction` or the `death_fraction` in Figure 3.15), then it is more likely to be used as an attribute in the definition of an agent.

In Listing 5.2, an agent without attribute is represented. The method `m` only contains the instruction `Out = f(In)` where the output of the method is a function of its input(s).

```

1 class AgentName from Agent.baseAgent
2     meth m(In ?Out)
3         Out = f(In)
4     end
5 end

```

LISTING 5.2: Agent without attribute

In Listing 5.3, an agent with one attribute is represented. In this case, the method `init` needs to be rewritten to take into account this/these attribute(s). Then the method `m` works as described above.

```

1 class AgentName from Agent.baseAgent
2     attr attribute
3     meth init(Name Value)
4         Agent.baseAgent,init(Name)
5         attribute := Value
6     end
7     meth m(In ?Out)
8         Out = f(In)
9     end
10 end

```

LISTING 5.3: Agent with attribute(s)

In Listing 5.4, an agent dependent on time is listed, typically a stock. In this case, the attribute `stock` is set to `true` and the equation in the method `m` shows that the value in time t is equal to the value in time $t - 1$ plus the difference between the inflows and the outflows multiplied by a delta fixed by the user at the beginning of the simulation.

```
1 class AgentName from Agent.baseAgent
2     attr attribute
3     meth init(Name Value)
4         Agent.baseAgent,init(Name)
5         attribute := Value
6         stock := true
7     end
8     meth m(In ?Out)
9         attribute := @attribute + (In.inflow - In.outflow) * @delta
10        Out = @attribute
11    end
12 end
```

LISTING 5.4: Agent dependent of the time

5.1.3 Example

To illustrate the use of Oz classes to represent agents, the equations from the example about flowers developed in Chapter 2 have been translated into agents in Listing 5.5.

```
area_of_flowers(t) = area_of_flowers(t-dt) + (growth-decay) * dt
INIT area_of_flowers = 10.0
INFLOW: growth = area_of_flowers * actual_growth_rate
OUTFLOW: decay = area_of_flowers * decay_rate
actual_growth_rate = intrinsic_growth_rate * growth_rate_multiplier
growth_rate_multiplier = - fraction_occupied + 1
fraction_occupied = area_of_flowers / suitable_area
decay_rate = 0.2
intrinsic_growth_rate = 1.0
suitable_area = 1000.0
```

```

1  class Area_flower from Agent.baseAgent
2      attr area
3      meth init(Name Value)
4          Agent.baseAgent,init(Name)
5          area := Value
6          stock := true
7      end
8      meth m(In ?Out)
9          area := @area + (In.growth - In.decay) * @delta
10         Out = @area
11     end
12 end
13 class Growth from Agent.baseAgent
14     meth m(In ?Out)
15         Out = In.area * In.growth_rate
16     end
17 end
18 class Decay from Agent.baseAgent
19     attr rate
20     meth init(Name Value)
21         Agent.baseAgent,init(Name)
22         rate := Value
23     end
24     meth m(In ?Out)
25         Out= In.area * @rate
26     end
27 end
28 class Actual_growth_rate from Agent.baseAgent
29     attr rate
30     meth init(Name Value)
31         Agent.baseAgent,init(Name)
32         rate := Value
33     end
34     meth m(In ?Out)
35         Out = @rate * In.multiplier
36     end
37 end
38 class Growth_rate_multiplier from Agent.baseAgent
39     meth m(In ?Out)
40         Out = ~1.0*In.fraction_occupied + 1.0
41     end
42 end
43 class Fraction_occupied from Agent.baseAgent
44     attr suitable_area
45     meth init(Name Value)
46         Agent.baseAgent,init(Name)
47         suitable_area := Value
48     end
49     meth m(In ?Out)
50         Out = In.area / @suitable_area
51     end
52 end

```

LISTING 5.5: Flowers agents

5.2 Definition of the graph

The basic idea was to be able to represent a graph in a record. To do so, a record with several features has been created. The different features will be explained in this section.

5.2.1 Agents

First of all, after defining all the agents as explained in the previous section, variables to represent these agents need to be initialised. The feature agents: is thus a list of tuples `Variable#Agents.class_name#init(agent_name <value(s)>)#init_output_value`

An example is listed in Listing 5.6 from the same example about flowers.

```

1 agents:
2   [Area_flower#Agents.area_flower#init(area 10.0)#10.0
3     Growth#Agents.growth#init(growth)#9.9
4     Decay#Agents.decay#init(decay 0.2)#2.0
5     Actual_growth_rate#Agents.actual_growth_rate#init(growth_rate 1.0)#0.99
6     Growth_rate_multiplier#Agents.growth_rate_multiplier#init(multiplier)#0.99
7     Fraction_occupied#Agents.fraction_occupied#init(fraction_occupied 1000.0)#0.01]
```

LISTING 5.6: Feature loop.agents

5.2.2 Graph

When all the agents are initialised, the links between them can be established. To do so for every agent, the links from input agents and to output agents will be written as follows: `Variable#[list of input variables]#[list of output variables]`.

For example, the Listing 5.7 gives the graph representation for the same flowers problem. The corresponding graph generated by the simulator is depicted in Figure 5.1.

```

1 graph:
2   [Area_flower#[Growth Decay]#[Growth Decay Fraction_occupied]
3     Growth#[Actual_growth_rate Area_flower]#[Area_flower]
4     Decay#[Area_flower]#[Area_flower]
5     Actual_growth_rate#[Growth_rate_multiplier]#[Growth]
6     Fraction_occupied#[Area_flower]#[Growth_rate_multiplier]
7     Growth_rate_multiplier#[Fraction_occupied]#[Actual_growth_rate]]
```

LISTING 5.7: Feature loop.graph

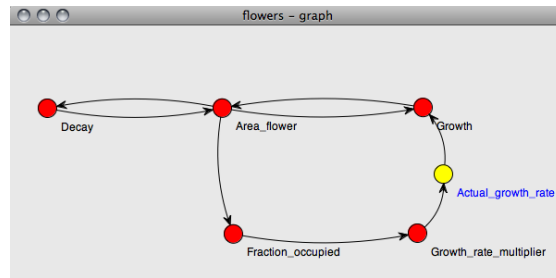


FIGURE 5.1: Flowers: causal loop diagram from the Simulator

5.3 Simulator engines

In parallel with the analysis of a non exhaustive list of feedback systems, the idea was to create a simulator to analyse the behaviour of these feedback structures. Two kinds of simulation strategies have been discussed and implemented in Oz: a synchronous engine and an asynchronous one. In the record filled by the user, the label corresponds to the choice of engine. The user has to type `sync(Delta)` for the synchronous engine and `async(Time Boolean Delta)` where `Delta` represents the delta used during the discretization of the model, that is, if delta is set to 0,1, it means that 10 computation steps are needed to represent simulated time units; `Time` corresponds to the number of milliseconds between two computations and `Boolean` corresponds to the possibility of randomly adding or removing between 0 and 10% of `Time` to/from the number of milliseconds between two computations. The two engines will be explained in detail in this section.

If `Delta` equals 1, each computation done by the stock agents represents one simulated time unit. If `Delta` is greater or less than 1 then $1/\text{Delta}$ computations are needed to compute one simulated time unit. A `Delta` of 0.1 means that 10 simulation steps are needed per time unit and a delta of 10 means each step will represent 10 time units (in this case, if 60 steps are required, only 6 will actually be computed).

Two kinds of agents appear: one that depends on the simulated time and one that does not. That is, stocks, for example, need the information to update their value but this information goes through a certain number of agents that have to be 'instantaneous' in order to avoid implicit delays in the simulations. Each engine is designed (with different techniques) to take into account this difference between both types of agents.

5.3.1 Synchronous engine

Agents have input agents and output agents. The output of an agent is function of its inputs. Thus, the basic idea of the synchronous engine is that at every time step each agent

of the loops computes the output corresponding to its inputs and puts the result on its output stream.

Practically, an agent is implemented by an active object in Oz, that is, “a port object whose behaviour is defined by a class. It consists of a port, a thread that read messages from the port’s stream, and an object that is a class instance. Each message that is received will cause one of the object’s methods to be invoked”. [28]

One agent receives messages from its input agents with their new value. This value is then stored in a stack that represents an input stream. When an agent computes its new value from its inputs, it then puts it in a stack and sends a message to its output agents. This technique corresponds to the *push method* where the information is pushed to the agents (see Figure 5.2).

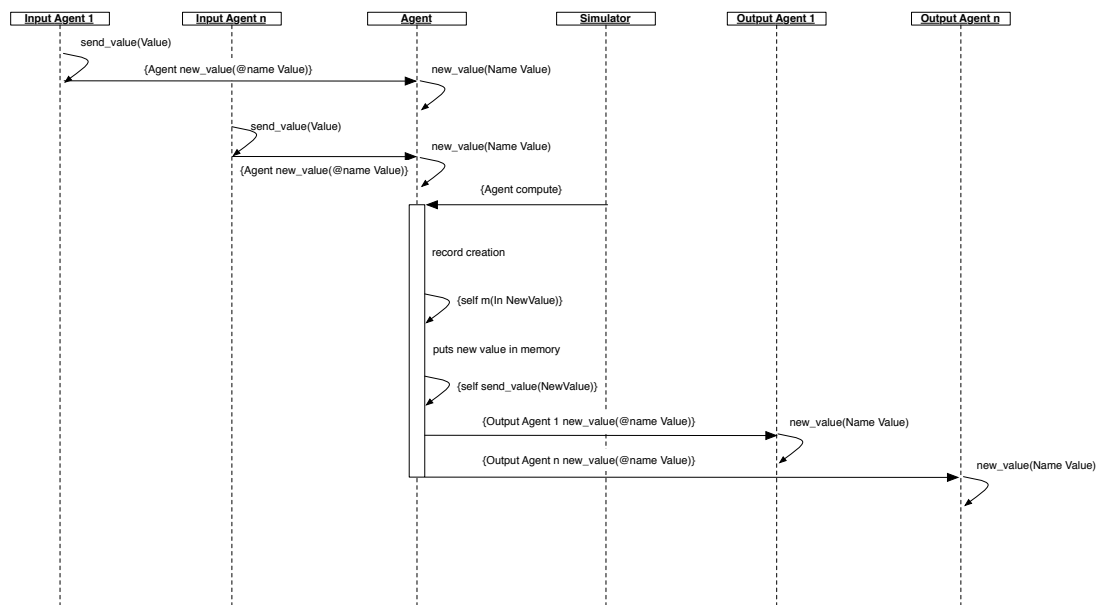


FIGURE 5.2: A synchronous agent sequence diagram

The synchronous engine uses a sequence of ticks to simulate time. Each step t corresponds to $1/\Delta$ units of simulated time for each agent. The idea here is that every agent computes its new value 'at the same simulated time' (agents' computations will be interleaved by the Oz scheduler). As explained above, there are two types of agents (depending on the simulated time or not). To avoid the creation of implicit delays that can make the system oscillate, agents must be executed in a certain order. Stock agents must be executed first before their value is propagated to all the instantaneous agents so that they can update their value. This process has to be done in one computation step. This is done by an execution sequence given by a topological ordering organised in layers so that all the agents in a layer can be executed in parallel.

The intuition for the topological sort organised in layers is as follows:

```

Dico ← empty set that will contain visited nodes
Layers ← empty list of layers
Nodes ← set of all agents

/* Computation of the first layer */
Layer1 ← empty list of agents
for each agent n in S do
    if n is a stock then
        insert n into Dico
        insert n into Layer1
add Layer1 to Layers

/* Computation of the following layer */
Layeri ← empty list of agents
for each agent n in S do
    if n is not a stock then
        if each input agent in of n is in Dico and n is not in Dico then
            insert n into Layeri
insert each agent from Layeri into Dico
add Layeri to Layers
(iterate the computation of the following layer until the resulting Layeri is empty)

```

This algorithm gives for the example given in Figure 5.1 the following list of lists of agents:

```

[Area_flower] | [Fraction_occupied Decay] | [Growth_rate_multiplier] |
[Actual_growth_rate] | [Growth]

```

The fact that Decay is executed before Growth has no influence on the Area_flower because this agent will receive both values before the next time step where it will compute its new value.

For each simulated time step, the simulator will execute each layer in order and will wait until each agent of one layer has finished before moving on to the next layer. This technique is done by a double synchronisation on each agent of a layer: each agent receives a message `compute(X Y)` where *X* and *Y* are unbounded variables; when all the agents of a layer have received this message, *X* is bound to `unit` thus allowing the agents to start their computation and to send their value to their output agents (they have been waiting on *X*); when they are done, they bind *Y* to `unit` and tell the simulator that they have finished their

work, when the simulator knows that every agent in the layer has finished its work, it can proceed to the next layer.

In Listing 5.8, the code for a synchronous agent is listed. This synchronous agent is also a class and extends the class defined by the user that itself extends the base agent explained before.

```

1  class Sync from Class
2      attr time ins_stream monitored
3      meth init(Init Out Delta)
4          Class,Init
5          delta := Delta
6          out_stream := [Out]
7      end
8      meth start
9          @ins_stream = stream()
10         for Agent in @ins do
11             Name in Name = {Agent name($)}
12             ins_stream := {Adjoin @ins_stream {Record.make stream [Name]}}
13             @ins_stream.Name = {NewCell nil}
14         end
15     end
16     meth send_value(Value)
17         for Agent in @outs do
18             if Value == init then
19                 {Agent new_value(@name Out)}
20             else
21                 {Agent new_value(@name Value)}
22             end
23         end
24     end
25     meth new_value(Name Value)
26         {Assign @ins_stream.Name Value|{Access @ins_stream.Name}}
27     end
28     meth compute(X Y)
29         {Wait X}
30         In NewValue in
31             In = {Record.make ins {Arity @ins_stream}}
32             for Name in {Arity @ins_stream} do
33                 List in List = {Access @ins_stream.Name}
34                 case List
35                     of X|nil then In.Name = X
36                     [] X|_ then In.Name = X
37                     else In.Name = 0.0
38                 end
39             end
40             {self m(In NewValue)}
41             out_stream := NewValue|@out_stream
42             {self send_value(NewValue)}
43             Y = unit
44         end
45     end

```

LISTING 5.8: Synchronous agent

init(Init Time Out): initialises the agent with the method `Init` meant for the agent created by the user then sets the attributes `delta` and `out_stream` to their initial values.

start: starts the agent by creating the cells for the incoming streams.

send_value(Value): sends a message `new_value(@name Value)` with the new value freshly computed or with the initial value that the user indicated in the graph definition.

new_value(Value): stores the new value in the corresponding cell representing the incoming stream. In this case, the stream is represented as a stack.

compute(X Y): at every computation step the simulator sends a message that enables this method to compute the new value that is a function of the input streams. The results are put onto the output stream and then sent to all the output agents. Both parameters are there to guarantee the synchronous characteristic.

5.3.2 Asynchronous engine

The idea of the asynchronous engine is that every agent works 'alone' and does not depend on the simulator to compute its new value. Once the agents have started, they compute their new value whenever they need to. The technique used to propagate values is not the *push method* but the *pull method*. That is, if an agent needs to compute its new value, it will ask its input agents to give it their current value. When the agent has received all of the necessary information, it will then proceed and compute the new value, and store it. This is shown in Figure 5.3.

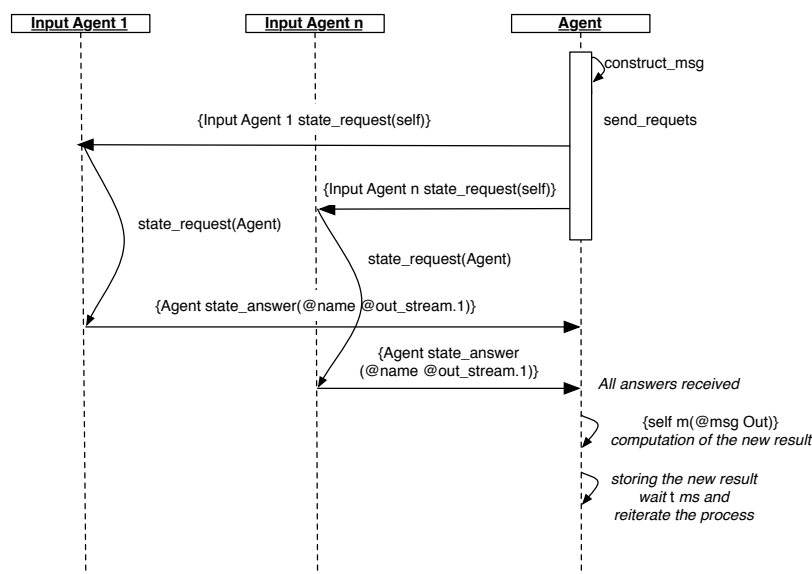


FIGURE 5.3: An asynchronous agent sequence diagram

The asynchronous engine does not use a sequence of ticks to simulate the time. It actually uses physical time to represent time. That is, agents that do depend on time will first compute their new value and will then wait for Time milliseconds before computing the next value. Also this time is waited in a thread to allow the agent to answer requests from other agents. The other agents that need to be 'instantaneous' will not wait and will continuously update their value. The waiting time is the same for all the time-dependent agents, it gives

approximatively the same amount of observation. This is because agents do not have the notion of how many computation steps they have done. So basically, they do not know when to stop. Another thread is thus waiting for the number of observations asked by the user times the time also set by the user. Once this time has passed, this thread sends messages to all the agents to stop them at the same time. The only problem is that this thread does not take into account the time consumed by the computations that should have been added to the total time it needed to wait. The number of real computations does not correspond to the one asked by the user but for a more convenient analysis, the resulting behaviour is plotted to correspond to the exact number of computations.

In this case, the topological ordering will not be used. Instead, the fact that physical time is used to simulate time is enough to allow the instantaneous agents to 'propagate' the new value from a stock as fast as possible before the agent depending on the time finished waiting and computes its new value. This technique may create implicit delays in the behaviour of models if the time indicated by the user is not enough for the instantaneous agent to propagate the new value from stocks to flows. That is, if there are four agents in a loop (the loop starting with a stock) then the information from the stock needs to propagate onto the three other agents. This process needs at least three computation steps, depending on the ordering of the agents given by the Oz scheduler and at most seven if the scheduler is fair. So, the `Time` set by the user should be of at least approximately 100 ms in order to allow the system to propagate the information from a stock to all the dependent 'instantaneous agents'.

The Oz scheduler defines the sequence in which the agent will be executed. Once this sequence is set, it will not change because of the use of the round robin system. To add the aspect of complete randomness to the simulation, an agent can pass and leave the sequence to position itself at the end of the queue. This process is only applied to stock agents (the application to non time depending agents is useless due to the fact that they compute new values all the time).

Another feature for tweaking the simulation has been added in order to allow the agents to wait for different times. This allows there to be stocks that work faster than others or vice versa. If the parameter `Boolean` is set to true then, when the asynchronous agent is created, a maximum of 10% of `Time` is added or withdrawn from it. This characteristic will change the behaviour of the system (see next Chapter).

In Listing 5.9, the code for an asynchronous agent is listed. This asynchronous agent is also a class and extends the class defined by the user that itself extends the base agent explained before.

```

1  class Async from Class
2    attr time msg insName
3    meth init(Init Time Out Random Delta)
4      Class, Init
5      delta := Delta
6      if Random then
7        Ten Sign in
8        if Time div 10 < 1 then Ten = 1
9        else Ten = Time div 10 end
10       Sign = {OS.rand} mod 2
11       if Sign == 0 then time := Time + ({OS.rand} mod Ten)
12       else time := Time - ({OS.rand} mod Ten) end
13     else time := Time end
14     out_stream := [Out]
15     insName := nil
16   end
17   meth start
18     for Agent in @ins do
19       Name in {Agent name(Name)}
20       insName := Name|insName
21     end
22     {self construct_msg}
23   end
24   meth construct_msg
25     msg := {Record.make state @insName}
26     msg := {Adjoin @msg {Record.make state [1]}}
27     @msg.1 = {NewCell 0}
28     {self send_requests}
29   end
30   meth send_requests
31     for Agent in @ins do
32       {Agent state_request(self)}
33     end
34   end
35   meth state_request(Agent)
36     {Agent state_answer(@name @out_stream.1)}
37   end
38   meth state_answer(Name Info)
39     @msg.Name = Info
40     {Assign @msg.1 ({Access @msg.1} + 1)}
41     if {Access @msg.1} == {Length @ins} then
42       Out in
43         if @stock andthen {OS.rand} mod 2 == 1 then
44           {Thread.preempt {Thread.this}}
45         end
46         {self m(@msg Out)}
47         out_stream := Out|out_stream
48         if @stock then thread {Delay @time} {self construct_msg} end
49         else {self construct_msg} end
50     else skip end
51   end
52 end

```

LISTING 5.9: Asynchronous agent

init(Init Time Out Random Delta): initialises the agent with the method `Init` meant for the agent created by the user then sets the attributes `delta`, `time`, `out_stream`, and `ins_name` to their initial values. The attribute `time` corresponds to the time entered by the user with a delta of maximum 10% depending on the value of `Random`. If this

boolean is true then a maximum 10% of Time is added or withdrawn to/from this time, nothing is changed if Random equals false.

start: starts the agent by collecting the names of the input agents. Their names are used to construct the message that will be filled progressively by requests sent to the input agents.

construct_msg: constructs a message state(1: 0 InputName1: _ ... InputNameN: _)

send_requests: sends the requests to all the input agents.

state_request(Agent): when a request is received from Agent, the agent sends a message to this same agent with its current state.

state_answer(Name Info): this is the answer received by the original agent with the Name of the input agent and its current value (Info). When all the answers are received, the agent computes its new output value and stores it. Then if stock is set to true then it will wait for Time ms before computing the next value. If it is set to false, then the next computation is carried out directly.

5.4 Graphical interface

To facilitate the use of the simulator, a graphical interface has been developed. This interface is implemented in Java using the Swing libraries. Two other libraries have been used to implemented the graph representation[29] and the chart representation[30].

5.4.1 Main panel

A main panel (see Figure 5.4) has been developed to facilitate the user input. This panel contains three main parts:

menu bar: to facilitate the access to commands such as the creation of a new project, opening an existing project, saving the current project, showing the graph or chart panel, etc.

agents left sub-panel: this sub-panel is dedicated to the definition of agents: the user can create new agents, edit existing agents, save their definition, delete agents, or simply look at the definition of an agent chosen from the list on the top of the panel.

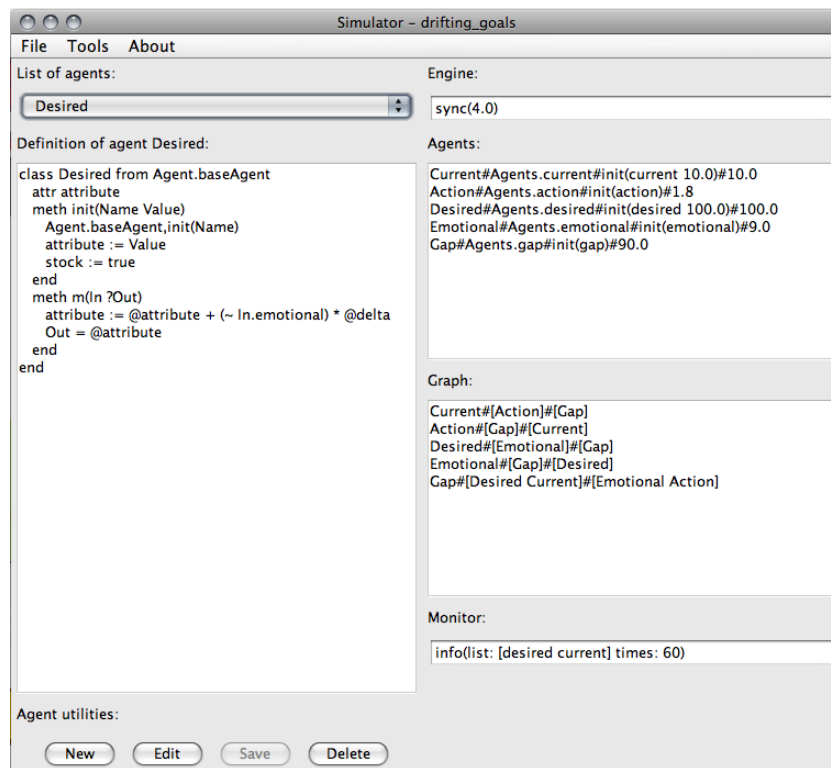


FIGURE 5.4: Simulator main panel

graph right sub-panel: this sub-panel is used to put information about the graph definition. First the user has to choose the engine of the simulator, then create the agents as explained earlier, then construct the links between the agents and finally add information about which agents need to be monitored during the simulation and the number of times they have to be monitored. Each agent name put in the `list:` feature will be represented in the chart.

More practically, a hashtable is used to store the agent definitions. This data structure is very convenient for this type of situation where access to an agent definition, or the creation or the deletion of an agent could be frequent.

When the user wants to save the current project, a folder is created with a name entered by the user in a folder `projects`. Then, two files are copied in this folder with the definition of a base agent and of the simulator. Finally, the content of the hashtable is copied in a file where agent definitions will be stored and the graph definition will also be copied in a separate file.

5.4.2 Graph representation

This tool was designed to give the user the possibility of observing the links he/she has created between agents. A library created by three researchers has been used to represent the graph. This library - Java Universal Network/Graph Framework - consists of vertex and edges, it also allows for the manipulation of the vertex in the panel in order to obtain the best disposition for the user. Thus, when the user chooses to get the graph representation, the text from the Graph text field is used to create the graph. An example of a graph is depicted in Figure 5.5.

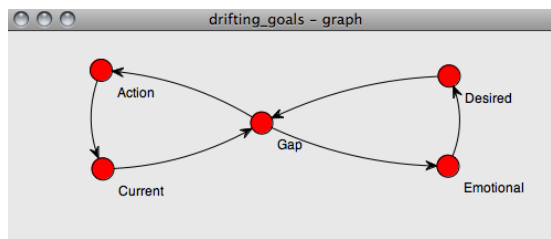


FIGURE 5.5: Simulator graph panel

This example consists of five vertices and six edges. The user can click on a vertex and move it corresponding to his/her convenience. To access this panel, the user can either go via the menu bar **Tools->Graph** or use the shortcut **Alt+g**

5.4.3 Chart representation

This other tool takes the output of the simulation and plots it on a chart. The user can use the record **info** in the text field **Monitor** to set the parameters of the monitoring. That is, the user can choose the agent(s) to be observed and the number of observations. An example is depicted in Figure 5.6

After the simulation has been undertaken, the output is captured and plotted in a chart. To access this panel, the user can either go via the menu bar **Tools->Behaviour** or use the shortcut **Alt+b**

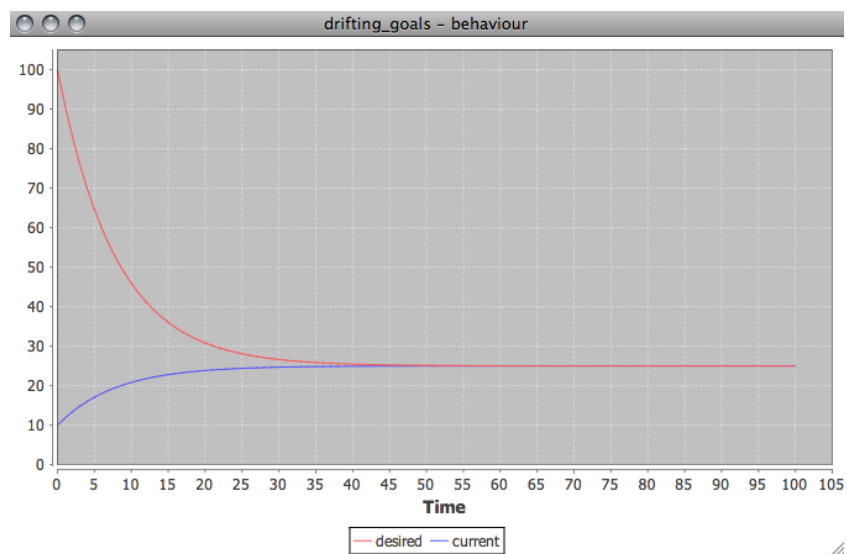


FIGURE 5.6: Simulator chart panel

Chapter 6

Simulator - results

Introduction

Feedback systems have been tested using the simulator to observe whether there are differences between them and the results presented in articles I have read. In this section, such differences will be explained.

In parallel with the simulator that I developed, I used another simulator called Stella[31] that uses stock-and-flow representation. For each result presented below, I tested the same model with the same equations and values, with Stella and then with my simulator. The synchronous engine developed here corresponds to the Stella engine where stocks (or agents depending on time) are computed first then their new value is used to calculate the value of the 'instantaneous' agents. Also, very few examples with a complete set of equations were found in all the articles that I read. Thus sometimes, I had to find the right parameters to remain close to the depicted behaviour.

6.1 Delta time

The delta time – dt – is the interval of time between calculations. The dt is the answer to the question: do I have to re-calculate the numerical values of the model once every time unit, twice, three times, ...? For example in the population model, if one time unit represents a year and that the user asks for 100 observations, the simulation will show the evolution of the population for a century. When the population is evaluated depends on the dt value.

Sometimes, the dividing into smaller fragments of a time unit might be interesting in order to analyse the change in the model's behaviour. In general, a smaller dt means smoother, more continuous changes. A greater dt means rougher changes that can disrupt the expected behaviour of the studied model.

Also, the choice of dt can create 'artefact oscillations' (see Section 6.2.7). Sometimes the change is too important due to large values of dt , which means that a value can go under or over its limit. This process often implies that the value needs to go in the other direction but once again due to the late change, the value goes on the other side of the limit which means that it needs to be redirected to its limit. This process creates oscillations where the value overshoots or undershoots its limit instead of progressing towards it. In this case, a good practice is to cut the delta value in two and to re-simulate the model until the oscillations disappears. The modeller has to find out whether these oscillations are real or artefacts created by the choice of dt . In general though, real oscillations due to delay persist even with small dt values.

In summary, if the dt is set to a value less than 1, it makes the analysis of change that occurs within a time unit possible. In general the trade-off is speed against smoothness and numerical precision. If the dt is small then changes are smoother and the results are more precise numerically but more calculations are needed which takes up more time to complete the simulation. Large values of dt give less precise results but, in some cases, can give a good approximation of the model's behaviour in a faster simulation.

6.2 Results

Results of the simulations with both engines and different values for the engines parameters will be presented in this section. All the result graphs will be put in the Appendix dedicated to these results.

6.2.1 Population

In Section 3.2.5, an example of coupling a negative feedback loop and a positive feedback loop was explained. Depending on the dominant loop, the behaviour of the combination of both loops is either an exponential growth or a exponential decay.

This model of the evolution of the population was tested in the simulator with both engines and both configurations for the asynchronous engine. The simulation used the following set of equations where the birth rate is fixed at 6% and the death rate at 3%. The resulting behaviour should therefore be an exponential growth. Results from the Stella simulation (with $dt = 1$) are depicted in Figure 6.1

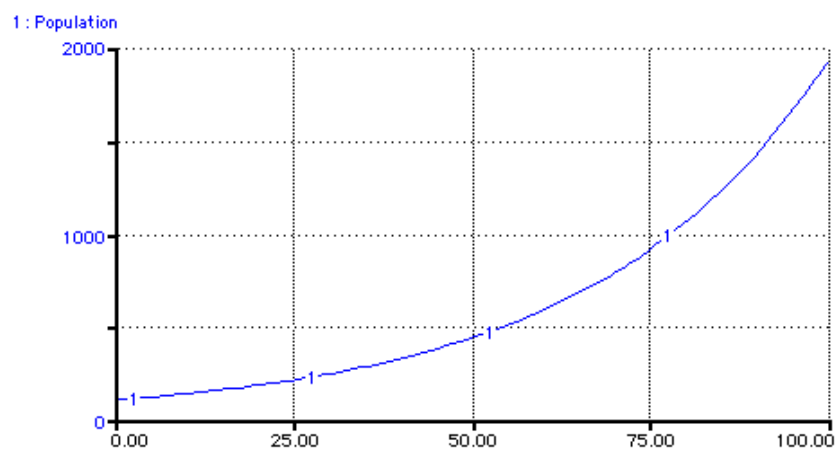


FIGURE 6.1: Stella: the population model ($\delta = 1$)

```
Population(t) = Population(t - dt) + (Births - Deaths) * dt
```

```
INIT Population = 100.0
```

```
INFLOW: Births = Birth fraction * Population
```

```
OUTFLOW: Deaths = Death fraction * Population
```

```
Birth fraction = 0.06
```

```
Death fraction = 0.03
```

6.2.1.1 Synchronous engine

The population model has been simulated with three different values for the delta time parameter of the synchronous engine. Results are depicted in Figure 6.2 One can clearly observe the facts about the delta value.

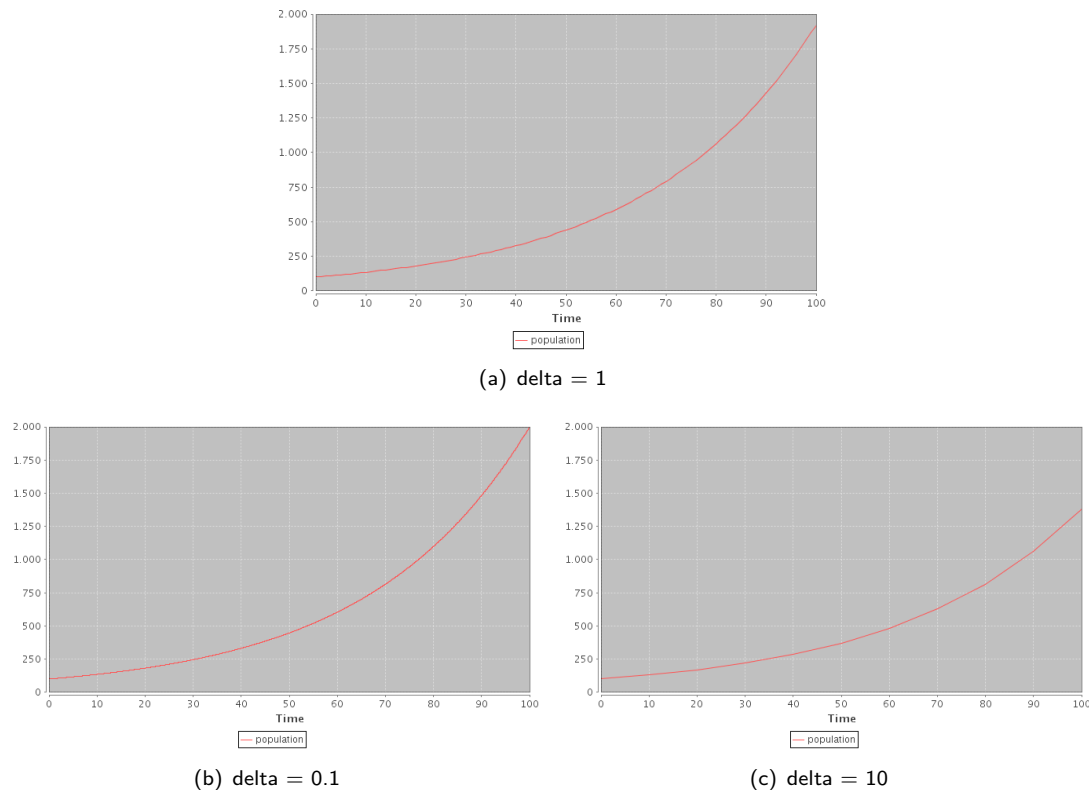


FIGURE 6.2: Simulator: the population model – synchronous engine

Figure 6.2(a) corresponds to the simulation done with Stella depicted in Figure 6.1. Both simulations give a resulting population of about 1900 people after 100 periods. This result uses exactly 100 computations of the population value (once a year during a century).

Figure 6.2(b) shows the results of the population where the simulations are made ten times a year. This configuration gives better accuracy in the final results (final population of 2000 people), but it took ten times longer to compute these results ($100/0.1 = 1000$ instead of 100). This model already gives good approximation of the model behaviour with a delta equal to 1. Lower value of delta will only give the 'same' approximation but using slower simulations.

Figure 6.2(c) depicts the situation where dt is equal to 10 which means that a computation will represent ten time units, that is, ten years. One can observe that the final population is of around only 1400 which is a lower than the two other simulations. This result has been obtained in a really fast simulation. It does not give the exact result but a good first approximation of the behaviour of the population model.

6.2.1.2 Asynchronous engine

As observed in the results from the synchronous engine, a delta value of 1 gives good approximation of the behaviour and also gives good approximation of the final value of the population after 100 years. Only the results for dt equal to 1 will be discussed here. The same conclusion may be applied in the two other cases. Results are depicted in Figure 6.3.

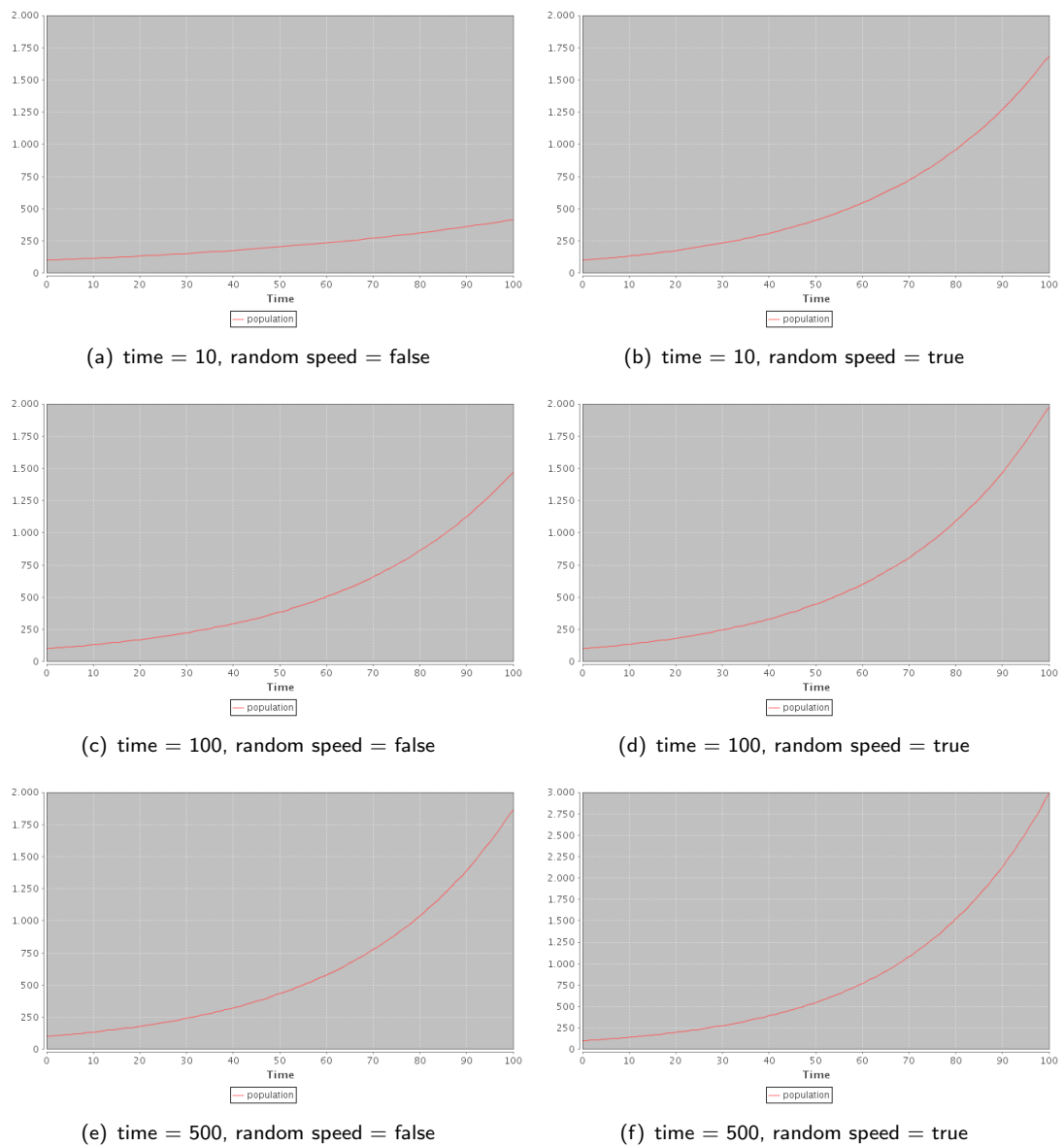


FIGURE 6.3: Simulator: the population model – asynchronous engine (delta = 1)

Figures 6.3(a)–(b) show the results from the simulation with a time set to 10 milliseconds. That is, each calculation will be carried out every 10 milliseconds. The final population in Figure 6.3(a) is of around 450 people. This can be explained as follows: because of a low time between each calculation for the population agent, its new value does not have time to

propagate to all the instantaneous agents before its next value is computed, which means that this new computation is based on an older value. That is, the resulting growth is lower than it should normally be. On the other hand, in Figure 6.3(b), a delta of maximum 10% of the time has been added or withdrawn to/from it. This means that the population is slightly slower which allows the instantaneous agents to better propagate the new value from the population agent. Then the new population value will be computed with better values. This gives a better approximation of the final population (around 1700 people) and of the model's behaviour.

Figures 6.3(c)–(d) depict the same simulation but with a time of 100 ms. This allows the instantaneous agent to better propagate new values from the population agent. In the first case, the results give a good approximation of the behaviour but a slightly worse approximation of the final population (less than 1500 people). When the population has a random speed between 90 ms and 110 ms, then the results could be the same as for the synchronous engine with a dt of 0.1 ('could' because this is one of the possible results of the asynchronous simulation engine).

The final population with a time period between 9 and 11 ms is better approximated than with a time period of 100 ms. Because of the asynchronous characteristic of the engine, the agents have to be stopped after a certain time because they do not know when to stop. In the case where all agents have the same speed, a thread is waiting for a number of observations times the time set by the user before stopping the agents. This generally gives less observations than required by the user. On the other hand, if the time must be random, the thread waiting will wait for the number of observations times the time + 10% given by the user. As the time is randomly changed at the creation of the asynchronous agent and the thread always wait for the case where 10% is added, the number of observations can be greater than the one required if the agent is faster than the original time. This fact can also be observed in Figure 6.3(f) where the final population is greater than the correct approximation given by the synchronous engine.

Figures 6.3(e)–(f) represent the results of the simulation with a time of 500 ms. In this case, Figure 6.3(e) gives good approximations of both the behaviour and the final population value. But as just explained above, one can observe that in Figure 6.3(f) the population is faster than the time set by the user and due to the waiting thread that will stop the agents, there are more observations. This gives a higher value of the final population that is not correct.

6.2.2 Drifting goals

This example was explained in detail in Section 4.13. This model consists of an expected situation defined by an organisation whose current situation is lower than the one expected. Actions are therefore taken in order to increase the current situation whilst emotional tension decreases the expected situation. As the emotional tension is stronger than the actions taken, the expected situation decreases faster than the current situation increases until the both meet just above 25 (if $dt = 1$). The resulting behaviour of the model from the Stella simulator is depicted in Figure 6.4. The following set of equations was used.

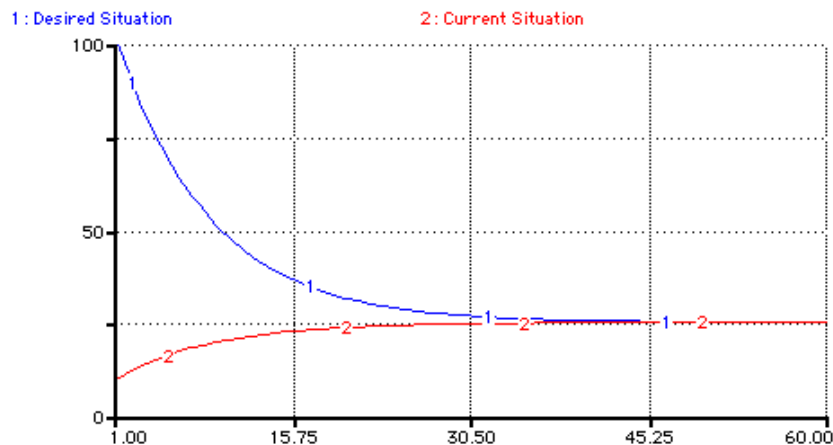


FIGURE 6.4: Stella: the drifting goals model ($\delta = 1$)

```

Current situation(t) = Current situation(t - dt) + (Action) * dt
INIT Current situation = 10.0
INFLOW: Action = 0.02 * Gap
Desired situation(t) = Desired situation(t - dt) + (- Emotional tension) * dt
INIT Desired situation = 100.0
OUTFLOW: Emotional tension = 0.1 * Gap
Gap = Desired situation - Current situation

```

6.2.2.1 Synchronous engine

The drifting goals model has been simulated with three different values for the delta time parameter of the synchronous engine. Results are depicted in Figure 6.5. Once again, one can observe the facts about the delta values. This time, artefact oscillations are also observable.

Figure 5(a) shows the same simulation results as the ones produced by Stella (see Figure 6.4). Both simulations give a dynamic equilibrium of just above 25 after 60 periods of time.

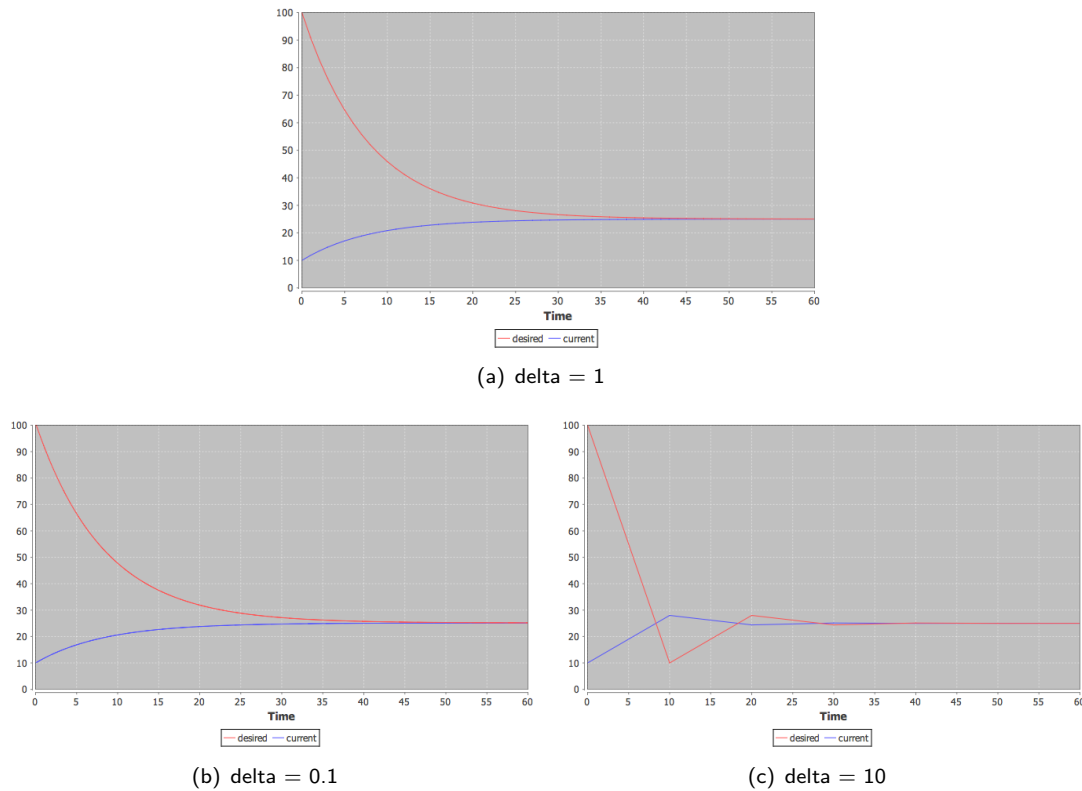


FIGURE 6.5: Simulator: the drifting goals model – synchronous engine

Figure 5(b) represents the same simulation results as for a dt of 1. This means that the model already gives good approximation of the behaviour and of the dynamic equilibrium value with only 60 computations (compared to the 600 calculations of the 0.1 dt).

Figure 5(c) depicts the situation where artefact oscillations appear in the model. These oscillations come from the δ in the equations of stocks. In this case, the δ is equal to 10 meaning that at every computation the effect of flows will be multiplied by 10. After 10 time units (simulated time) or the first computation, the emotional effect is more important than in the case where δ equals 1. This affects the desired situation that will go below the current situation. In this case, the gap is now negative, implying that the two situations will go in the opposite direction: the current situation will now decrease and the desired situation will increase due to their definition (it will inverse the direction of both flows). During the next step, the change still being too important, both situations will return to their 'normal situation' (the current one under the desired one). This process will happen until they both reach their dynamic situation at about 25. This value of dt gives a good approximation of the dynamic equilibrium's value but not a good approximation of the model's behaviour due to the 'big steps' in the simulation.

6.2.2.2 Asynchronous engine

Three interesting conclusions have been drawn from the analysis of the behaviours resulting from different configurations of the asynchronous engine for the drifting goals model.

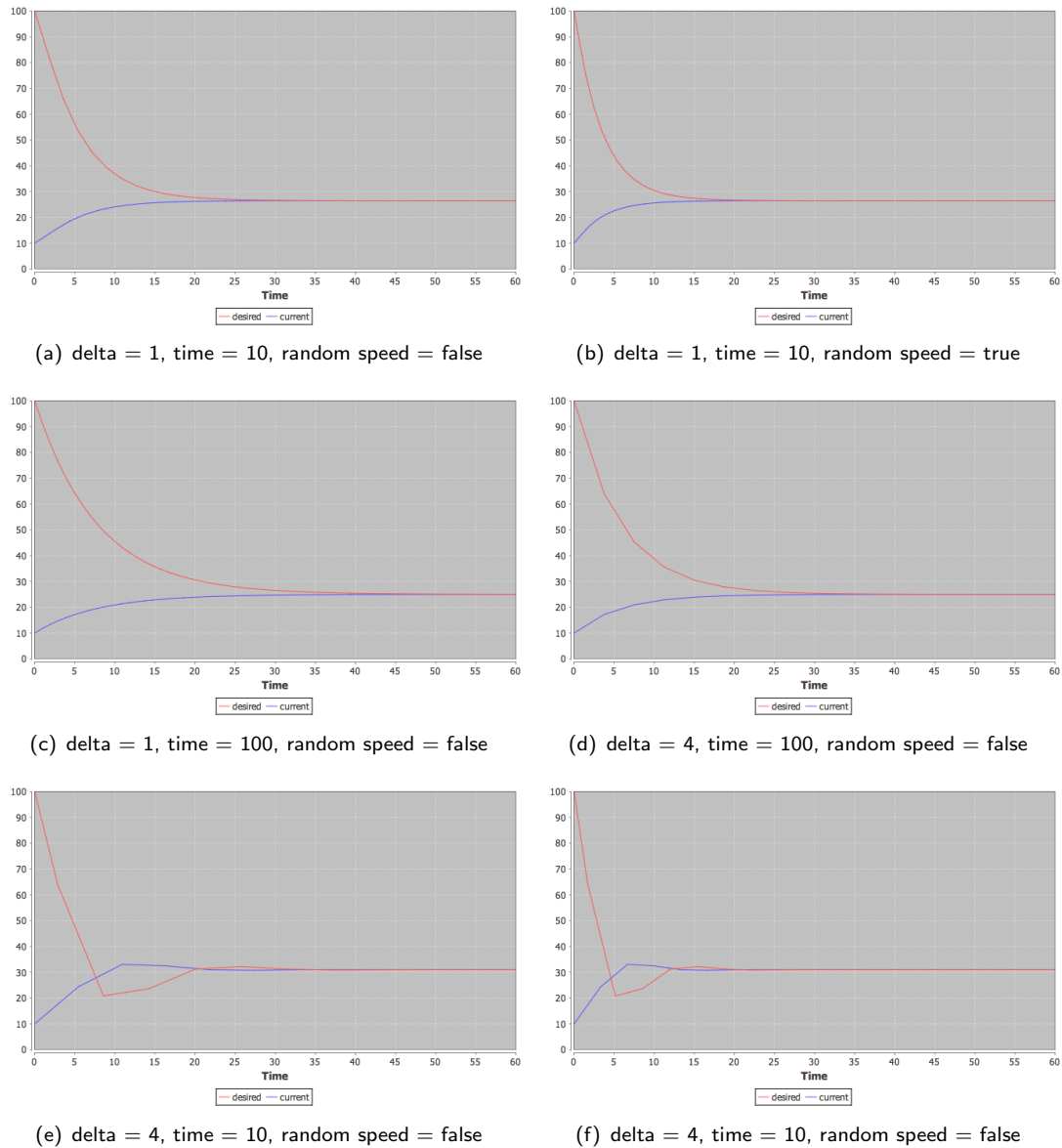


FIGURE 6.6: Simulator: the drifting goals model – asynchronous engine

Figures 6.6(a)–(b) shows that the dynamic equilibrium is obtained earlier in the simulation when stock agents have different time. They produce more observations than in the situation where they all have the same time because of the total time waited by the thread that stops the agents.

Figures 6.6(c)–(d) represents the variation of the value of δt . One can observe that if δt equals 1 then the curves are smooth and that if δt is equal to 4 then the results give the

same approximation for the model's behaviour as for the value of the dynamic equilibrium but that the curves are not as smooth as for a dt equal to 1. This is due to the number of calculations done per simulated time unit.

Figures 6.6(e)–(f) depict three facts. The first one is the creation of artefact delays as explained in the results of the synchronous engine. The second one is due to the low time assigned to the agents. The 10 ms between calculations of new value for stock agents is not enough to propagate this information to the 'instantaneous agents'. The stock agents base their calculations on older values that are translated by a higher dynamic equilibrium value. The third one is the earlier equilibrium due to the different speed of the stock agents creating more observations than required.

6.2.3 Epidemic

The dynamic behaviour of epidemic was explained in Section 4.2.2. An initial population of people susceptible of falling ill meets another population of ill people. This contact will progressively drain the susceptible people to the infected population until it reaches a value near 0 where the infected will progressively decay due to the lack of susceptible people. One can observe that the illness will disappear and that the susceptible population will grow again. Results from the Stella simulator are presented in Figure 6.7.

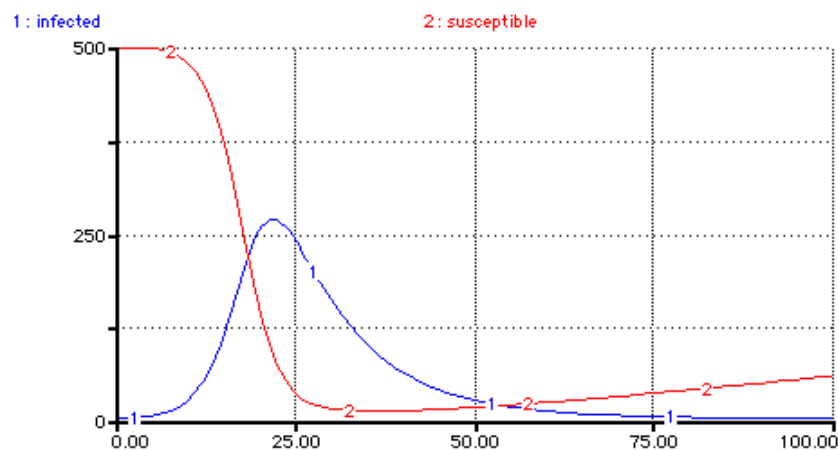


FIGURE 6.7: Stella: the epidemic model ($\delta = 1$)

The following set of equations was used:

```

Infected(t) = Infected(t - dt) + (Infection rate - Removal) * dt
INIT Infected = 1.0
INFLOW: Infection rate = Infection fraction * Contacts
OUTFLOW: Removal = Removal fraction * Infected

```

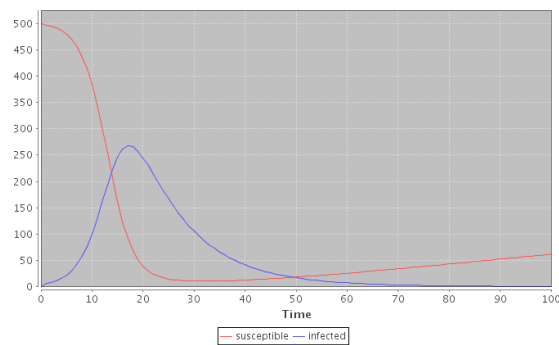
```

Susceptible(t) = Susceptible(t - dt) + (In - Infection rate) * dt
INIT Susceptible = 500.0
INFLOW: In = 1.0
OUTFLOW: Infection rate = Infection fraction * Contacts
Contacts = Contact fraction * Susceptible * Infected
Contact fraction = 0.1
Infection fraction = 0.01
Removal fraction = 0.1

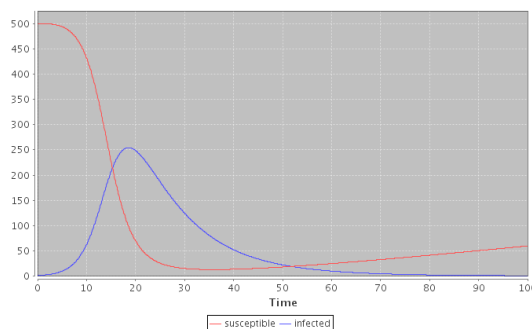
```

6.2.3.1 Synchronous engine

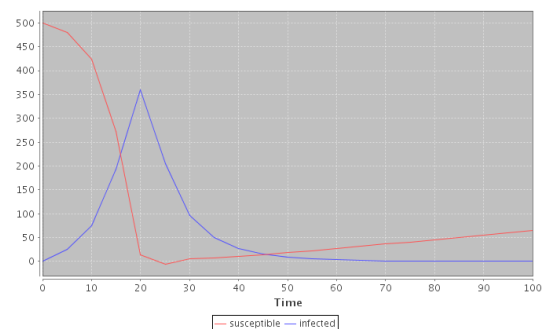
The drifting goals model has been simulated with three different values for the delta time parameter of the synchronous engine. Results are depicted in Figure 6.8. Once again, one can observe the facts about the delta values.



(a) delta = 1



(b) delta = 0.1



(c) delta = 5

FIGURE 6.8: Simulator: the epidemic model – synchronous engine

Figure 9(a) shows the same simulation results as the ones produced by the Stella simulator (see Figure 6.7). Both simulations give a dynamic equilibrium of 60 for the susceptible population and 0 for the infected population.

Figure 9(b) represents the situation where dt equals 0.1. In this case, this dt value gives a better value for the successive changes in both populations. The decrease of the susceptible population is less steep due to the increased number of computations for each time unit. This implies a lower maximum for the infected population.

Figure 9(c) depicts the situation where dt is set to 5. This configuration gives rougher approximations of the model's behaviour: the curves are not as smooth as for the two other dt values. The maximum of infected people is also higher because changes appear less often and are more important than in the other configurations. One can also observe that the susceptible population becomes negative between 20 and 30. These negative values are not possible for a population model: population cannot be negative. This raises a question: could non linearity be added in the equations of flows to limit stocks to their 'natural values'?

During the construction of the simulator, such a technique was used in order to restrict the stocks to their physical values. If a stock agent has a negative value, this technique consists in considering the stock as a zero value stock if it is negative. This gives a qualitatively better approximation of the model shape. Moreover, limiting stock in their equations is not a good technique as it can break the conservation law.

6.2.3.2 Asynchronous engine

Three observations can be extracted from the simulation results of the asynchronous engine with different parameter values. These are presented in Figure 6.9

Figures 6.9(a)–(b) show two interesting facts. Firstly, when all of the agents have the same time, the illness is not eradicated. This is due to the low time value that implies the non correct propagation of new values of stock agents to the instantaneous agents (stocks will thus base their calculations on old values). Second, when agents have different time, the final value of the susceptible population is higher than in the synchronous equilibrium. The susceptible population seems to be faster than the infected one and as the inflow of the susceptible population is constant, it reaches a higher final value. On the other hand, the peak in the infected population is lower than in the synchronous situation.

Figures 6.9(c)–(d) represent the situation where δ equals 1. As the time is set to a low value (in both cases), the propagation is also not done properly. This gives a less steep decrease of the susceptible population. But the decrease is too important (because older values are used) which means that the susceptible population once again goes below its natural limit. When the agents are set with different time, the change appears earlier in the simulation. This process happens because there are more observations than required (also implying a higher final value of the susceptible population).

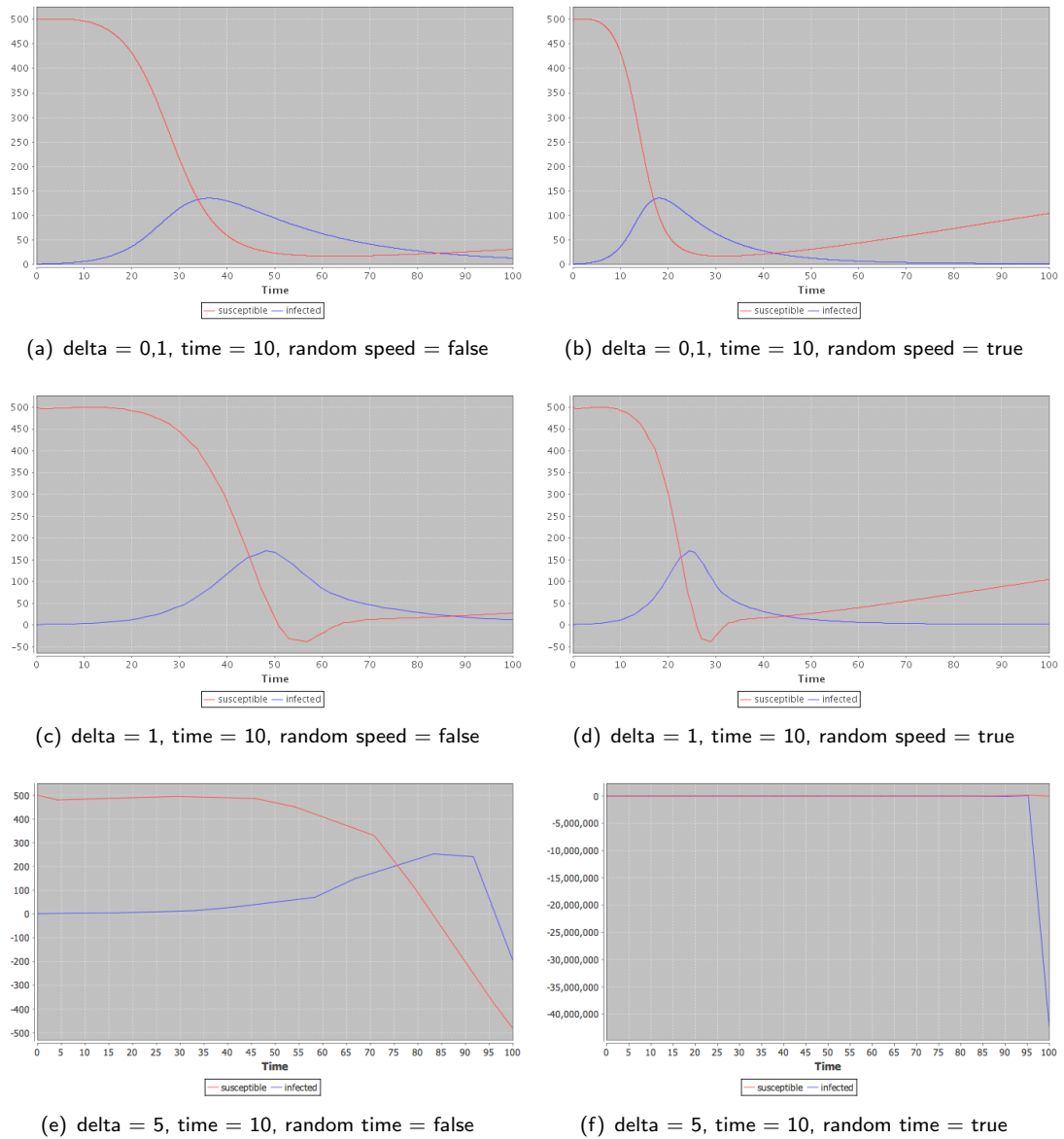


FIGURE 6.9: Simulator: the epidemic model – asynchronous engine

Figures 6.9(e)–(f) depict the combination of two phenomena: firstly, the fact that the time between computations is only 10 ms does not allow the propagation of new values from stocks to be correctly propagated throughout all of the dependent instantaneous agents. In addition to this 'bad' effect, computations will only be carried out every 5 simulated period implying more 'drastic' changes. These phenomena can be observed in Figure 6.9(e) where the agents all have the same time. The susceptible population goes under its natural limit due to computations with older value and the multiplication of changes by 5. On the other hand, in Figure 6.9(f), values for the infected population converge to minus infinity. The model does not support high delta values and small time.

6.2.4 Crowding

In Section 4.2.1, an example of limited growth was explained. It was said that the S-shaped growth was the more interesting part because it consists of two phases: an exponential growth and a goal-seeking phase to the limit. Results from the Stella simulator are depicted in Figure 6.10. Simulations have been run with the following equations and values:

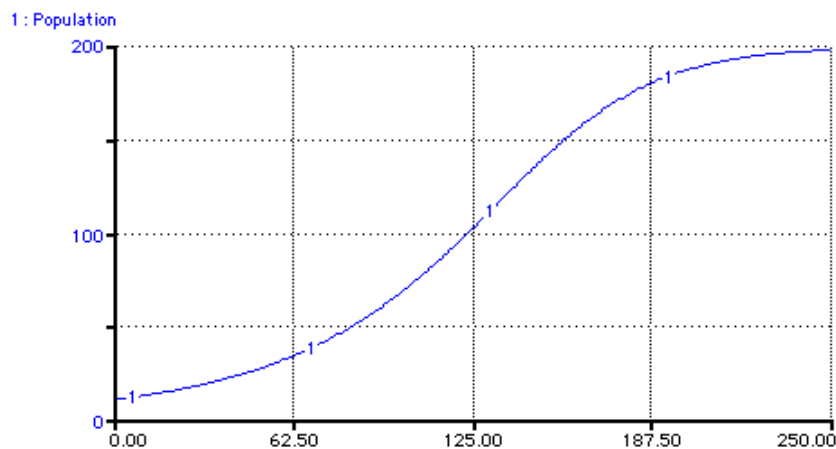


FIGURE 6.10: Stella: the crowding model (delta = 1)

```

Population(t) = Population(t - dt) + (Births - Deaths) * dt
INIT Population = 10.0
INFLOW: Births = Birth fraction * Population
OUTFLOW: Deaths = Death fraction * Population
Birth fraction = 0.08 - 0.000028561 * Crowding - 0.02 * Crowding^2.0
Capacity = 200.0
Crowding = Population / Capacity
Death fraction = 0.06

```

6.2.4.1 Synchronous engine

The crowding model has been simulated with three different values for the delta time parameter of the synchronous engine. Results are depicted in Figure 6.8.

In this case, because the dynamic equilibrium is obtained after 250 simulated time unit, the value of dt does not really influence the approximation of the model's behaviour nor the final value of the population. One can observe that the smaller the dt value, the later the transition between the two phases appears: the exponential growth followed by the goal-seeking phase.

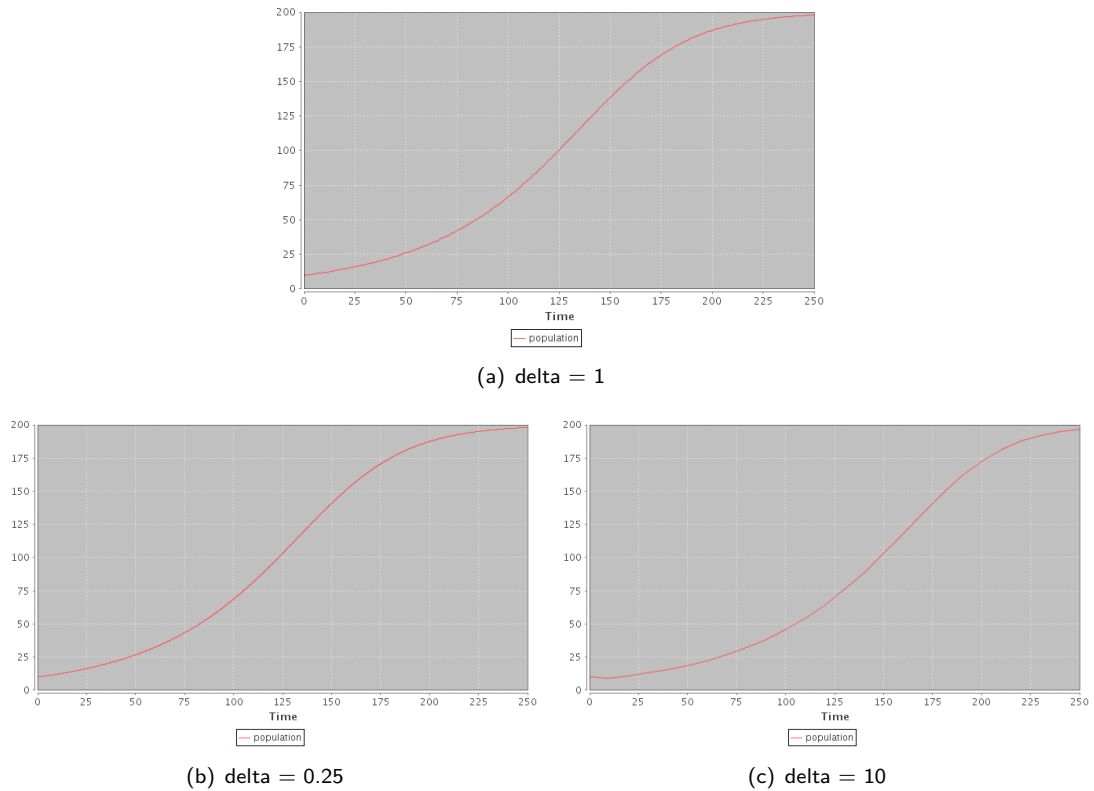


FIGURE 6.11: Simulator: the crowding model – synchronous engine

6.2.4.2 Asynchronous engine

Two observations can be made from the analysis of the simulation results. They are depicted in Figure 6.12

Figures 6.12(a)–(b) show the results of the simulation with a Δt of 1 and a time of 10 ms. In the case where agents have the same time (Figure 6.12(a)), the new values from stock agents are not correctly propagated due to the low value of the time. Computations are therefore based on older values that are in fact lower than they should be, which creates a less steep growth. Because of the lower values, the second phase is not triggered and only the exponential growth is observable. If the agents have different speeds (Figure 6.12(b)) then the two phases are observable and the final value of the population is well approximated even with a low time (more computations than required have been carried out which allows for the model to be correctly simulated).

Figures 6.12(c)–(d) represent the same situation as described above but with a δ of 10. In this case, in addition to the observations made for Figures 6.12(a)–(b), one can observe several kinds of oscillations that are artefact oscillations made by the configuration of the asynchronous engine. They are due to the combination of a high δ value and a low time value. In Figure 6.12(c), only the 'exponential growth' is observable. In Figure 6.12(d), the

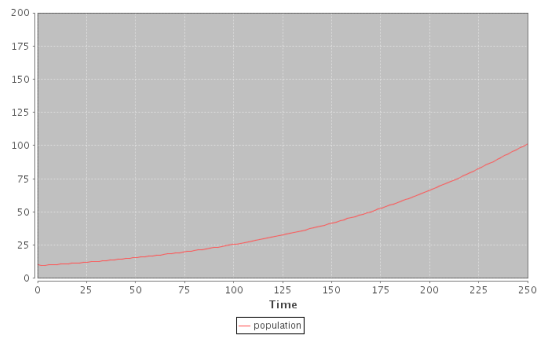
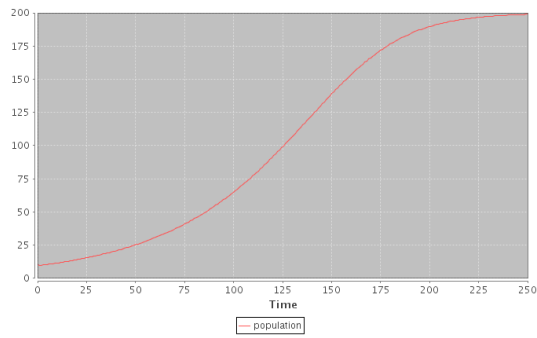
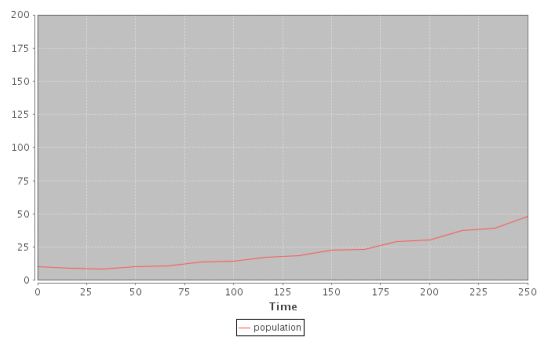
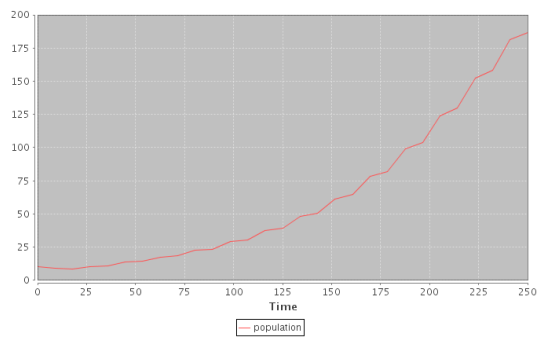
(a) $\delta = 1$, $\text{time} = 10$, $\text{random time} = \text{false}$ (b) $\delta = 1$, $\text{time} = 10$, $\text{random time} = \text{true}$ (c) $\delta = 10$, $\text{time} = 10$, $\text{random time} = \text{false}$ (d) $\delta = 10$, $\text{time} = 10$, $\text{random time} = \text{true}$

FIGURE 6.12: Simulator: the crowding model – asynchronous engine

final value of the population is more or less correctly approximated but the behaviour is not. It looks like an exponential growth that is as not steep as it should be. The second phase – goal-seeking – is not observable in both cases.

6.2.5 Flowers

In the example explained in 2.3.2.3, it was said that the flowers model has the same behaviour as the S-shaped pattern. With the same equations as in Section 2.3.2.3, simulations were carried out with both engines. The results from the Stella simulator are depicted in Figure 6.13.

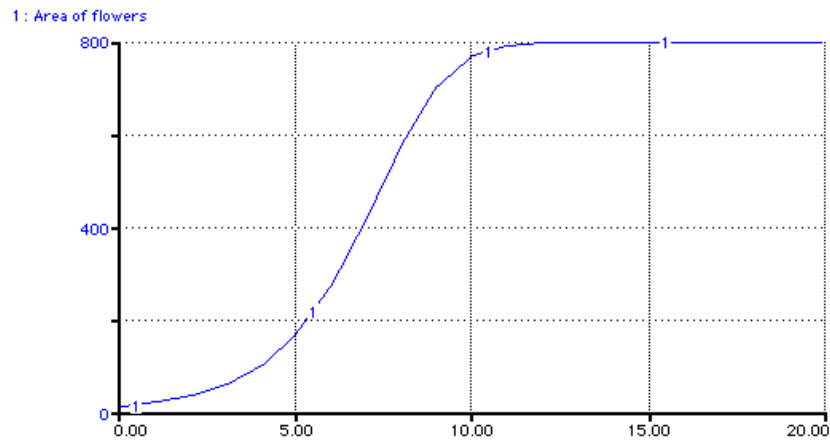


FIGURE 6.13: Stella: the flowers model (delta = 1)

```

area_of_flowers(t) = area_of_flowers(t - dt) + (growth - decay) * dt
INIT area_of_flowers = 10.0
INFLOW: growth = area_of_flowers * actual_growth_rate
OUTFLOW: decay = area_of_flowers * decay_rate
actual_growth_rate = intrinsic_growth_rate * growth_rate_multiplier
growth_rate_multiplier = - fraction_occupied + 1.0
fraction_occupied = area_of_flowers / suitable_area
decay_rate = 0.2
intrinsic_growth_rate = 1.0
suitable_area = 1000.0

```

6.2.5.1 Synchronous engine

Three possible values for the synchronous engine have been used to test the model.

Figure 17(a) represents the case where one computation is done by simulated time unit. The model reaches its dynamic equilibrium after 13 time units, this is therefore a good approximation of the shape of the model's behaviour and of the final value of the area of flowers.

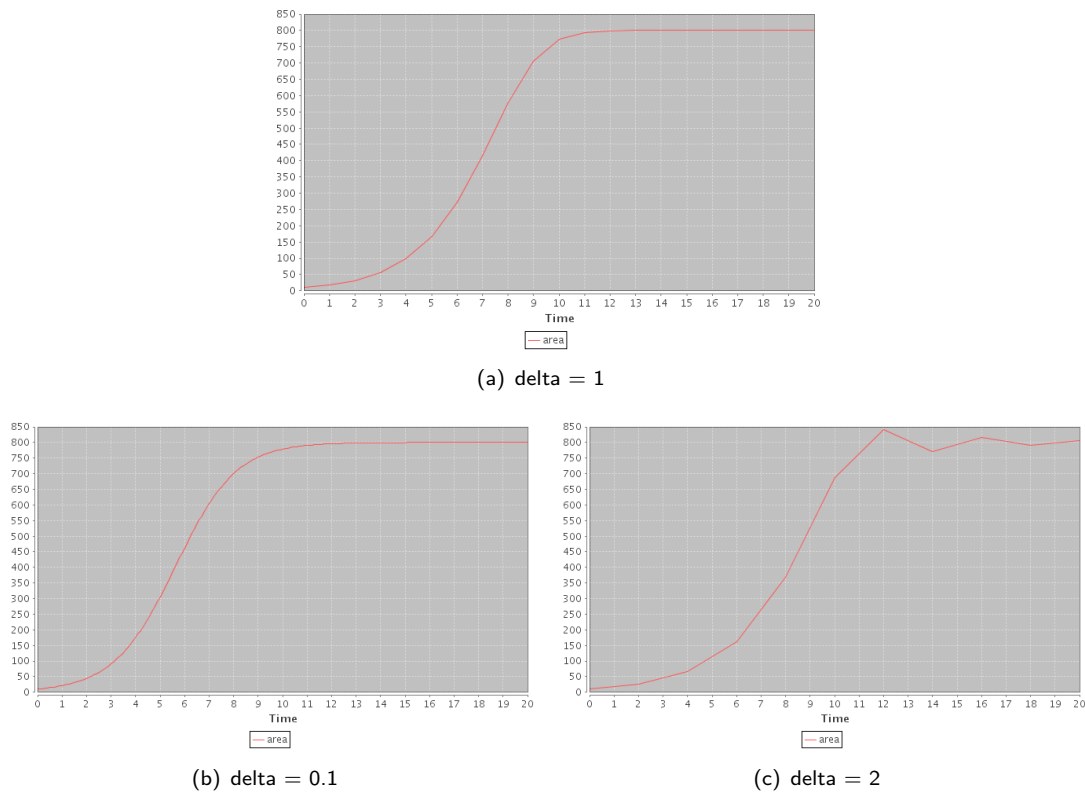


FIGURE 6.14: Simulator: the flowers model – synchronous engine

In Figure 17(b), one can observe the same approximation as in the first case but thanks to a smaller value of dt , the second phase – goal-seeking – is smoother than when only one computation is done per simulated time unit.

Figure 17(c) shows two characteristics of the variation of the dt value. Firstly because a computation is done every two simulated time units, the curve is not smooth at all. Secondly, artefact oscillations appear when the model reaches its limit of 800 flowers. This is due to the fact that changes are less frequent and are more important. The value thus overshoots its limit before being redirected under its limit too brutally and so on until it reaches the correct value.

6.2.5.2 Asynchronous engine

Three interesting behaviours can be spotted from the analysis of the simulation results. They are presented in Figure 6.15

Figures 6.15(a)–(b) represent the situation where a time of 10 ms is used. The propagation of the new values from the stock agent is not carried out correctly. This results in the use of older values for the computation of new values for the area of flowers. Because of the

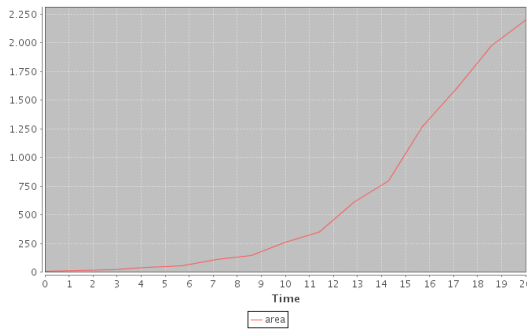
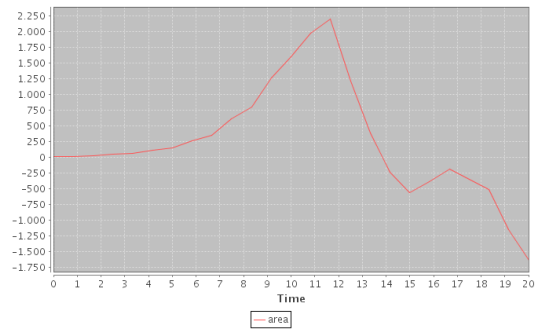
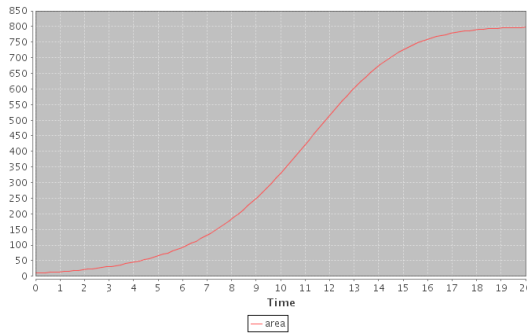
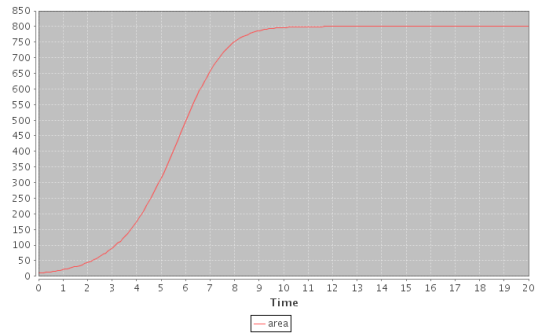
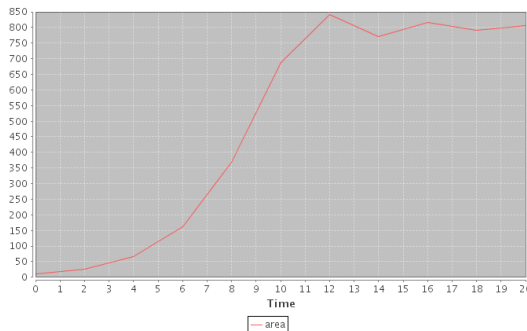
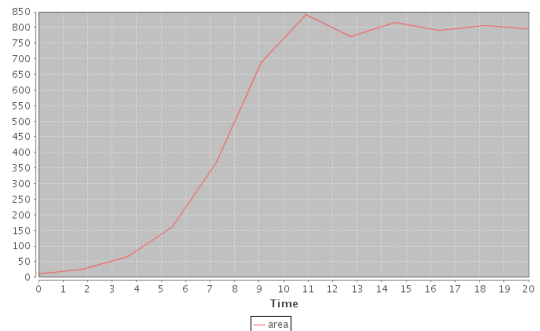
(a) $\delta = 1$, time = 10, random time = false(b) $\delta = 1$, time = 10, random time = true(c) $\delta = 0.1$, time = 10, random time = false(d) $\delta = 0.1$, time = 10, random time = true(e) $\delta = 2$, time = 100, random time = false(f) $\delta = 2$, time = 100, random time = true

FIGURE 6.15: Simulator: the flowers model – asynchronous engine

model's S-shaped behaviour with its exponential phase, and the computations of new values with older values than required, the model overshoots its limit to aim a total area of more than 2000 acres. Moreover, due to the delta of time added or withdrawn to/from the stock agents, the number of observations is greater than those required (see Figure 6.15(b)). This results in the brutal change of direction between step 11 and 12 where the model shoots under its natural limit of 0.

Figures 6.15(c)–(d) show that even with a low time that forces the stock agents to use older values than they should use, the model can be well approximated for both the behaviour and the final value. This is because 10 computations are carried out per time unit, allowing the model to avoid artefact delays. One can observe the difference between the two figures,

on one hand, the dynamic equilibrium is obtained at the end of the 20 simulated time units and on the other, it is obtained after only 9 steps. This is due to the different speed of the stock agents that implies more observations than required by the user.

Figures 6.15(e)–(f) depict the same situation described for the same delta value for the synchronous engine. The propagation can be done efficiently due to a correct value of time. The artefact oscillations come from the frequency of computations (1 every 2 simulated time unit). Changes are thus more brutal, the area overshoots its limit before being redirected below it and so on until the stock agent eventually reaches it.

6.2.6 Balancing loop

The balancing loop problem is the same example as in Section 4.6 but with no delay between the action and the current state. This pattern is also called the goal-seeking pattern where the current value tends to evolve towards the desired value. Results from the Stella simulation are depicted in Figure 6.16.

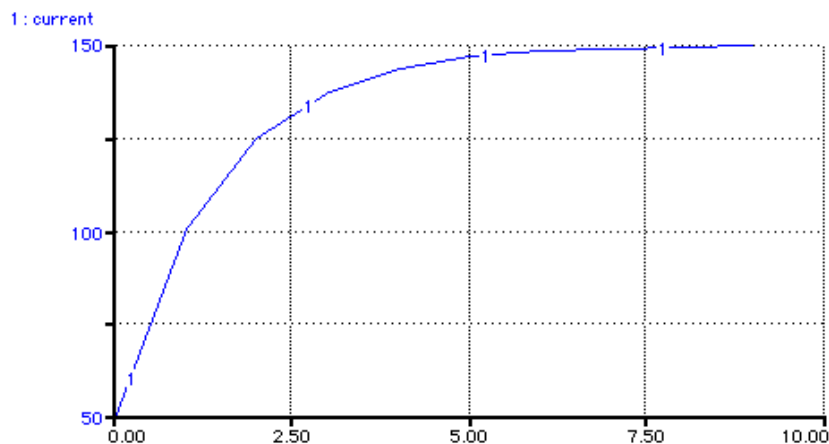


FIGURE 6.16: Stella: the balancing loop model (delta = 1)

This model has been tested with the following equations:

$$\text{Current value}(t) = \text{Current value}(t - dt) + (\text{Action}) * dt$$

$$\text{INIT Current value} = 50.0$$

$$\text{INFLOW: Action} = \text{Gap} / 2.0$$

$$\text{Desired value} = 150.0$$

$$\text{Gap} = \text{Desired value} - \text{Current value}$$

6.2.6.1 Synchronous engine

Three different delta values have been used to configure the synchronous engine. Results are depicted in Figure 6.17

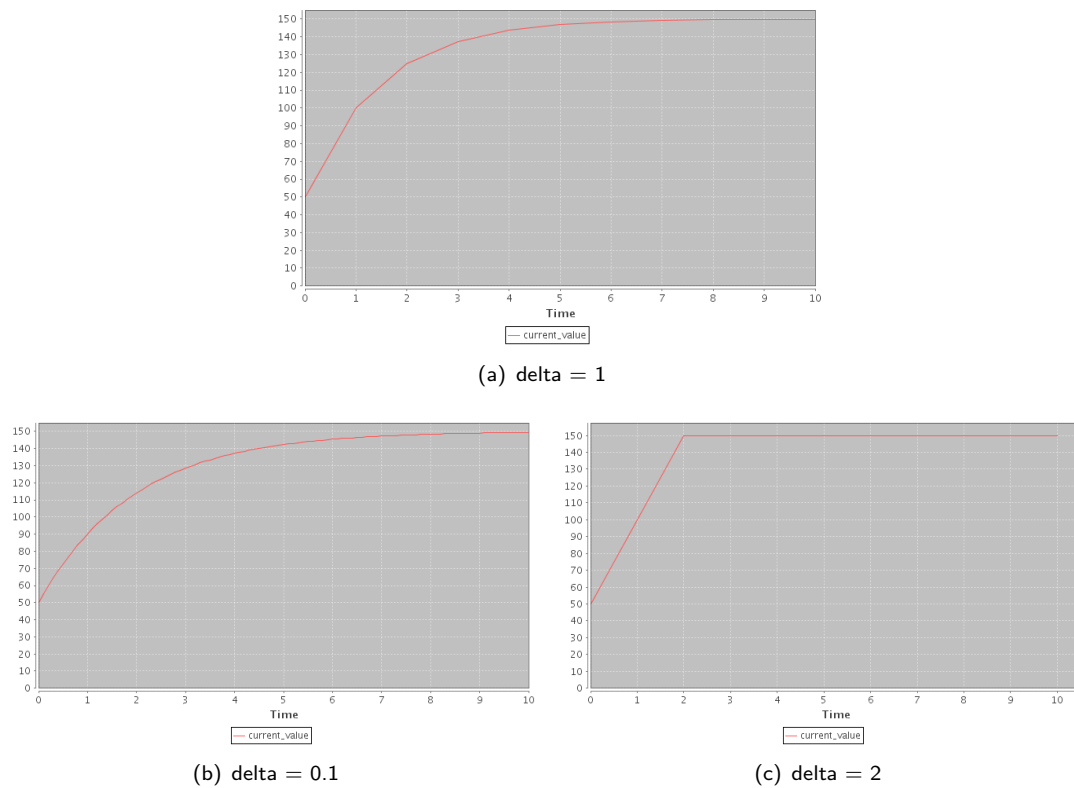


FIGURE 6.17: Simulator: the balancing loop model – synchronous engine

One can observe that for a δt of 1, the curve is not smooth. If the δt is smaller than 1 then the curve becomes smoother and smoother because more and more computations are carried out per time unit. If δt equals 2, then the equilibrium is obtained in only 2 simulated time units. In this case the approximation of the model's behaviour is quite rough but it gives the correct final value.

6.2.6.2 Asynchronous engine

Two observations can be made from the analysis of the simulation results for the asynchronous engine (see Figure 6.18).

Even with a delta of 1, the correct shape of the model's behaviour could not be obtained. This is due to a low value of the time allowing the stock agent to compute twice with the same value (the propagation did not change the value of the instantaneous agent soon enough). This can be observed in Figures 6.18(a)–(b).

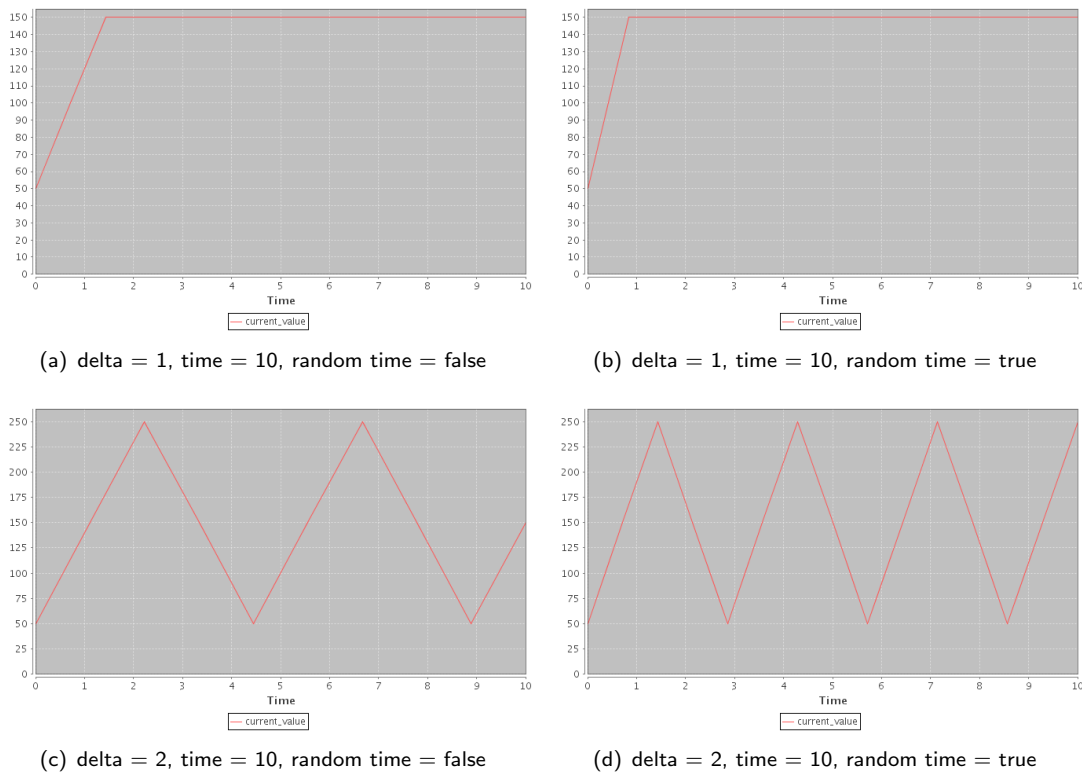


FIGURE 6.18: Simulator: the balancing loop model – asynchronous engine

Artefact oscillations can also appear for a higher value of dt and a smaller time value. In this case (see Figures 6.18(c)–(d)), the changes are computed once every two simulated time units, thus creating a more brutal change and allowing the model to overshoot its limit. This is due to the change computation frequency and the good values not being propagated to all the instantaneous agent. Oscillations are more frequent in the right-hand side figure because agents compute their changes more often due to a withdrawal of maximum 10% and because of the waiting thread that waits too long allowing the model to have more observations than required.

6.2.7 Knowledge diffusion

The knowledge diffusion is a kind of limited growth. There are two stocks: people with and without knowledge. But there are no sources or sinks which means that the initial amount of people without knowledge is progressively flowing towards the people with the knowledge thanks to the knowledge diffusion and the word of mouth factor. The results from the Stella simulation are depicted in Figure 6.19

This model, which was taken from [15], uses these equations:

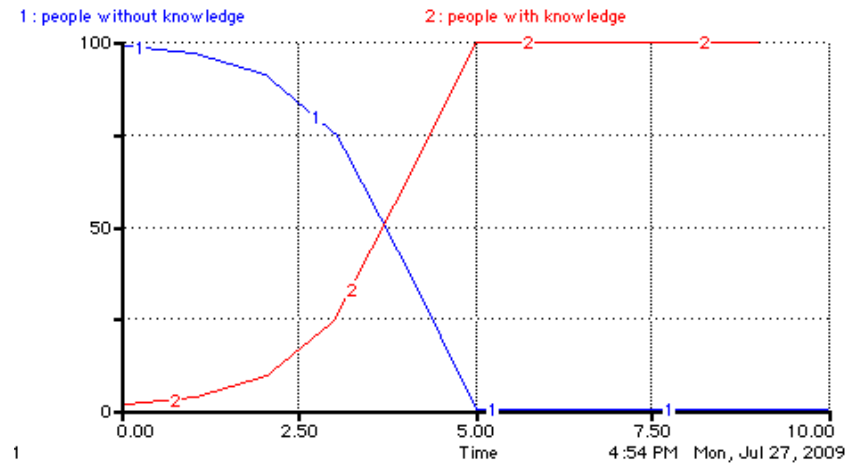


FIGURE 6.19: Stella: the knowledge diffusion model (delta = 1)

```

People without knowledge(t) = People without knowledge(t - dt) +
    (- Knowledge infusion rate) * dt
INIT People without knowledge = 99.0
OUTFLOW: Knowledge infusion rate = (Word of mouth factor
    * People with knowledge) * Gap to fill
People with knowledge(t) = People with knowledge(t -dt)
    + (Knowledge infusion rate) * dt
INIT People with knowledge = 1.0
INFLOW: Knowledge infusion rate =
    (Word of mouth factor * People with knowledge) * Gap to fill
Gap to fill = (People with knowledge + People without knowledge)
    - People with knowledge
Word of mouth factor = 0.02

```

6.2.7.1 Synchronous engine

Three possible values for the synchronous engine have been used for the simulations. Results are presented in Figure 6.20.

In this case, only delta time values of 1 or less have been used. The Stella simulation does not give any artefact oscillations for a delta of 1 but this is not true in the case of the simulator that has been built for this project. The dynamic equilibrium should be obtained at step 5, the fact that the approximation of the behaviour in the three cases is well simulated can be observed. In the last step, too many people are withdrawn from the population without the knowledge which means that the model undershoots below its natural limit (population under 0). In this model no sources or sinks are used. There are therefore 100 people at the

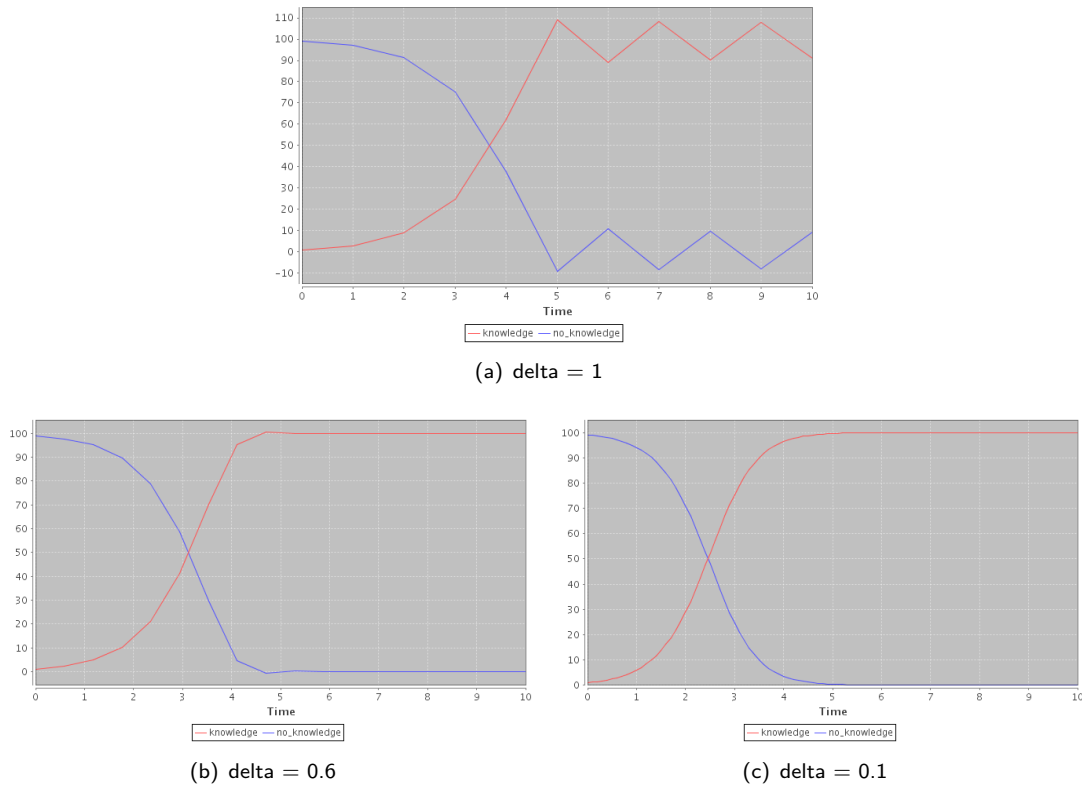


FIGURE 6.20: Simulator: the knowledge diffusion model – synchronous engine

beginning of the simulation. This is why having a negative population does not make sense. A dt of 0.1 is needed to avoid these artefact oscillations (other simulations have been carried out with other dt values and 0.1 is the first one to avoid oscillations).

6.2.7.2 Asynchronous engine

Only a few comments will be made about the results of the simulations. All the simulations produced interesting behaviours but they all have the following points in common. Figure 6.21 illustrates the following points in common:

- A small value of time allows the stock agents to compute with a 'wrong value' (propagation not quick enough), which creates brutal changes if an exponential characteristic is present (see Figure 6.21(a)).
- Different speeds for stock agents can create non regular artefact oscillations if they are present (see Figure 6.21(b)).
- Different speeds and small time values can create large non regular oscillations even for a small dt value (more computations are done by simulated time unit). This can lead to results like in Figure 6.21(c).

- Different speeds can add simulated people or withdraw people (NB: no sources or sinks are used, the conservation law must be respected). In this case, as the two stock agents do not compute their new values at the same time, some people may disappear from the system like in Figure 6.21(d) where the equilibrium for the population with the knowledge is 90 (it stays at 90 because the other population equals 0).

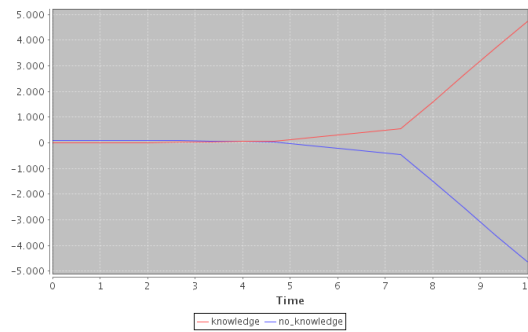
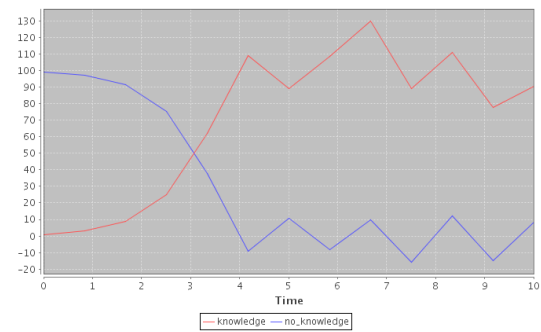
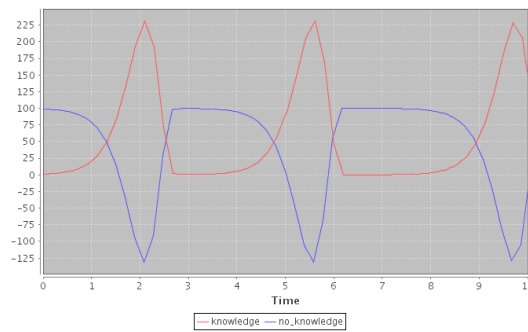
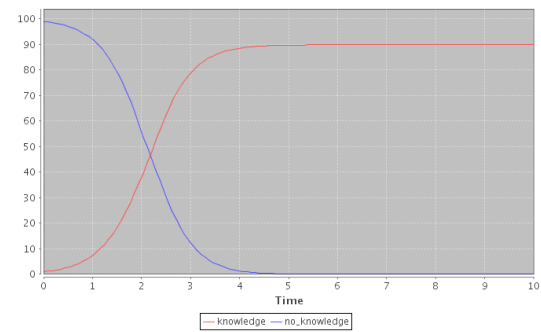
(a) $\delta = 1$, time = 10, random time = false(b) $\delta = 1$, time = 500, random time = true(c) $\delta = 0.1$, time = 10, random time = false(d) $\delta = 0.1$, time = 500, random time = true

FIGURE 6.21: Simulator: the knowledge diffusion model – asynchronous engine

6.2.8 Escalation

The escalation pattern explained in Section 4.7 shows a behaviour where the success of B follows the success of A with a certain delay. Results from the Stella simulator are depicted in Figure 6.22

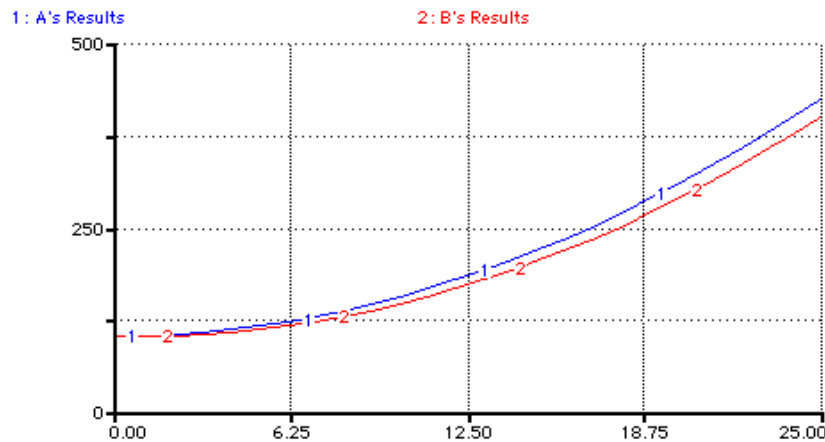


FIGURE 6.22: Stella: the escalation model (delta = 1)

The simulations used the following set of equations:

```

A's results(t) = A's results(t - dt) + (Activity by A) * dt
INIT A's results = 101.0
INFLOW: Activity by A = Results of A relative to B + 1.0
B's results(t) = B's results(t -dt) + (Activity by B) * dt
INIT B's results = 100.0
INFLOW: Activity by B = Results of A relative to B
Results of A relative to B = A's results - B's results
  
```

6.2.8.1 Synchronous engine

Three different values for the delta time parameter were used in order to simulate the model with the synchronous engine. Results are depicted in Figure 6.23.

One can observe what has been said about the variation of the dt value. The smaller the delta, the smoother the curve. For a dt of 5, the final values of both results are lower than in the two other cases due to changes that are less frequently computed (once every 5 simulated time units). In the three results, the approximation of the model's behaviour is good except for in the case of the final values.

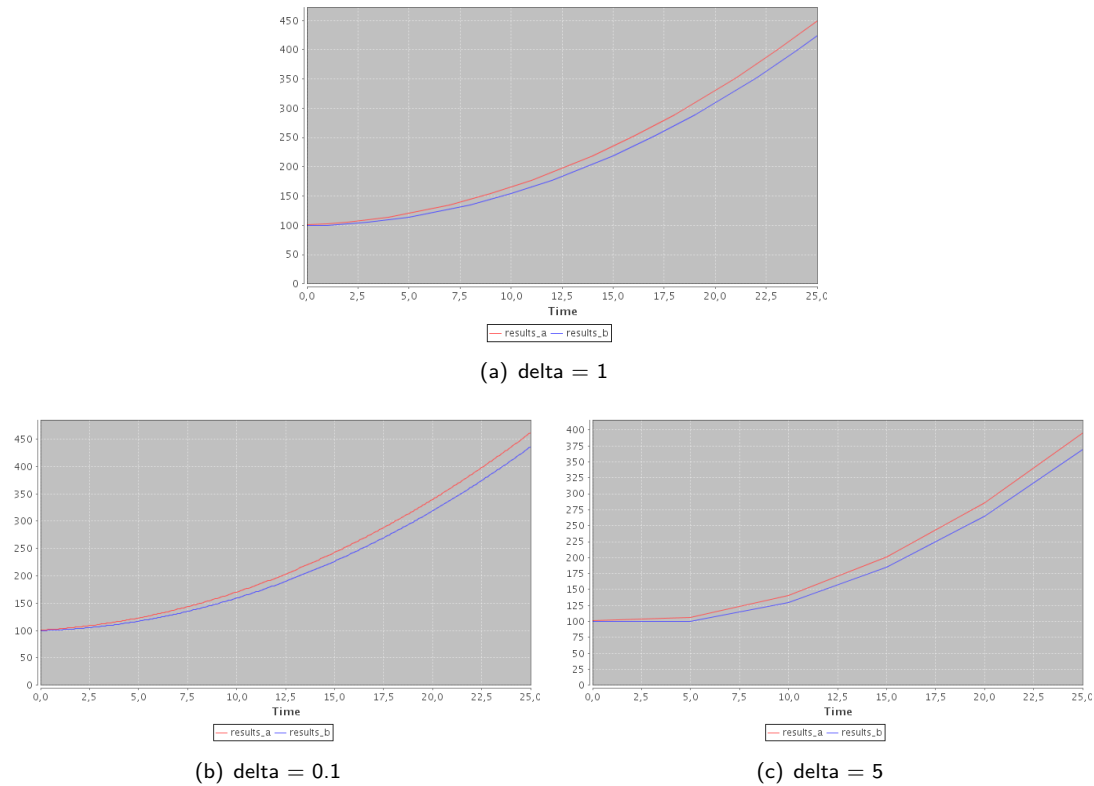


FIGURE 6.23: Simulator: the escalation model – synchronous engine

6.2.8.2 Asynchronous engine

Interesting behaviours appeared in the simulation results for the asynchronous engine. They are presented in Figure 6.24.

Figures 6.24(a)–(b) show artefact oscillations that are due to a low time that do not allow for a good propagation of new values from the stock agents. These agents therefore compute values on older values that should have been replaced by the propagation.

Figures 6.24(c)–(d) depict a strange behaviour. On the left-hand side the expected behaviour of the model appears when the stock agents have the same time. But on the other hand, on the right-hand side where both agents have different speeds, a phase change appears. That is, the model is supposed to have a quadratic behaviour, the difference between both agents is added to both agents but one of them receives an extra 10 (it has a larger success). If this extra had not been there, then the model would have been linear. Due to the different speeds, it appears that at a certain moment B catches up with A which gives relative results of 0. Both agents then stay together with a constant difference. One has to keep in mind the fact that in the worst case, both agents can have a time difference of 100 ms (one with $500 - 10\% = 450$ and one with $500 + 10\% = 550$). The quadratic phase will then re-appear and once again becoming linear.

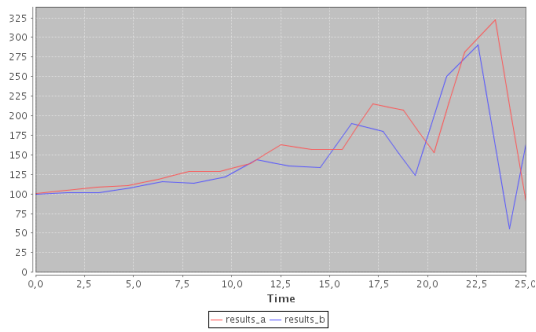
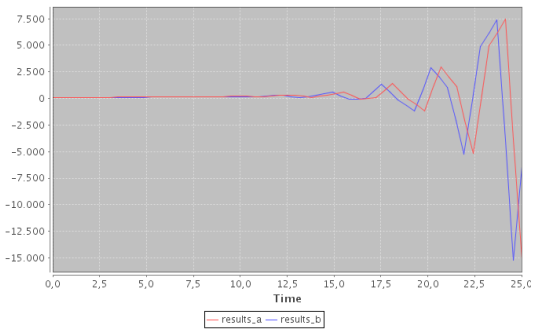
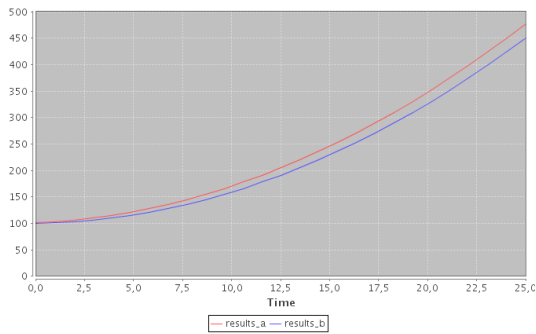
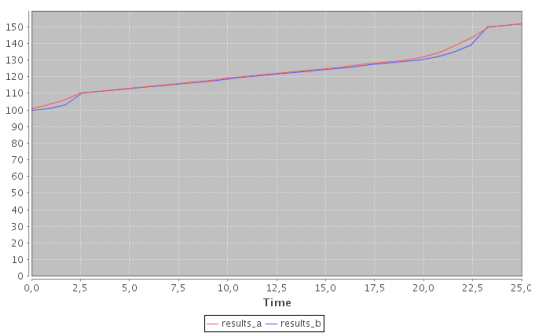
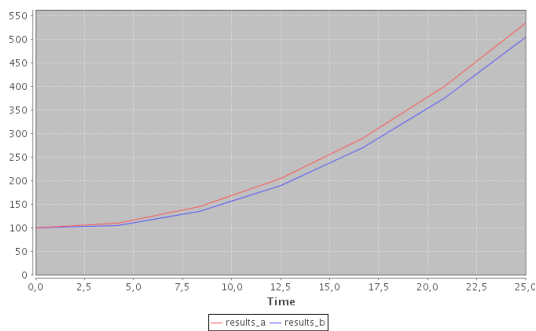
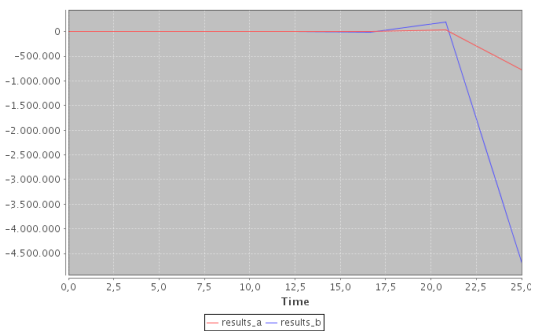
(a) $\delta = 1$, time = 10, random time = false(b) $\delta = 1$, time = 10, random time = true(c) $\delta = 1$, time = 500, random time = false(d) $\delta = 1$, time = 500, random time = true(e) $\delta = 5$, time = 500, random time = false(f) $\delta = 5$, time = 500, random time = true

FIGURE 6.24: Simulator: the escalation model – asynchronous engine

Figures 6.24(e)–(f) represent the situation where on one side both agents have the same time value and even if only 5 real computations have been carried out, the correct behaviour appears (propagation is done properly with a time of 500 ms). But on the other hand, when they have a different time, a strange and unknown behaviour appears about which only hypotheses can be formulated, a solution having not yet been found.

6.2.9 Success to the successful

This pattern was presented in Section 4.1. Two people start with the same amount of resources but progressively the success of one of them will decrease the success of the other one. Results from the Stella simulator are depicted in Figure 6.25.

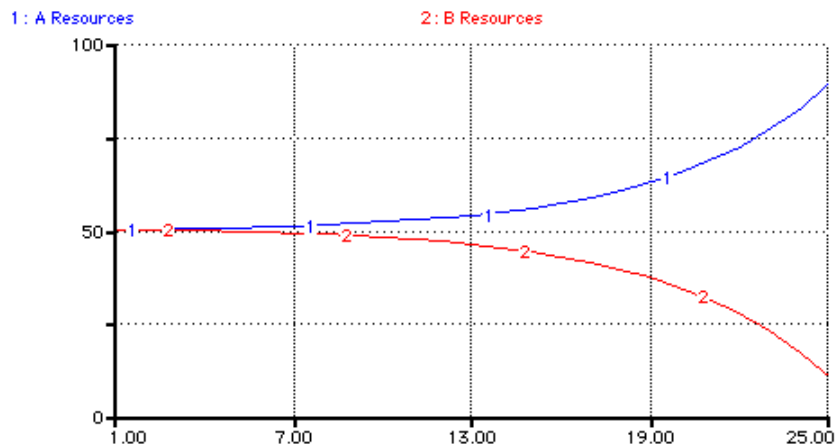


FIGURE 6.25: Stella: the success to the successful model ($\delta = 1$)

The following set of equations was used.

```

A resources(t) = A resources(t - dt) +
    (Resource allocation to A instead of B) * dt
INIT A resources = 50.0
INFLOW: Resource allocation to A instead of B =
    0.1 * Success of A relative to B
B resources(t) = B resources(t - dt) +
    (- Resource allocation to A instead of B) * dt
INIT B resources = 50.0
OUTFLOW: Resource allocation to A instead of B =
    0.1 * Success of A relative to B
Success of A = A resources + 1.0
Success of A relative to B = Success of A - Success of B
Success of B = B resources
  
```

6.2.9.1 Synchronous engine

Three different values were used for the delta time parameter of the synchronous engine. Results are presented in Figure 6.26.

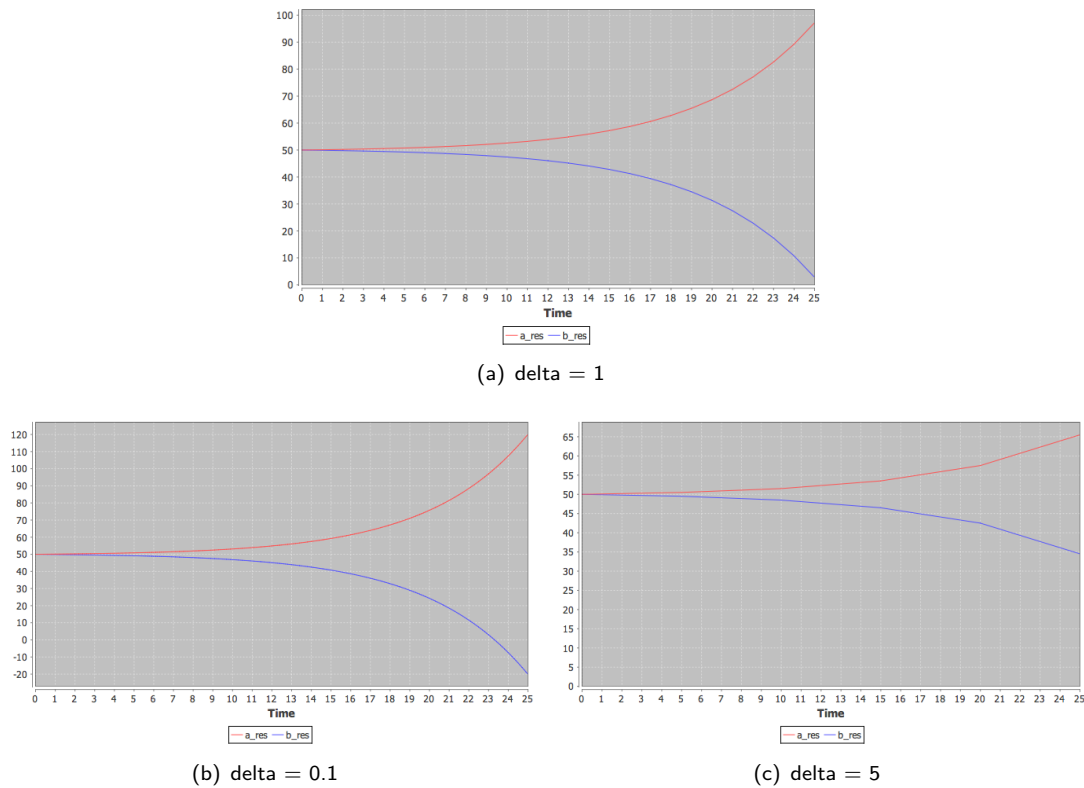


FIGURE 6.26: Simulator: the success model – synchronous engine

A dt of 0.1 gave a good approximation of the model's behaviour and of the final values of both results. If dt equals 1, both approximations are good but lower values are found. The same can be said for a dt equal to 5 where only one observation is made every 5 simulated time unit.

6.2.9.2 Asynchronous engine

Two observations have been made from the analysis of the simulation results with asynchronous engine.

Figures 6.27(a)–(b) show that if stock agents have different time, they can make more observations than required by the user. This results in a higher value than in the case where they have the same speed. In this case, both agents seem to have been withdrawn with about 10% and due to the waiting thread that waits for too long here, there are many more observations than actually required.

Figures 6.27(c)–(d) present the same situation as described above where more observations are present but in addition the two agents have different speeds. This results in one that grows faster than the decay of the other one.

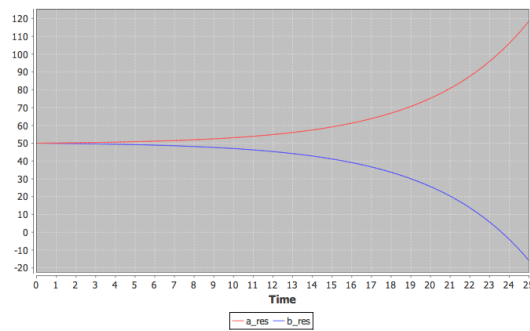
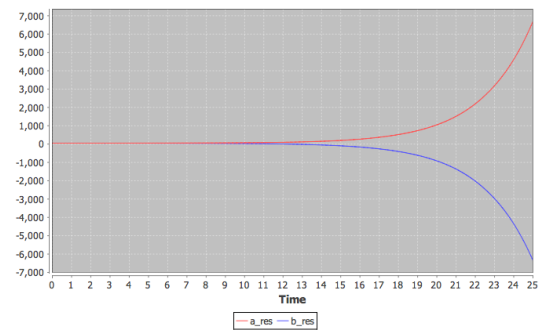
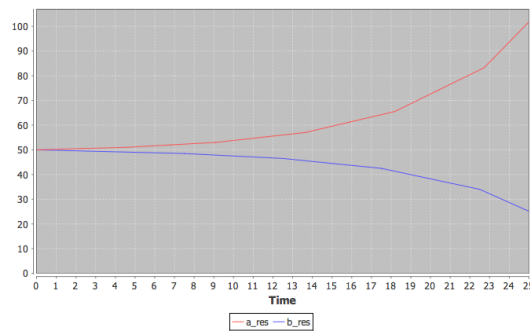
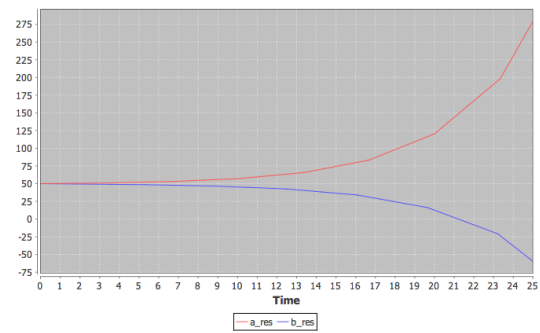
(a) $\delta = 0.1$, $\text{time} = 10$, $\text{random time} = \text{false}$ (b) $\delta = 0.1$, $\text{time} = 10$, $\text{random time} = \text{true}$ (c) $\delta = 5$, $\text{time} = 10$, $\text{random time} = \text{false}$ (d) $\delta = 5$, $\text{time} = 10$, $\text{random time} = \text{true}$

FIGURE 6.27: Simulator: the success model – asynchronous engine

Chapter 7

Conclusion

As an introduction to this conclusion, an example of an application of the theory on feedback loops presented in this project is given. Let us consider a protocol used every day by millions of people all over the world. The 'Transmission Control Protocol' known as TCP can be considered as an application using feedback loops. The heart of the TCP has two interacting feedback loops (through management) that implement a reliable byte stream transfer protocol with congestion control (see Figure 7.1).

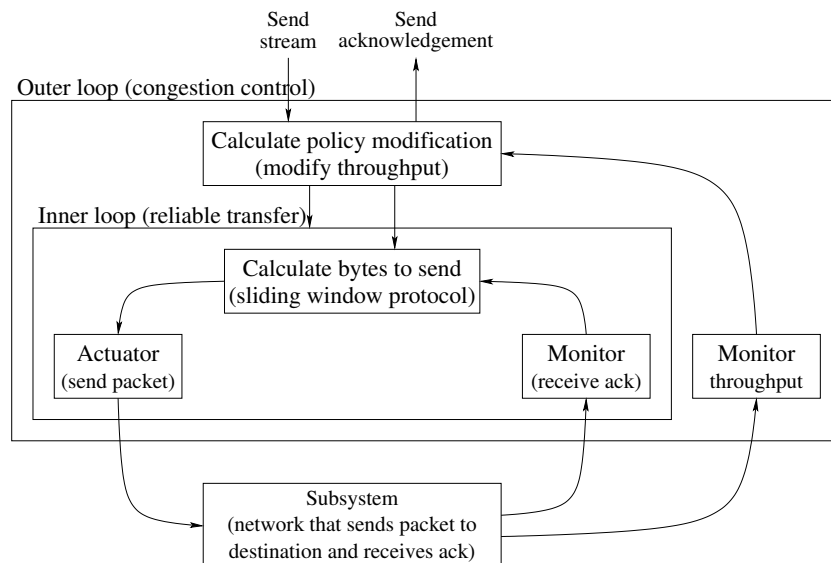


FIGURE 7.1: TCP as a feedback system

The inner loop controls the reliable transfer of a stream of packets. This loop sends packets and monitors the acknowledgements that these packets have arrived successfully to their destination. It also controls the sliding window that is a sort of a negative feedback loop using monotonic control.[32]

The outer loop is responsible for the congesting control. It looks at the throughput of the system and can act in two different ways: it can change the policy of the inner loop (if the rate of acknowledgements decreases then it reduces the size of the sliding window) or it can change the inner loop itself (if the rate drops to zero then it terminates the inner loop and aborts the transfer).

7.1 Modelling a feedback system for software applications

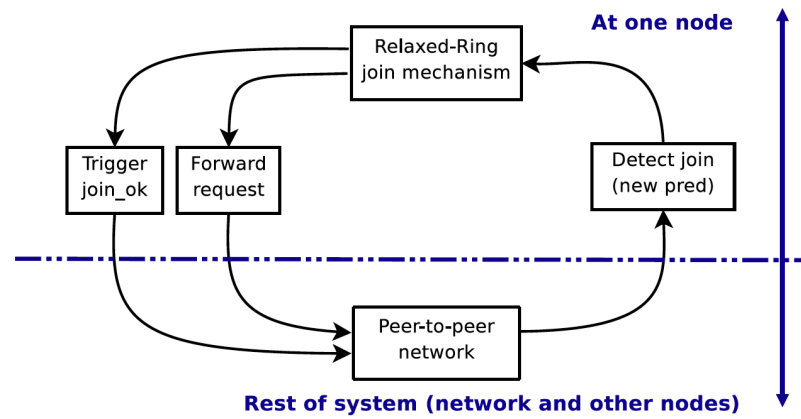
When an IT team is asked to build a new application that consists of multiple components that interact with each other, they should apply the methodology described in Chapter 2 and build a representation of the system with the different elements and their interaction. This methodology must be adapted to the needs of this IT team but the guidelines are quasi identical, that is, they need to identify all of the required components in order to formulate hypotheses on the dynamic of the system, to construct a formal representation of this model thanks to the causal loop diagrams, to test the credibility of the model, to test the sensitivity of the model, to test the impact of policies, and to try to validate their model.

This modelling technique can be used at every level of the architecture of the software. That is, from the high-level modules to the implementation of components of these modules. Feedback loops are not pieces of code that you put straight into your application. They are high-level design rules to help the designer to understand the behaviour of his/her application.

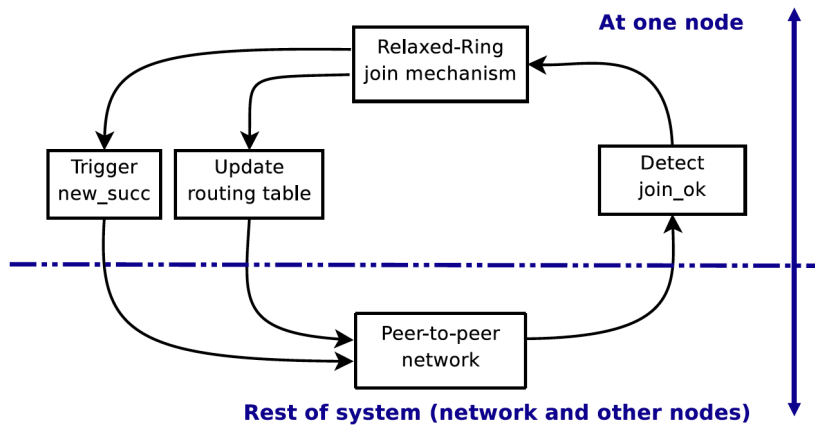
For example, feedback loop structures have been used in the project SELFMAN to build applications with self-management characteristics. The gPhone application developed by the UCL proposed several feedback loop structures used to build automatic behaviours.[7]

The gPhone application is built on top of structured overlay networks (SON) providing distributed hash tables (DHT - relaxed ring). The relaxed-ring is related to self-healing and fixes the ring when a peer leaves the network. The three feedback loops present in Figure 7.2 depict the feedback loop structures that represent automatic behaviours and that are embedded in a peer. In this example, the joining process of a peer is described. At least three different peers are needed for this process: a *successor*, a *predecessor*, and a *new peer*.

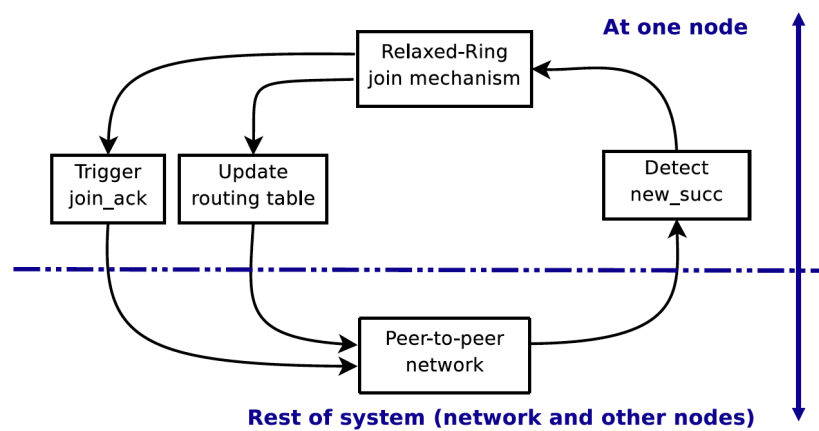
The sequence is as follows: the new peer asks the successor to join the ring, the successor accepts this offer. At this moment, the new peer tells its predecessor that it is its new successor, then this predecessor sends to the successor of the new peer an acknowledgement that the new node has joined the system. Figure 7.2(a), from [7], represents the *successor's* automatic behaviour, Figure 7.2(b) the *new node's* automatic behaviour, and Figure 7.2(c) the *predecessor's* automatic behaviour.



(a) New peer joins as new predecessor of the current responsible of its key



(b) New peer joins as new predecessor of the current responsible of its key



(c) New peer joins as new predecessor of the current responsible of its key

FIGURE 7.2: Ring maintenance as a feedback system

Figure 7.3, taken from [7], represents the feedback loops of the failure recovery where when a peer p crashes, its predecessor reacts to the crash event and sends an event to the successor of the crashed peer to create a link. This successor responds with an acknowledgment. The numbers of the figure represent the sequence corresponding to the scenario just described.

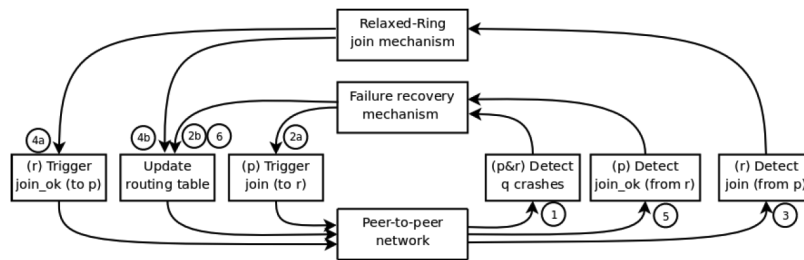


FIGURE 7.3: Failure recovery as a feedback structure

Figure 7.4, taken from [7], shows the automated behaviour set to manage the finger table for routing on the relaxed-ring. The two feedback loops represent the failure handling and the correction-on-use.

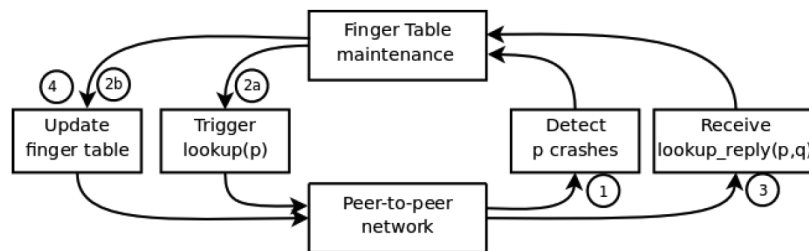


FIGURE 7.4: Finger maintenance with failure detection and correction-on-use

The SELFMAN project also proposed guidelines to evaluate qualitatively and quantitatively the autonomic features of applications.[7, 33]

7.2 Feedback structures and design rules

In Chapters 3 and 4, a non exhaustive list of feedback structures with multiple interacting feedback loops were studied. These patterns could help the IT team to understand the behaviour of the system they have created.

The patterns in Chapter 4 are especially used in System Thinking where systemic specialists analyse problems in organisations and confront the members of the organisation with these patterns. They work with these patterns like a debugging system. The archetype tree,

presented in Section 4.15, is a kind of debugging tree where, by following the branches, one can reach a special feedback structure and understand its dynamic.

System thinkers have also proposed solutions for sorting out the problem caused by the patterns presented in Chapter 4. For example, for the 'Shifting the Burden' they proposed:

"In trying to understand a 'Shifting the Burden' situation, start the same way as you would with 'Fixes that Backfire': What is the problem symptom which you tried to fix? What is the fix you tried? What were the unexpected results, and how did they affect the original source or root cause of the problem?

Then comes the leap: What alternative solutions might you have tried, if the quick-fix avenue were not available to you? Would any of those alternatives have been more fundamentally satisfying? And how do you know that these corrective actions would truly address the source of the problem?

Use the archetype as a tool for inquiry, not as a tool for advocacy. There is a temptation to assume that your preferred solution, whether you tried it or not, is the 'right' solution – and to simply write that solution into the slot. In many cases, top management sees one solution as fundamental, while front-line workers see another, and marketing sees a third. Each 'fundamental solution' would suggest a different sense of appropriate leverage. That's why, especially in teams, it's important to suspend your preconceptions about which 'solution' fits the slot, and instead try to explore, as an interfunctional group, the deeper sources of the problem. This type of sustained dialogue often unearths mental models and cultural assumptions as the real root causes of problem."[19]

Some patterns are just pure common sense, like this 'Shifting the Burden' pattern, but others can be very useful in analysing problems. For example, imagine two threads sharing the same resources. Monitoring these two threads reveals the following behaviour: both activities are growing but their performance is decreasing. This situation corresponds to the 'Tragedy of the Commons' where the common activity eventually is too large for the common resources to support.

7.3 Further work

What should be done now is, for example, to start from an existing program and to analyse it on order to discover whether strange behaviours occur or not. If possible, it might be interesting to apply what has been discussed in this project in order to construct the application model following the modelling techniques and to try to match this model's behaviour with

the actual application behaviour. Then, if problems arise, it would be possible to identify the corresponding pattern(s) and to find a solution if necessary.

Another kind of classification of the feedback structures can be done. That is, feedback structures with the 'same behaviour' can have a different number of feedback loops. It might be interesting to see if it is possible to find a certain number of 'canonical' patterns within feedback loops (the minimal number of loops necessary to obtain a given behaviour). For example, the regulation pattern can be found in 1-loop systems (negative feedback loop) as well as structures with more loops. In general, these extra loops reinforce the regulation (the push-pull pattern). Nevertheless, the number of loops and the number of elements in loops is left to the modeller and to the level of abstraction used in the study.

Yet another classification process might be interesting in order to be able to quickly classify feedback structures in three categories: stable, unstable, unknown stableness systems. For example, in the feedback structures simulated in Chapter 6, one can observe that the escalation pattern is very sensitive to the parameter value: the model's behaviour is at times very unexpected and only hypotheses can be made. This model could be clearly identified as unstable (two negative feedback loops that can act as a big positive feedback loop). On the other hand, a pattern like the limited growth can be seen as a stable system that resists variations of parameters values (a negative and a positive feedback loop where the positive one is regulated by a negative feedback loop). It might be interesting to find a way to classify feedback patterns by looking at their structure without simulating them.

Bibliography

- [1] IBM. *Autonomic Computing: IBM's Perspective on the State of Information Technology*". 2001.
- [2] J. O. Kephart and D. M. Chess. *The Vision of Autonomic Computing*. 2003.
- [3] Selfman project, . URL http://www.ist-selfman.org/wiki/index.php/SELFMAN_Project.
- [4] P. Van Roy, S. Haridi, A. Reinefeld, J.-B. Stefani, R. Yap, and T. Coupaye. *Self Management for Large-Scale Distributed Systems: An Overview of the SELFMAN Project*. 2006.
- [5] SELFMAN Project. *D5.7: Guidelines for bulding self-managing applications*. 2009.
- [6] A. Al-Shishtawy, J. Höglund, K. Popov, N. Parlavantzas, V. Vlassov, and P. Brand. *Distributed Control Loop Patterns for Managing Distributed Applications*.
- [7] SELFMAN Project. *D5.4A: Qualitative evaluation of autonomic features of Selfman applications*. 2009.
- [8] Wikipedia. System dynamics. URL http://en.wikipedia.org/wiki/System_dynamics.
- [9] Y. Barlas. *System Dynamics: Systemic Feedback Modeling for Policy Analysis*. Encyclopedia of Life Support System, 2002.
- [10] J. D. Sterman. *System Dynamics Modeling: Tools for Learning in a Complex World*. California Management Review, 2001.
- [11] N. Wiener. *Cybernetics, or Control and Communication in the Animal and the Machine*. MIT Press, 1948.
- [12] G. S. Brown and D. P. Campbell. *Principles of Servomechanisms*. Wiley, 1948.
- [13] Wikipedia. Causality, 2009. URL <http://en.wikipedia.org/wiki/Causality>.

- [14] G. B. Hirsch, R. Levine, and R. L. Miller. *Using system dynamics modeling to understand the impact of social change initiatives*. Springer Science+Business Media, 2007.
- [15] R. J. Madachy. *Software Process Dynamics*. Wiley, 2008.
- [16] A. Ford. *Modeling the Environment: An Introduction to System Dynamics Modeling*. Modeling the Environment: An Introduction to System Dynamics Modeling of Environmental Systems. Island Press, 1999.
- [17] J.-R. Kim, Y. Yoon, and K.-H. Cho. *Coupled Feedback Loops From Dynamic Motifs of Cellular Networks*. Biophysical Journal, 2008.
- [18] P. Van Roy. *Self Management and the Future of Software Design*. Electronic Notes in Theoretical Computer Science, 2006.
- [19] P. Senge. *The Fifth Discipline Fieldbook: Strategies and Tools for Building a Learning Organization*. Nicholas Brealey Publishing, 1994.
- [20] X.-P. Zhang, Z. Cheng, F. Liu, and W. Wang. *Linking fast and slow positive feedback loops creates an optimal bistable switch in cell signaling*. The American Physical Society, 2007.
- [21] J. Heo, D. Henriksson, X. Liu, and T. Abdelzaher. *Integrating Adaptive Components: An Emerging Challenge in Performance-Adaptive Systems and a Server Farm Case-Study*. 28th IEEE International Real-Time Systems Symposium (RTSS 2007), 2007.
- [22] T. Kobayashi, L. Chen, and K. Aihara. *Modeling Genetic Switches with Positive Feedback Loops*. Journal of Theoretical Biology, 2003.
- [23] Strongly connected component, . URL http://en.wikipedia.org/wiki/Strongly_connected_component.
- [24] P. L. Kunsh, M. Theys, and J. P. Brans. *The importance of systems thinking in ethical and sustainable decision-making*. Springer Verlag, 2007.
- [25] W. Braun. *The System Archetypes*. 2002.
- [26] 2004. URL <http://www.systems-thinking.org/>.
- [27] The mozart programming system, . URL <http://www.mozart-oz.org/>.
- [28] P. Van Roy and Seif Haridi. *Concepts, Techniques, and Models of Computer Programming*. MIT Press, 2004.
- [29] JFree. Jfreechart. URL <http://www.jfree.org/jfreechart/>.
- [30] Java universal network/graph framework, . URL <http://jung.sourceforge.net/>.

- [31] Stella - systems thinking for education and research, . URL <http://www.iseesystems.com/software/Education/StellaSoftware.aspx>.
- [32] P. Van Roy. *Overcoming Software Fragility with Interacting Feedback loops and Reversible Phase Transitions*. Electronic Workshops in Computing, The British Computer Society, 2008.
- [33] SELFMAN Project. *D5.4B: Quantitative evaluation of autonomic features of Selfman applications*. 2009.
- [34] V. Gascogne. *Les modèles de dynamique des systèmes: des outils pédagogiques pour une aide à la gouvernance des systèmes ?* Journées Asfcet, 2006.
- [35] K. A. Stave. *A system dynamics model to facilitate public understanding of water management options in Las Vegas, Nevada*. Journal of Environmental Management, 2003.
- [36] G. Yücel and C. M. Chiong Meza. *Studying transition dynamics via focusing on underlying feedback interactions: Modelling the Dutch waste management transition*. Springer, 2008.
- [37] K. Madani and M. A. Mariño. *System Dynamics Analysis for Managing Iran's Zayandeh-Rud River Basin*. Springer Science+Business Media, 2008.
- [38] P. Pfaffenbichler, G. Emberger, and S. Shepherd. *The Integrated Dynamic Land Use and Transport Model MARS*. Springer Science+Business Media, 2007.
- [39] K. O. Jensen, P. E. Barnsley, J. Tortolero, and N. Baxter. *Dynamic modelling of service delivery*. BT Technology Journal, Vol. 24, No.1, 2006.
- [40] I. Winz, G. Brierly, and S. Trowdale. *The Use of System Dynamics Simulation in Water Resources Management*. Springer Science+Business Media, 2008.
- [41] M. H. Friedman. *Principles and Models of Biological Transport, Ch. 8 Regulation and Transport*. Springer Science+Business Media, 2008.
- [42] B. Campbell. *Process Failure in a Rapidly Changing High-Tech Organization: A System Dynamics View*. Springer-Verlag Berlin Heidelberg, 1998.
- [43] H. Purnomo, Y. Yasmi, R. Prabhu, S. Hakim, A. Jafar, and Suprihatin. *Collaborative Modelling to Support Forest Management: Qualitative Systems Analysis at Lumut Mountain, Indonesia*. Small-scale Forest Economics, Management and Policy, 2003.
- [44] T. R. Rohieder, D. P. Bischak, and L. B. Baskin. *Modeling patient service centers with simulation and system dynamics*. Springer Science+Business Media, 2006.

- [45] S. Sirois and L. Martin Cloutier. *Needed: system dynamics for the drug discovery process*, volume 13. Drug Discovery Today, 2008.

Appendix A – Pattern examples

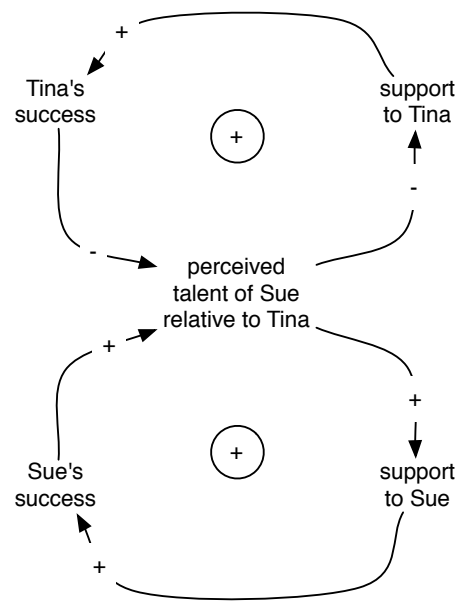
In this appendix, examples for the different patterns presented in Chapter 4 are presented:

Examples for the following patterns:

- Success to the Successful
- Limits to Growth
- Tragedy of the Commons
- The Attractiveness Principle
- Growth & Under Investment
- Balancing with Delay
- Escalation
- Indecision
- Fixes that Backfire
- Accidental Adversaries
- Shifting the Burden
- Addiction
- Drifting Goals

Success to the Successful

Self-Fulfilling Prophecy

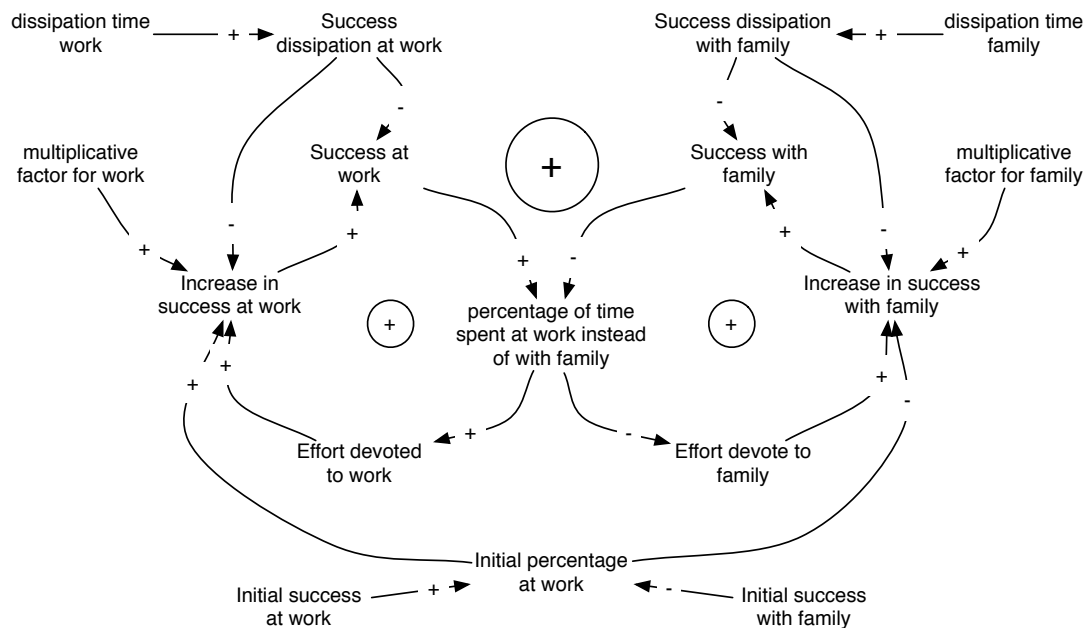


Number of loops	2
Pattern	Success to the Successful
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	how easy it is to influence a result based on an initial belief, without ever realizing the result is being influenced by the belief.
Reference	[26]

"Being that I am responsible for two individuals, Sue and Tina, it may be that I perceive that Sue is more talented and capable than Tina. Realize that I may or my not be consciously aware of this belief. Because of this perceived talent of Sue relative to Tina I tend to provide more support to Sue. This support may be in terms of planning, coaching, etc. This support of Sue enhances the success of Sue, which simply serves to reinforce the perceived talent of Sue relative to Tina. While this perception influences me to provide enhanced levels of support of Sue it influences me to provide less support to Tina. This reduced support to Tina hinders the success of Tina, which simply serves to further enhance the perceived talent of Sue relative to Tina.

This situation results in two reinforcing loops both driving the system in the same direction, the success of Sue relative to the success of Tina. And, isn't it interesting how the situation turned out to prove that my initial perception was in fact true!"

Self-fulfilling prophecies



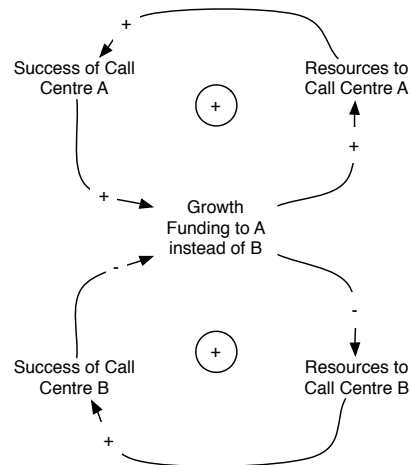
Number of loops	3
Pattern	Success to the Successful
Type	Stock-and-flow
Origin	Model from existing model
Validation	Yes - behavior analysis (success at work vs success with family)
Context	Development of a mimetic crisis, as the violence of everyone against everyone is perceived as doomed to happen.
Reference	[24]

"Simple personal case where a conflict exists between allocating the resources in time between work and family (three reinforcing positive loops, including the combined eight-shaped loop).

The expectation that some evolution will follow some suspected path strengthens this expectation by taking away resources from equally possible and perhaps more favorable or less dramatic alternatives.

The good and ethical decision is here to think of co-operation rather than competition."

Call Centres



Number of loops	2
Pattern	Success to the Successful
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	Call centres success
Reference	[25]

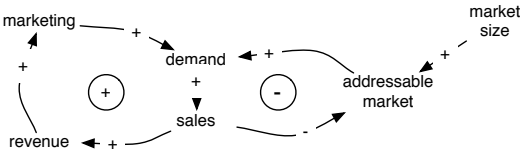
"Two call centers are established in different parts of the country. Some rationale for resource allocation results in one of them experiencing better performance than the other. Not only is the lesser performer looked down upon, but its lack luster performance is cited as a sound rationale not to put any more resources into it.

Managers should exercise caution before quickly concluding that intrinsic merit is a complete explanation for good performance. This archetype may also reveal in depth the axiom that we manage what we measure. Stated otherwise, are the measurements that have historically been used to assess performance still relevant? Are they still accurate? Is there an increased level of noise in the data that is used for decisions making? Have delays in information caused managers to reach conclusions that appear to favor one person, department or product over another, when in fact refining measurements to better reflect what customers think, want and/or need would offer a different view of performance?

Finding itself bogged down in this archetype can also lead to the erosion of innovation and change. Concluding that this is our best product and we have to stay with it because it is the best performer (at present) can obscure a long, slow decline in the products position in the market. Taking a fresh look at marginal performers, in a new light, may lead to insights that can rejuvenate an organizations approach to its internal management, its products or to its customers."

Limits to Growth

Failure of Success



Number of loops	2
Pattern	Limits to Growth
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	Point out how success can deceive us into believing that we have found the answer to success, an answer which often eventually leads to failure
Reference	[26]

"We begin with a marketing effort which interacts with the addressable market to add to demand. Since the marketing effort does in fact produce an increase in demand we make the assumption that it represents a viable marketing effort.

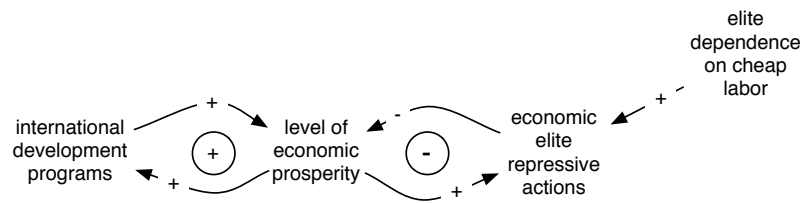
The point of concern here is whether the marketing activity is actually increasing the market size or simply interacting with the addressable market influencing it to purchase sooner than it would have. This is often the case with promotions, and that will be considered to be happening in this example.

As marketing interacts with the addressable market to add to demand the increased demand adds to sales. Sales then adds to revenue which adds to marketing. As such, as marketing appears to increase demand resulting in more sales and more revenue, the tendency is to do even more marketing. This reinforcing loop drives the growth of revenue.

While the reinforcing loop drives the increase in sales, sales is subtracting from the addressable market. The addressable market continues to interact with marketing and add to demand, yet to a smaller and smaller extent as the addressable market decreases. At some point the decrease in addressable market will be such that there will no longer be a growth in demand. At this point the addressable market has been addressed, and there's nothing left to address.

Yet, since marketing has continue to increase demand the natural tendency is to increase the marketing effort even more to spur demand. Continued success has fostered a belief that marketing is the answer to generating demand. When this increase in marketing doesn't increase demand the marketing organization is often quite confused. It has been said that you can lead a horse to water but you can't make them drink. Well you can't lead a nonexistent horse anywhere."

When They're Down, Keep 'em Down



Number of loops	2
Pattern	Limits to Growth
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	Limits to Growth in international development policy
Reference	[26]

Condition: level of economic prosperity in a rural region.

Growing Action: international development programs (UN, religious groups, USAID, etc)
As more development programs come in, the level of economic prosperity of the region begins to improve.

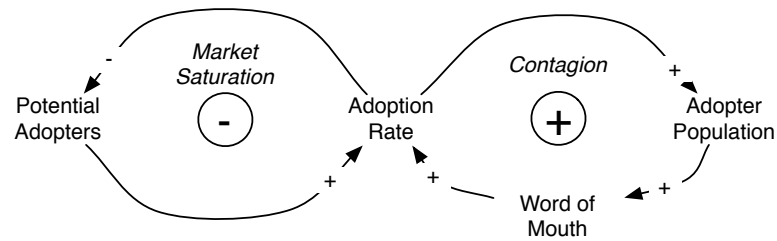
Slowing Action: repressive actions on the part of the economic elite against the cooperatives that the rural people have formed.

Limiting Condition: The economic and political elite of a country (this was certainly true for Guatemala) actually depend upon the people in rural areas staying "underdeveloped". These rural people provide a cheap labor force to the city businesses, the plantations, and the city elite as housemaids, cooks, etc. The LAST thing the political elite want is for the rural villages to become prosperous and stop providing a cheap labor force. In Guatemala, the Army eventually went out into the villages with lists of all the heads of cooperatives and killed them for being "communists".

This is all very well documented in Cultural Survival and other areas.

How to get around the "limiting condition"? We need international pressure to be brought to bear on the national elite along with some assistance in making the activities of this elite more productive so that they can compete in national and international markets without the repressive policies against ethnic groups within their borders."

Adoption of a new product

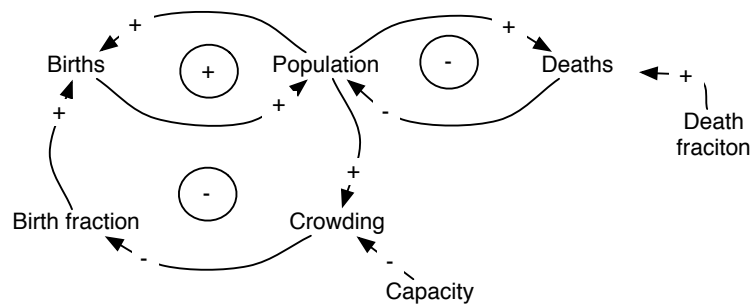


Number of loops	2
Pattern	Limits to Growth - S-shaped growth
Type	Stock-and-flow
Origin	Model from existing model
Validation	Yes - behavior analysis and comparison with reality through simulation
Context	Adoption of new product
Reference	[10]

"If the new product is sufficiently attractive, the early adopters will generate favorable word of mouth, stimulating further adoption, increasing the adopter population, and leading to still more word of mouth. This self-reinforcing loop is named the contagion loop to capture the process of social contagion by which the innovation spreads.

But, no real quantity can grow forever. There must be limits to growth. These limits are created by negative feedback (self-correcting). They counteract change. Growing adoption of the innovation causes various negative loops to reduce adoption until use of the innovation comes into balance with its carrying capacity in the social and economic environment."

Density dependent growth

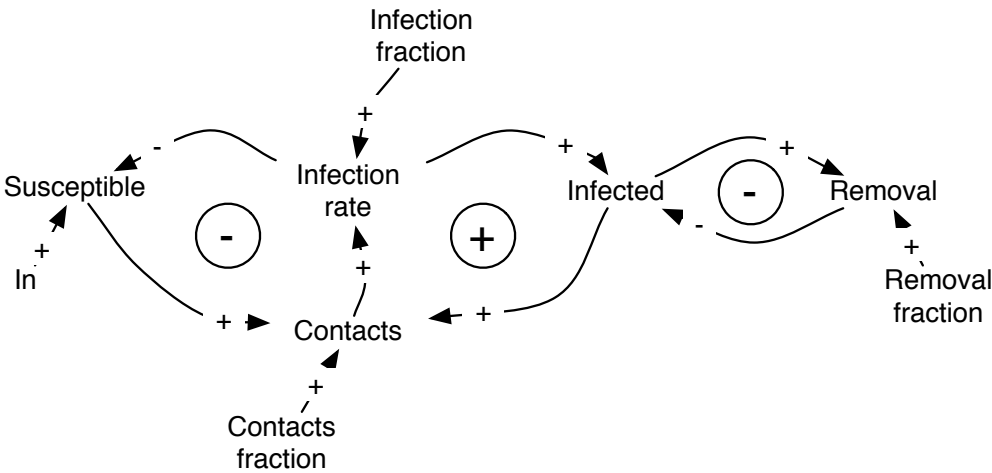


Number of loops	3
Pattern	Limits to Growth - S-shaped growth
Type	Stock-and-flow
Origin	Model from existing model
Validation	Yes - behavior analysis
Context	Non-linear coupling of negative and positive feedback loops
Reference	[9]

"In linear coupling, the exponential growth caused by the positive would continue indefinitely. This is of course not realistic, as all growth must eventually face some limits (area, food, water, resource, air,...). To model such a limiting process, we can introduce crowding as a population/maximum capacity. Thus crowding has a negative effect on birth fraction, that is a balancing loop: it suppresses indefinite growth of population.

Due to this non-linearity, the population growth in a density-dependent model can have two different phase: an exponential growth phase, followed by a goal-seeking growth phase. The first phase is caused by the dominance of the simple positive loop and the second phase is caused by the shift of dominance to the negative density dependence loop."

Growth generated by contacts between two groups



Number of loops	3
Pattern	Limits to Growth - Overshoot-and-decline
Type	Stock-and-flow
Origin	Model from existing model
Validation	Yes - behavior analysis
Context	Dynamics of epidemics
Reference	[9]

"Imagine a contagious disease that can spread by contact between two groups of people (Susceptible and Infected).

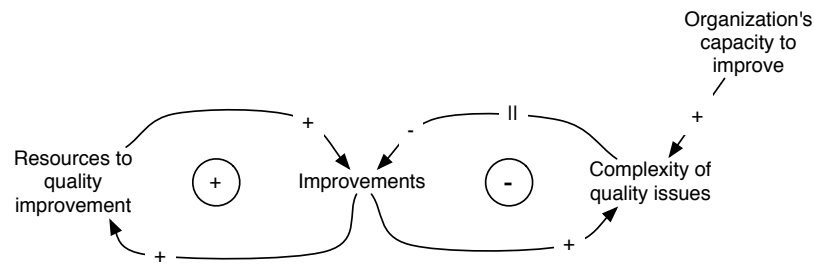
There is an outflow from the Infected population (Removal), representing both deaths and recoveries. It is also assumed that the susceptible group has a constant inflow (such as a net immigration).

Thus, in the case where a epidemic outbreaks, it will eventually settle down to an equilibrium level (of a relatively few number of people).

With certain values, we can observe epidemic oscillations.

And with lower infectivity values and/or higher initial values of the infected stock, there would be no epidemics at all (Infected would decline monotonically over time)."

Picking the low-hanging fruit



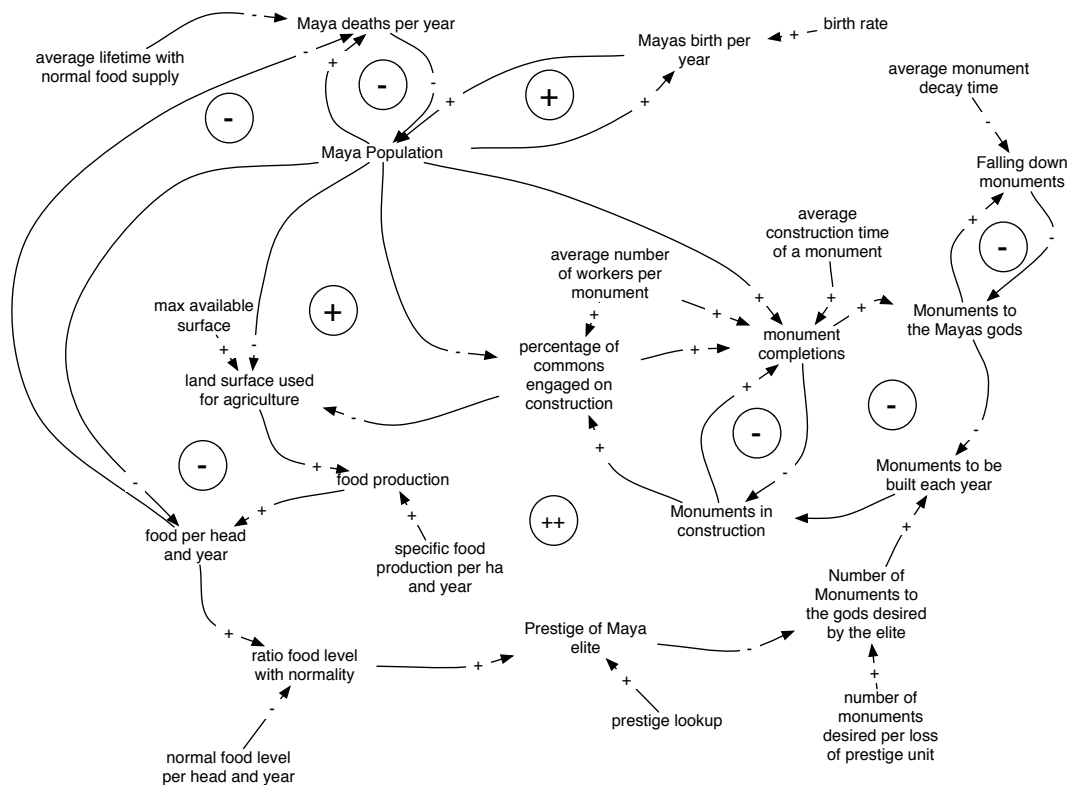
Number of loops	2
Pattern	Limits to Growth
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	Quality improvement campaign
Reference	[19]

"At the beginning of a quality improvement campaign, the first efforts (such as training in the statistical process control tools) lead to significant gains in the quality of products, services, and processes. This lends cachet, support, and impetus to the quality efforts. But as the easy changes (known as "low-hanging fruit" among quality veterans) are completed, the level of improvement plateaus, much to everyone's disappointment. The next wave of improvements are more complex and tougher to manage; they involve co-ordinating several different parts of the organisation. The lack of organisation-wide support, and the attitudes of senior management, now become limits. Unless the company makes more widespread changes at higher levels, its quality gains will be limited."

Other examples from [19]: "The software artists: computer hardware continues getting "faster, cheaper, and better," at an astonishing rate, virtually without limits. However, the production of software for these increasingly complex machines lags behind, often years behind. Without sufficiently sophisticated software, there are limits to the usefulness and popularity of computers. Faced with this limit, hardware producers push to make even faster, better, and cheaper machines."

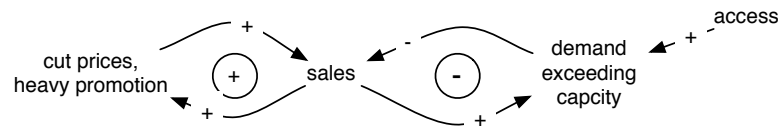
"Reformers creating distance: school administrators and teachers in a community develop an innovative "restructuring" education reform effort. However, as the number of restructured schools goes up, the increased community awareness generates a backlash from parents and other community members who do not want innovation and reform. This is aggravated by the fact that the community perceives it had little to say in the reforms. The educators begin to fight harder to get their point across..."

Collapse of the Maya civilisation



Number of loops	9
Pattern	Limits to Growth - Overshoot-and-decline
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	The collapse of the Maya civilisation
Reference	[24]

"It assumed that the Maya elite tried to recover their lost prestige after recurrent food crises, possibly due to local climate changes, by erecting huge monuments to their gods. The large percentage of commoners and the important space needed in the construction may have aggravated the crisis in a vicious circle causing a further reduction in the food production."

America On Line

Number of loops	2
Pattern	Limits to Growth
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	The success of a fee-per-minute business model
Reference	[25]

"America-On-Line experienced initial success on a fee-per-minute business model. Their competition offered a flat-rate for connecting and accessing the internet. In an effort to both recapture their eroding market share and grow subscribers, AOL began an aggressive marketing campaign, flooding the market with CDs designed to make subscribing and connecting easy and attractive.

The campaign was an enormous success, so much so that the demand completely overwhelmed their technical capacity to deliver service. Not only were new subscribers alienated, so too were existing subscribers who left in significant numbers.

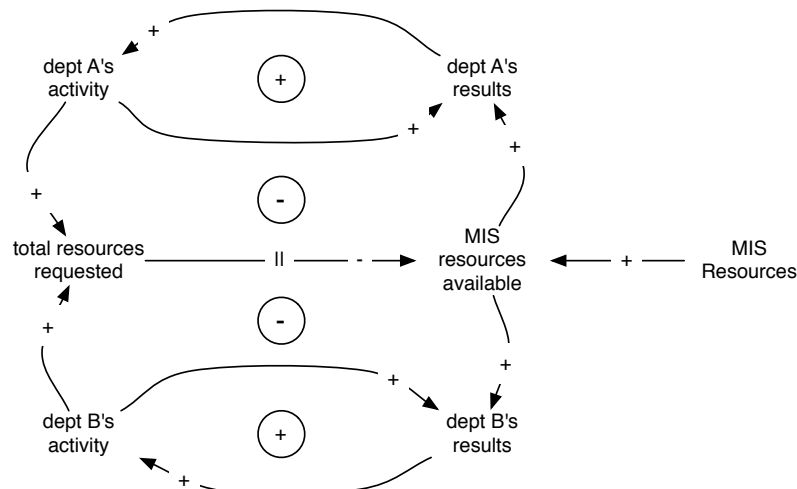
Managers are encouraged to be action oriented and proactive, constantly engaged in the process of pushing on people and situations to make them change or move. Typically, they focus their attention on the sphere of activity in the organization that coincides with their title and job description.

The Limits to Growth archetype (or Limits to Success as it applies) reminds managers to take the time to examine what might be pushing back against their efforts. The counter-force may come, and most likely will come, from either (a) parts of the organization not under the control of the manager or (b) from the external environment. Expansionistic thinking is a key competency for locating Limits to Growth.

By focusing their attention on these limits, managers may find opportunities to either continue the improvement curve they were on, or identify the elements in the system that represent the counter-force and devise new improvement initiatives that would reduce or remove the limits."

Tragedy of the Commons

If It's Free I Want All I Can Get



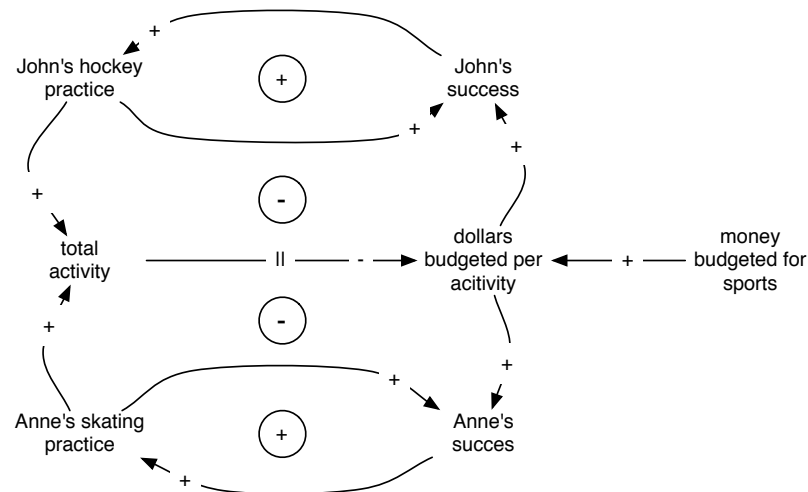
Number of loops	4
Pattern	Tragedy of the Commons
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	Departments within an organisation using MIS resources
Reference	[26]

"As departments employ MIS resources the use contributes to their success, i.e. dept A's results and dept B's results. As each department likes its results they develop plans to use even more MIS resources increasing the total resources requested.

At some point the total resources requested exceeds the MIS resources available. When this happens projects and support become more and more delayed. As the individual groups had planned developments which were contingent on their use of MIS resources they begin to experience a decrease in their results, A's results and B's results, because they have exceeded the capacity of the resources.

If the individual groups had to pay for the services they used they probably wouldn't use so much, and we'd probably be back to a Growth and Underinvestment dilemma. Nothing comes for free!"

When Fair is Unfair



Number of loops	4
Pattern	Tragedy of the Commons
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	Family history
Reference	[26]

"My uncle and aunt deeply believed that one should give their kids all the chances they could to develop their talents. They have two children (let's call them John and Anne, to keep the names short) that fortunately for their parents are really gifted in sports. John is a very good hockey player and Anne is an beautiful figure skater.

Their parents always invested a lot of money for their kids to progress in their respective sports. And as the caliber in which they evolve involves more dollars, both John and Anne had to quit the highly competitive level they reached.

The "Resource Limit" would be the total budgeted money for sports (and - that's not in the archetype, the family was blowing it year after year). The "Gain per Individual" would be the dollars budgeted per activity (hockey vs. figure skating). The "Total Activity" could then be the total money spent on sports by the family.

Now, say A is John and B is Anne... A's gain could be John's practice of hockey, which, as the competitive level increases, gets more and more expensive (travel, more equipment...I guess this would be shown in the arrow going from "A's activity" - competitive level - to "Total Activity"). Another point is that as John perseveres in playing hockey at better levels, he gets better at it, and wants to play more — Reinforcing loop...

The same could be said about Anne's gain is Anne's practice of figure skating, which increases as the competitive level increases. She's getting better and better, wants to practice even

more, and cost goes up (again, an element between "B's activity" and "Total Activity") as now on top of a coach, she needs a choreographer and to rent an ice rink for herself...

This is how the story ended.

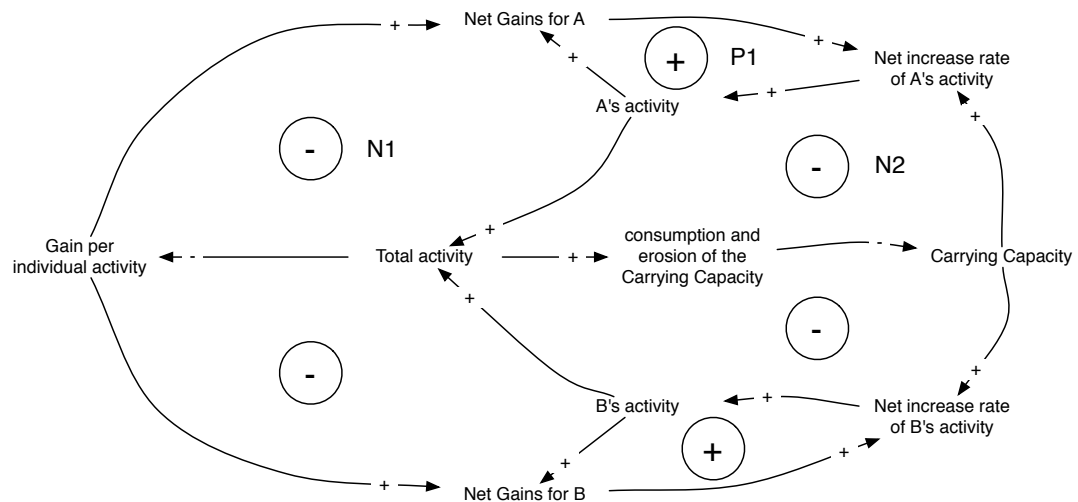
As the parents could not find sponsors for Anne (a friend of hers told me she would be good enough for the National Team) and since no such thing exists for individual hockey players (at least not too easily in Canada), money dried up. They already had taken a second mortgage on the house, my uncle had found himself a second job...

The day of their parent's 20th Anniversary, the kids announced their parents they were both quitting the competitive side of their sport - Anne became coach herself, and John left home to go to university. This was originally a shock to the parents, but after a while, all the family recognized it was the best thing to do...

Anne and John's talent didn't bloom in sync. At one point, when it became apparent that Anne was really gifted for figure skating, John was just an "above the average" hockey player. It was time to take a decision for Anne's future - hiring of a choreographer, big expenses to be foreseen. Their parents took a fair decision to make sure both of their kids had access to similar resources (money) - that's when the second mortgage was taken on the house - and John was sent to a special hockey school. That year, he quickly became the best player in his league...

They might not know, but by avoiding to go in a "Success to the Successful" archetype - all the sports budget could have been used by Anne, they got caught in the Tragedy of the Commons..."

The Tragedy of the Commons

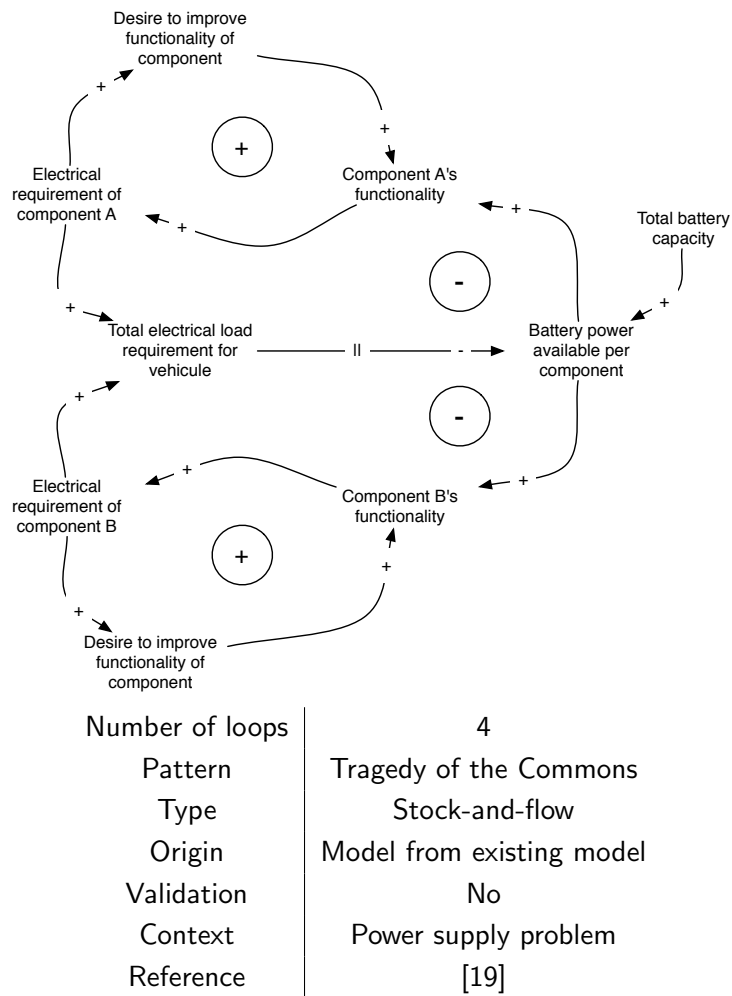


Number of loops	$2 + 2n$ (where $n > 1$ (# of competitors), no tragedy otherwise)
Pattern	Tragedy of the Commons
Type	Stock-and-flow
Origin	Model from existing model
Validation	Influence graph: influence of food resource on population
Context	All for One & None for All
Reference	[24]

"Collapse of a population with the destruction of its renewable food supply. Only two competitors for the common good (A,B) represented in a symmetrical diagram but they are in reality many of them.

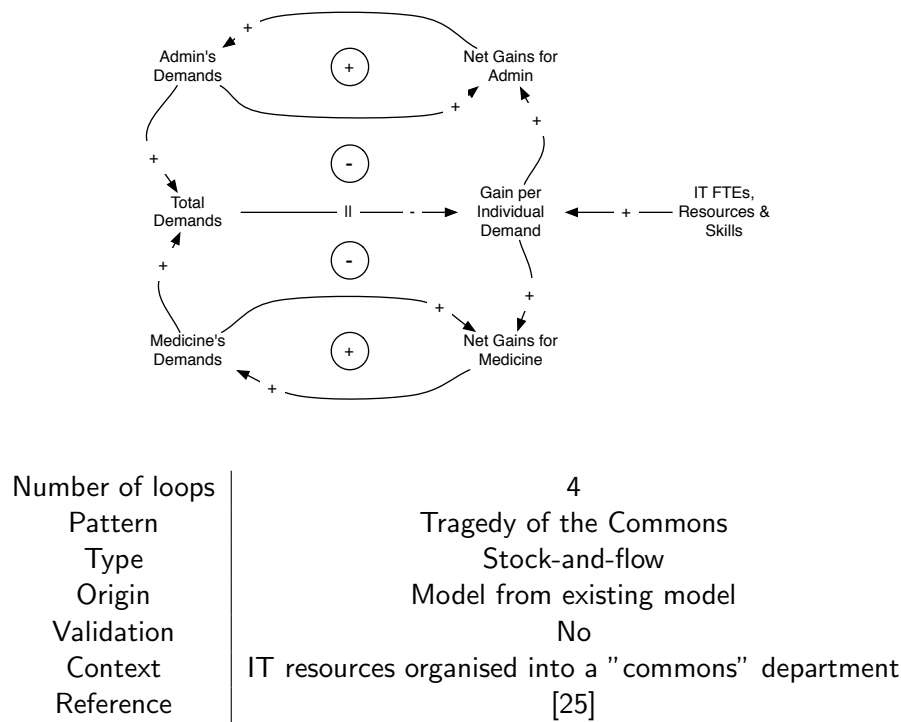
- The unique positive loop P1 generates growth thanks to the resource usage;
- A first negative loop N1 limits the gain of each individual due to the limits in the shared resource. (P1,N1) would bring the system to the well-know stable logistic trajectory;
- But, a second negative loop N2 (erosion loop) brings the progressive disappearance of the resource and its eventual collapse because of its use beyond the carrying capacity."

The Tragedy of the Power Supply



"In Ford's 1994 Lincoln Continental project, the number of electricity-draining components designed for the car overloaded the battery power available. Non of the component designers would back down and reduce their power consumption, because it was in their interest to design electrical components with high functionality. Recognising limits, each design team, within its own group, added even more functionality, to justify being allotted as much battery as possible from the common good. As Nick Zeniuk, business planning manager for the project, tells the story, the team member finally realised that "each person would still look out for his or her own interest unless a) somebody discovered new technology, which wasn't going to happen in the next few months, or b) somebody from the outside came in and dictated. What did we do? I came from the outside and dictated." "Dictating from the outside" worked here only because of the effort Ford's team make to discover the "Tragedy of the Commons" dynamic. Everyone had seen themselves that the system encouraged them to pursue their own individual rewards, not the optimisation of the whole."

IT Project Requests



"IT resources are typically organized into a commons department, with each part of the organization seeking their support on an as-needed basis. Since separate parts of the organization typically do not keep track of the IT problems in other parts of the organization, it is fairly common for each part of the organization to see the IT department as its own. When the IT department is crushed under the weight of all the demands placed upon it, its performance for every department begins to erode or fail.

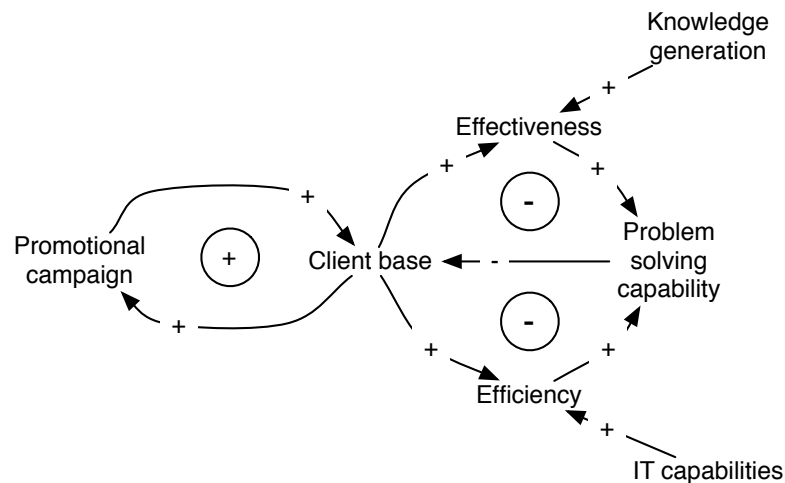
In many respects the Tragedy of the Commons is a classic example of reductionistic thinking. By remaining unaware of the effect of the parts on the whole, people continue to think and behave as though there are no connections within the organization that affect their ability to meet goals and objectives. Focused on their own part, behaving as though it depended on no other, demands on the commons are issued with only the present in mind.

Sustainability is increasingly put forward as a guiding principle for the planet we inhabit. Sustainability has applications within organizations, with respect to their structure and practices, with an eye on the long-term future. Structures that create commons and policies and practices that govern them (leading to depletion or replenishment) are critical success factors.

Ultimately, firms may conclude that structures that include a commons are ineffective means of distributing and allocating resources. Alternately, they may gain insight into how commons have to be governed, and recognize that structures and policies, other than the commons itself, all interact and have a pronounced effect upon the utility the commons bring to organizations."

The Attractiveness Principle

Consulting Firm



Number of loops	3
Pattern	Growth & Under Investment
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	Consulting firm that faces a decision problem
Reference	[25]

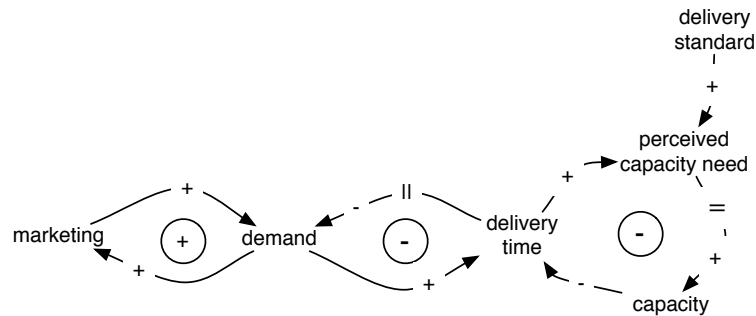
"A consulting firm is faced with the decision/dilemma on how to improve its overall performance for clients, choosing between shoring up its IT capabilities or growing its knowledge base, both of which are under attack from existing clients, and are acting as deterrents to acquiring new clients.

The Attractiveness Principle pits managers against growing complexity and the interactions between parts that are increasingly difficult to anticipate. Although implied with many of the archetypes, it makes a strong case for dynamic modeling to reveal the synergies that may emerge from the firms response to growth engines as complexity increases.

At its core is expansionistic thinking; the requirement that managers seek to solve systems of problems in the largest system to which they have access. The archetype reinforces the distinction between understanding and knowledge. Knowledge, the know-how managers rely on to make decisions, precedes from the contained parts of the whole to the containing whole, while understanding precedes from the containing whole to its parts."

Growth & Under Investment

Creating the Future, Undesirably



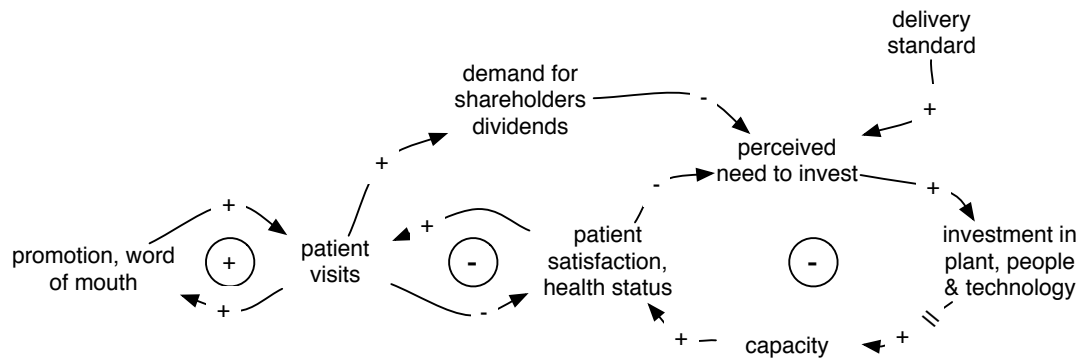
Number of loops	3
Pattern	Growth & Under Investment
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	This pattern is simply a Limits to Growth pattern with some implications in terms of the minimising of the limits
Reference	[26]

"In a business context marketing can generate demand. When the marketing effort succeeds in this fashion the normal inclination is to do more marketing. If some works more should work better.

The difficulty comes into play when the demand runs into the limiting aspect of capacity. Greater demand will result in a requirement for more product to be shipped, and when the demand approaches and finally exceeds capacity it will result in longer product delivery time. Longer delivery time is fine as long as it's less than what's acceptable to the market. If delivery time exceeds acceptable delivery time (not shown) then it will begin to impact demand. As delivery time interacts with the delivery standard, and there should be such a thing, it will add to the perceived capacity required. Since capacity is something that usually requires substantial investment and takes time to develop there is a delay associated with its increase. Because of this delay it is quite possible that by the time capacity has been increased the increases in delivery time will have negated the increase in demand created by the marketing effort.

Thus, as the organization ponders action, which quite often it does, for far too long, the demand will dissipate and they will say, "See, we didn't really need the added capacity after all." And I guess they didn't need the extra sales that would have resulted from the capacity increase either."

The Shareholders Problem



Number of loops	3
Pattern	Growth & Under Investment
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	The Shareholders problem
Reference	[25]

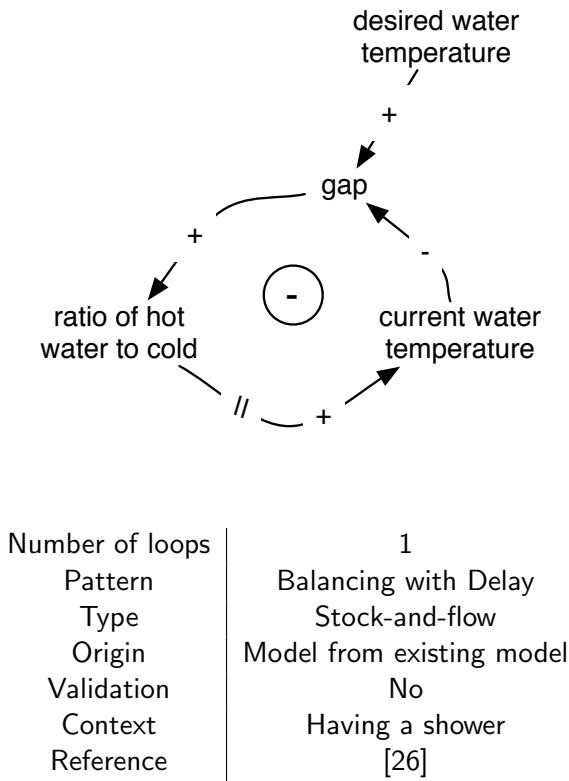
"In private practice, shareholders historically treat the business firm as a wealth generator for their families. There is typically a tension between the desire to remove profits from the practice and the need to invest in infrastructure, especially technology. Over time, performance slips so far, that patients find it increasingly difficult to receive care at the practice, mostly for operational reasons (though clinical equipment and technology could likewise be affected.)

Growth and Underinvestment is the archetype that brings special attention to planning for limits. In this case, it is the capabilities and core competencies that give firms their competitive advantage. This is part and parcel of strategic planning as well as internal policy formation.

It also draws attention to the insidious nature of the failure to meet customer demands over long periods of time - the constant (albeit hard to notice in any one period) decline in the firms opinion of itself and in its commitment to, and ability to perform at, customer demands and expectations."

Balancing with Delay

Adjusting the Shower

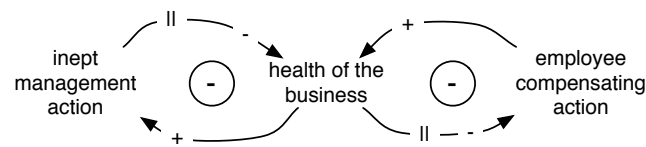


"When the current water temperature is less than the desired water temperature it influences me to add to the ratio of the hot water to cold. I can do this by increasing the hot water or decreasing the cold water. Because changing the faucet position doesn't result in an immediate change in the current water temperature there is a tendency to believe that the ratio isn't correct so I turn the faucets more in the same direction. When the delay finally catches up I find that the current water temperature is much hotter than the desired water temperature so I adjust the faucets in the opposite direction. This action still adds to the ratio of hot water to cold, just less than before. Depending on how impatient I am, and how much I overreact to the situation, it could take some time before I get the water temperature to where I really want it.

The real problem in this structure is based on my not understanding the structure coupled with my impatience for what I want. The strategy for dealing with this structure is, first understand the structure, and then exercise patience. This is an example in support of the statement, "Patience is a virtue."

Escalation

The Niagara Effect



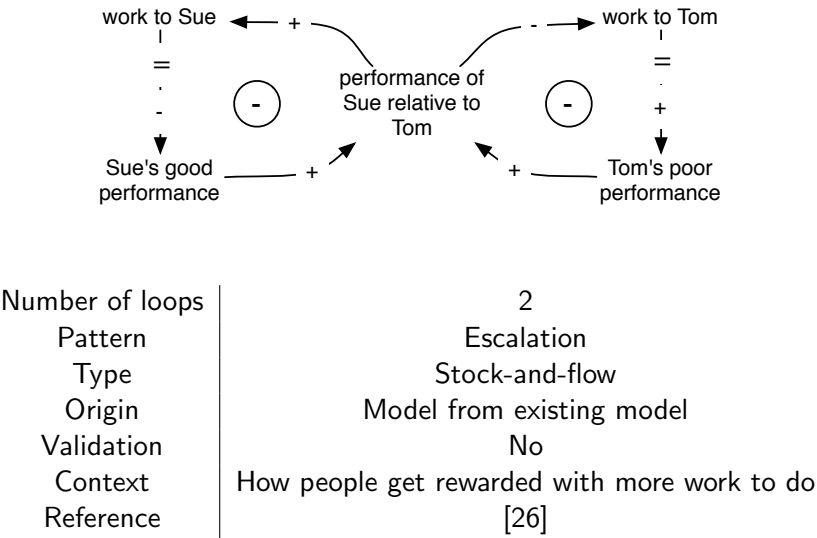
Number of loops	2
Pattern	Escalation
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	Business environment in which the management of the organisation is substantially less capable than it needs to be and the employees are very astute
Reference	[26]

"Inept management action over time subtracts from the health of the business. The health of the business the subtracts from employee compensating action to maintain the health of the business. If the health of the business is good then there is little need for employee compensating action. You might say it's business as usual. Yet as the health of the business declines it subtracts less from the employee compensating action. Employee compensating action increases.

Employee compensating action adds to the health of the business essentially compensating for management ineptness. The health of the business then adds to inept management action. That is, since the business is healthy, the ineptness of management is compensated for by employee compensating action and management is deceived into thinking things are fine. This situation simply promotes management's continued ineptness.

If something within the system is not changed sooner or later inept management action will subtract from the health of the business to a point where employee compensating action will no longer be able to compensate for it. When this happens employees essentially reach their limit and give up and the system crashes, meaning the health of the business will decline very rapidly. When this happens management is most apt to be quite puzzled as to how things got to be so bad so quickly. This is often referred to as the Niagara Syndrome where everything seems relatively calm, and when the falls are finally perceived it's already too late to recover."

Punished for Success



"How many times have you seen situations where the most productive people are rewarded with more work, or when a segment of a process doesn't deliver the value it should you go around it. Although these actions seem sensible in the name of immediacy they actually lead to the longer term decline of the whole structure. There are probably numerous appropriate alternative labels for this example. A few that come to mind are: Failure by Success, Overworking the Overworked, Punishing the Gifted, Rewarding Mediocrity, etc.

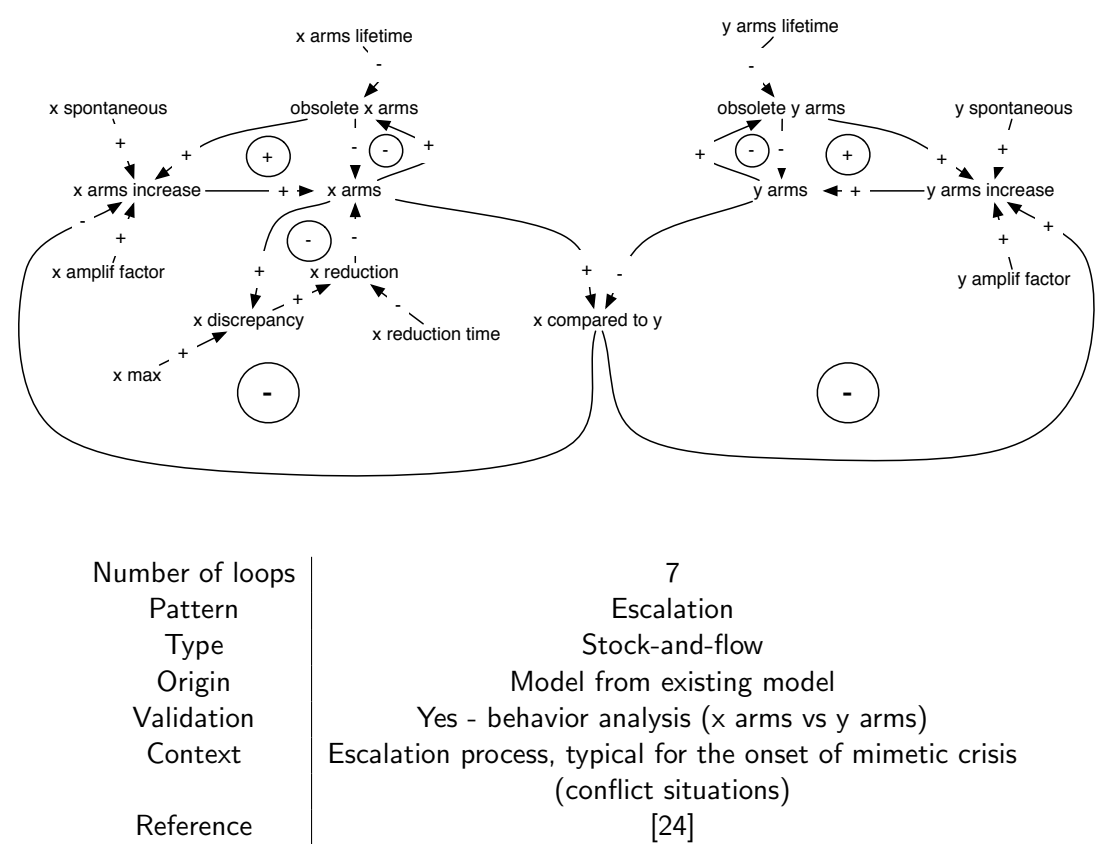
Consider the following example in which Sue is a superior performer and Tom is quite a mediocre performer. This is really an example of a viscous escalation structure in which the overall results decline over time.

Because of the performance of Sue relative to Tom, Tom is rewarded with less work to Tom. Since Tom is a poor performer this work simply adds to his poor performance which in turn adds to the performance of Sue relative to Tom.

Since the performance of Sue relative to Tom is in Sue's favor the next critical, urgent, or essential assignment of work will add to the work to Sue. This will happen over and over until Sue is essentially overloaded. At this point the work to Sue will subtract from Sue's good performance. As Sue's good performance declines it will add less to the performance of Sue relative to Tom, yet she will still out perform Tom.

The continued positive performance of Sue relative to Tom will continue to subtract from the work to Tom and the structure repeats."

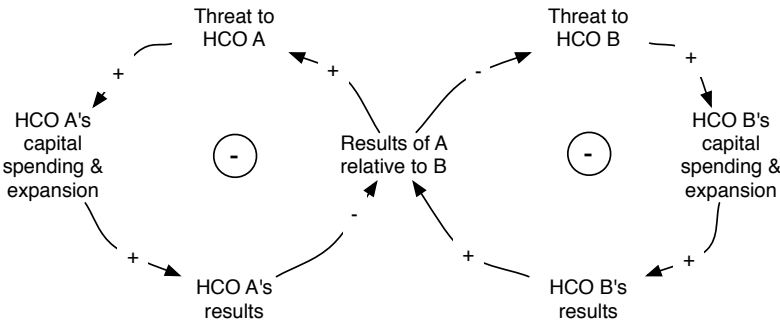
The 'Arms race' archetype



"The eight-shaped loop traced by the combination of the two central negative loop increases tension in an exponential way.

De-escalation can only be obtained if one party accepts stepping back, diminishing the source of threat (To make the first step is more than often a courageous decision, because of the risk of loosing face). In this case, x stepped back."

HCO Expansion



Number of loops	2
Pattern	Escalation
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	Market competitions
Reference	[25]

"In the health care industry, especially in a geographically defined market, it is not uncommon for competitors to engage in a campaign of erecting buildings as a tactic for securing market share. Each facility is seen as a threat by the competitor, who after some delay, will respond in kind. This can continue for some time until the cost of doing so becomes prohibitive and the escalation stops. This may result in one competitors eventual market dominance (if it had the resources to support the construction boom) or in one competitors collapse due to overextending itself financially.

This archetype is difficult to apply - it appears to strike at the heart of the core tenets of free enterprise. Thinking and/or behaving any other way could have ramifications for the manager and the firm - engaging in anti-trust practices for example.

It may be that this archetype may find its value in the public policy arena, or in industry and/or community based assessments of the needs, expectations and requirements of customers and other stakeholder constituencies."

Indecision

Supply and demand



Number of loops	2
Pattern	Indecision
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	How supply and demand operate through price
Reference	[26]

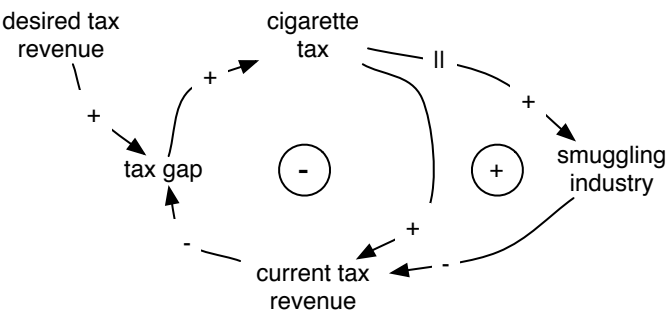
"Demand and supply interact to establish the price. If demand rises, after some delay, it will add more to price. This increase in price will have a short term effect on demand, subtracting from it, and after some time it will add to the supply. If the product is worth more then there is a tendency to produce more. As supply increases it subtracts more from price, just at the time demand is made it through the delay to add less to the price. As such supply is more than appropriate for the demand and price decreases.

The decrease in price subtracts less from demand, increasing it, and after some time will produce a reduction in supply. So, by the time the increased demand reaches price supply has been reduced and they're out of sync one more time. This induces price to go back up.

Because of the delays in this structure supply and demand never seem to be what is appropriate for each other and price oscillates up and down, endlessly."

Fixes that Backfire

Tax Revenue Pressure



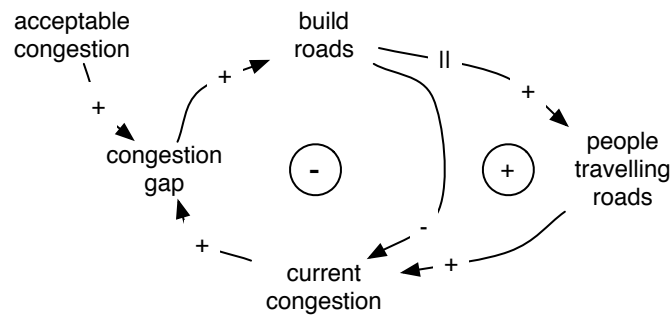
Number of loops	2
Pattern	Fixes that Backfire
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	Tax revenue problem
Reference	[26]

"The Canadian Government was facing a tax gap due to the difference between their desired tax revenue and their current tax revenue. Their solution to this situation was to increase cigarette tax with the intent of increasing current tax revenue. As it turned out current tax revenue actually declined.

It seems the increase in cigarette tax was enough so Canadian cigarettes were more expensive than American cigarettes. This fostered the development of a smuggling industry between the US and Canada. This reduced the sale of taxed cigarettes thus reducing the current tax revenue. When asked why they didn't police the smugglers the border police commented that they were carrying hand guns while the smugglers were carrying machine guns.

When the situation was understood and the cigarette tax reduced so Canadian cigarettes cost less than American cigarettes the smuggling industry died of its own accord within 5 days."

Road Congestion



Number of loops	2
Pattern	Fixes that backfire
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	Road Congestion
Reference	[26]

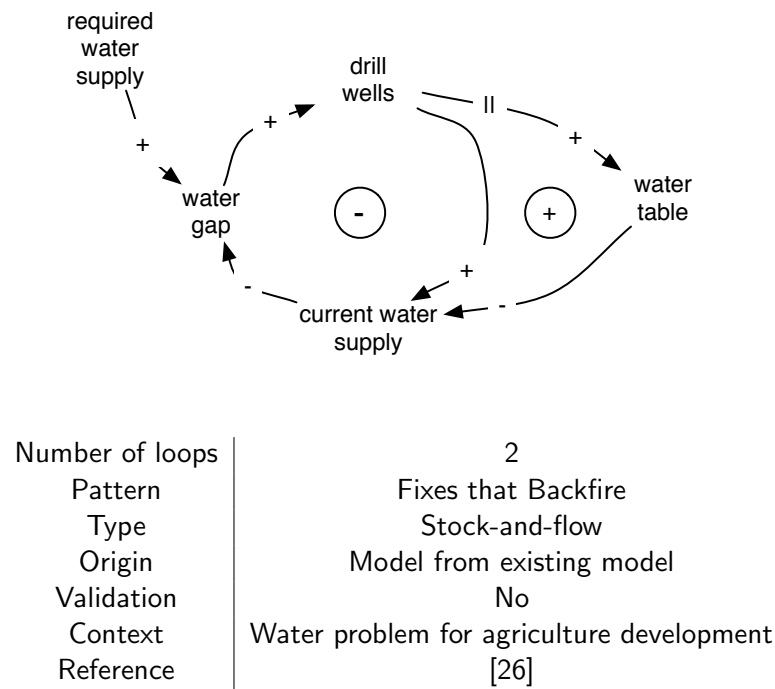
"Here's but one more fixes that backfire example of how the initially most sensible answer leads to exactly what isn't desired, and costs money besides.

As the number of vehicles on the road increases the amount of congestion also increases.

The difference between the current congestion and acceptable congestion creates a congestion gap. As people complain more and more about the congestion the appropriate agency finally decides they need to do something to get people off their back. The short term (not so short) response to this situation is to build roads which serves to reduce the current congestion. The unintended consequence of this is that more roads adds to people traveling roads which only serves to increase the current congestion. This also serves to increase the congestion gap necessitating just another cycle of build roads.

A better solution would be to provide more effective mass transit systems which would get people out of their cars thus reducing the current congestion and alleviating the need to build roads. Which implies this example could be elaborated into a shifting the burden structure as the burden for addressing the congestion situation is shifted from the fundamental long term solution using mass transit to road building."

Quest for Water

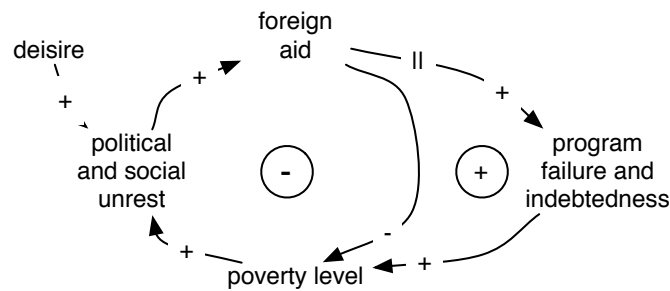


"The following is a fixes that fail structure associated with an initial problem perceived as farmers need more water for agriculture development.

As a short term resolution to the water gap it is decided to drill wells. In the short term the additional wells increase the current water supply, thus reducing the water gap. The longer term affect is that as water is used it draws down the water table eventually decreasing the available water which results in the need to drill more deeper wells.

Specific examples of this are the state of the Ogallala Aquifer in the Midwest and current conditions in China."

Perpetuating Poverty



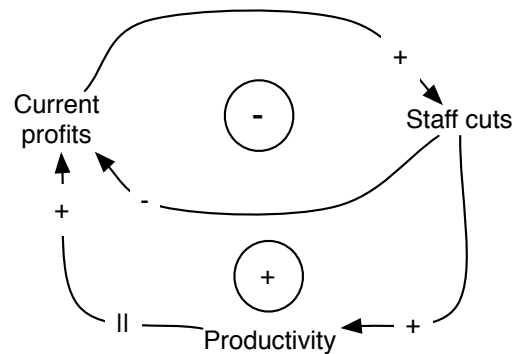
Number of loops	2
Pattern	Fixes that Backfire
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	Poverty in rural area
Reference	[26]

"Problem: poverty in a rural area in the Less-developed-country (LDC) which is causing political and social unrest . Another example of fixes that fail.

Quick Fix: the government (often with USAID or the World Bank) designs a development program for the poor farmers of the region which often times involves making loans to farmers to implement the "development" program – to buy equipment, fertilizer, seeds, whatever.

Unintended consequence: Because the program was designed from the outside and did not involve local input, it is nearly always inappropriate to the ecological and social conditions of the region. As a result, the project fails and the people are even more in debt to the banks and there has also been a significant degradation of the natural landscape because of the introduction of inappropriate technology. The people are unhappy and there is even more social and political unrest which means we need another FIX."

Downsizing to improve profits

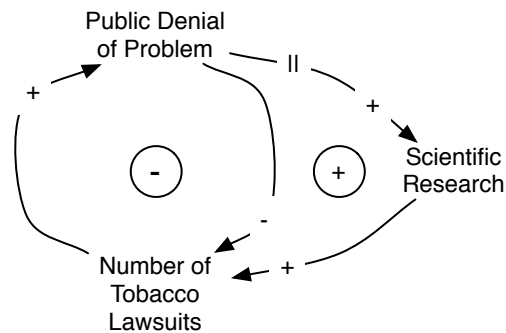


Number of loops	2
Pattern	Fixes that Backfire
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	A company reduces staff to reduce costs and raise profitability
Reference	[19]

"The most leverage seems to come from encouraging older workers, who generally have higher wages, to take early retirement. To everyone's delight, profitability immediately improves. However, the staffing cuts also eliminate some of the older, more experienced staff. Morale problems from layoffs drain enthusiasm. Production costs increase through error and overwork. These factors contribute to lowered productivity (the unintended consequence) and drain away all the added profitability from the "layoff fix", and then some. Management decides, with a heavy heart, that it has no choice but to make more staffing cuts ..."

Other example from [19]: "Expediting customer orders: a large semiconductor manufacturer experiences production problems and runs behind schedule on some shipments. The company knows its customers (computer makers) will have to shut down production lines until the chips are delivered. The Moon Computer Company calls demanding that its chips be delivered immediately, so the semiconductor manufacturer assigns an expeditor to track down Moon's order and push it through the line (the fix). Of course, it's not simply a matter of finding the right chip and escorting it to the loading docks; expediting Moon's order means wading through the entire factory, and repeatedly disrupting the production line, at great extra cost and effort. Unfortunately, no sooner has Moon's order left the warehouse when the LaSt Computer Company calls, demanding *its* shipments. Another department, meanwhile, is expediting for Conneq Computers. As a result, the production line is continually disrupted - leading to more missed delivery dates, and more customer calls."

Tobacco Industry



Number of loops	2
Pattern	Fixes that Backfire
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	III health effects from smocking
Reference	[25]

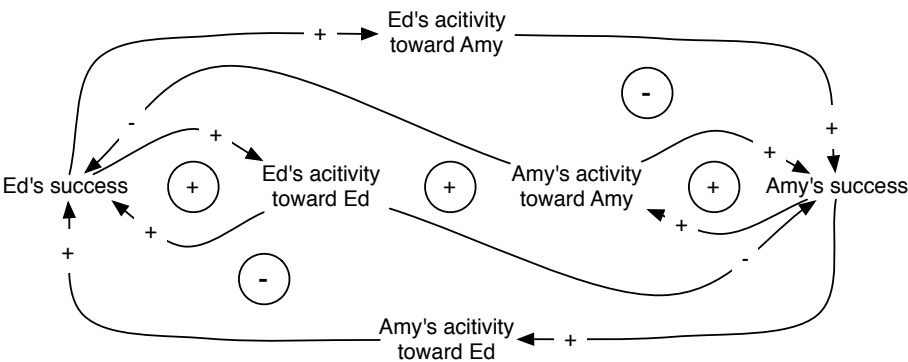
"For years the tobacco industry steadfastly denied that there were any ill health effects from smoking, pouring vast amounts of money into advertising and a pattern of denials. The tactic served the industry well. However, each time it denied that smoking caused health problems, it stiffened the resolve of scientists, and research into the effects of smoking on health steadily grew. Ultimately, the amount of evidence grew so large that no amount of PR or advertising could overcome the industrys claims.

The key to appreciating the Fixes that Fail archetype is the delay in the balancing loop. The time that elapses between the fix and the worsening problem symptoms frequently makes the connection between the fix and the deteriorating problem symptoms hard to identify. Managers tend to attribute the worsening problem symptom to something other than the prior decision(s) they made in their efforts to fix the problem symptom(s).

Despite its apparent simplicity, Fixes that Fail can be devilishly hard to unravel. It requires a deep commitment to setting aside mental models that may strongly influence managers not to see, or even consider, that there may be a connection between the problem symptoms that are visible and the fix(es) they are applying in an effort to alleviate the problem symptoms."

Accidental Adversaries

Self-Defeating Action



Number of loops	6
Pattern	Accidental Adversaries
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	Collaborative work
Reference	[26]

"This is a rather contrived Accidental Adversaries example consisting of two individuals working in the same group. Amy is very well organized and methodical, while Ed is very insightful and creative. Together they could make a great team, yet over time they defeat each other's, and eventually their own, success.

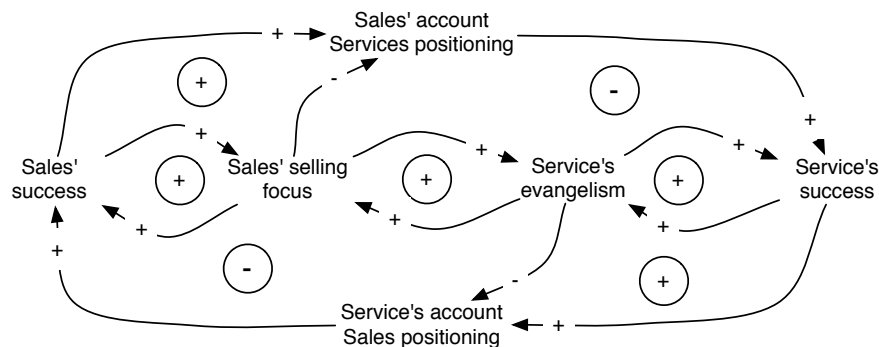
The insightful and creative person, Ed, continues to come up with new ideas for improving things, yet has difficulty figuring out how to present his ideas and how to plan their implementation. Amy's activity supporting Ed is then in terms of presentation development and planning. This leads to Ed's success for the presentations go very well and the implementations are perceive as very well planned. Ed's activity supporting Amy is then in terms of giving credit to Amy for the assistance in the presentation development and planning. This contributes to Amy's success, encouraging Amy to continue to act supportingly toward Ed. This is very definitely a virtuous reinforcing cycle.

All this works well until Ed becomes wrapped up in his own glory and begins to take credit not only for the concepts and ideas but for the presentation and planning of them. This is Ed's activity toward Ed which detracts from Amy's success. This also acts to decrease Amy's activity supporting Ed. At the same time this is happening Amy becomes dissatisfied with Ed getting credit for all the creative thought and Amy's activity toward Amy tends to be in terms of beginning to take some measure of credit for the development of the ideas. This action in fact detracts from Ed's success thereby diminishing Ed's Activity Supporting Amy as Ed is likely to start looking for someone else to assist him with presentation and

planning, or even worse Ed may begin to believe his own lies and consider he is quite capable of doing his own presentation development and planning.

As it turns out the two balancing loops that are created tend to negate the reinforcing nature of the initial cooperative activities. We are most certainly our own worst enemies."

Sales & Service: a Marriage Made in Hell



Number of loops	7
Pattern	Accidental Adversaries
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	Collaborative work
Reference	[26]

"This Accidental Adversaries example consists of two groups, Sales and Service, which working in the collaborative fashion have tremendous potential. Yet, operating from their own myopic unenlightened self-perspective, over time they defeat each other's, and eventually their own, success.

Sales's success leads to appropriate Sales's account Services positioning. What this means is that the appropriate expectations regarding services are established within the account when it's sold, which then supports Service's success. Service's success then leads to appropriate Service's account Sales positioning. This means preparing the account for further sales. This action leads to further Sales's success. This sequence represents a virtuous reinforcing loop promoting a continued increase in the success of both Sale and Service.

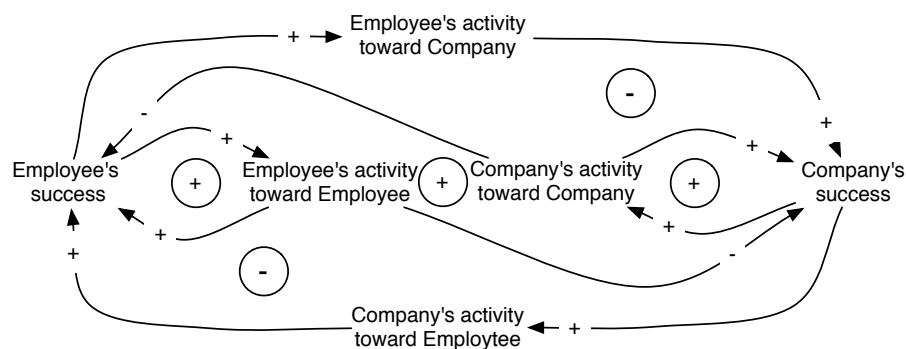
Service's evangelism essentially ensure successful customer implementation and Service's success, which further promotes Service's evangelism. Sale's selling focus, promoted by the fact that they're only compensated for sales and not service, essentially ensures Sale's success, which in turn promotes Sales's selling focus.

Now when we look a little deeper we find that Sales's selling focus detracts from Sale's account Services positioning by discounting the level of effort and time required for implementation. Sales's selling focus also serves to promote Service's evangelism to compensate for Sales's selling focus. On the other side of the coin, since there are no one sided coins, Service's evangelism detracts from Service's account Sales positioning by extending the implementation cycle. Let's face it, it takes a long time to create converts. Service's evangelism also promotes Sales's selling focus to compensate for Service's evangelism.

As it turns out, the interactions described above also create two additional insidious balancing loops and two viscous reinforcing loops. These loops simply serve to further degrade the overall result of the structure.

Reality depends on who believes it, while truth doesn't care who believes it."

Employee/Company Relationship: The Failure of Short Sighted Pursuit



Number of loops	6
Pattern	Accidental Adversaries
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	Collaborative work
Reference	[26]

"This Accidental Adversaries example consists of two groups, Employees and the Company, which working in the collaborative fashion have tremendous potential. Yet, operating from their own myopic unenlightened self-perspective, over time they defeat each other's, and eventually their own, success.

The external loop of this structure forms a virtuous reinforcing structure where the Employee's activity toward the Company adds to the Company's success. The Company's success then adds to the Company's activity toward the Employee which in turn adds to the Employee's success. The Employee's success then adds to the Employee's activity toward the Company. This structure actually supports the premise that the customer comes second

(Rosenbluth, 2002) as employees are unlikely to treat customers any better than they're treated by the company.

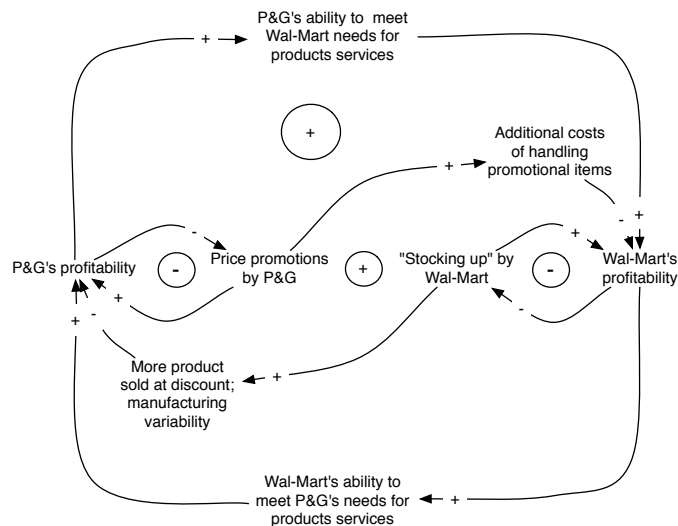
Because of myopic short-sighted unenlightened self-interest, flawed mental models, and lack of trust, both employees and the company pursue actions which end up derailing this virtuous structure.

Employees engage in gamesmanship, more often political than otherwise, to promote their own success, and to the extent they perceive that it succeeds, it promotes them to engage in even more of the same activity. This activity actually detracts from the amount of activity employed toward the company's success. Even worse is the fact that the environment all the gamesmanship creates actually detracts directly from the company's success.

The company on the other hand tends to focus on growth, rather than development, which is perceived to improve the company's success. To the extent that this appears to result in company success it encourages even more focus on growth. This focus on growth detracts from the efforts toward employee development and even has a direct negative impact on employee success. Organizations though their own actions become much like cemeteries in that they grow each year though don't develop. This eventually leads to their demise.

As it turns out, in the long run, neither group can benefit from unilaterally attempting to enhance their own success as it degrades the entire structure. If the situation were a zero-sum game the current situation would represent a Nash Equilibrium. Though, since it's not a zero sum game, because of the virtuous nature of the outside reinforcing cycle, the individual positioning seems to work. And, as it generally takes quite a while for the activity to negate the reinforcing portion of the structure, the dynamic complexity of the environment escapes the participants. When the structure fails everyone is in a quandary as to what caused the problem, when in fact they all did."

Retailing companies



Number of loops	4
Pattern	Accidental Adversaries
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	The largest consumer products and retailing companies in the world
Reference	[19]

"One classic case, where this structure was first recognised and articulated, concerned the largest consumer products and retailing companies in the world. Procter & Gamble and Wal-Mart both had the same goal - improving the effectiveness and profitability of their production/distribution system - but they each felt the other was acting (perhaps deliberately) in self-serving ways that damaged the industry. These perceptions were not unique to Procter & Gamble and Wal-Mart; they were rampant in the industry

As two of the most capable corporations in the world, Procter & Gamble and Wal-Mart had long been aware of the advantages of co-operating closely with (respectively) their distributors and their suppliers. (This co-operation, which gently reinforced itself, forms the outer reinforcing loop in the diagram). In the mid-1980s, however, both companies realised that their relationships had deteriorated, partly as a result of a fifteen-year-long pattern of behaviour. Manufacturers (like Procter) had learned through the 1970s and 1980s to heavily discount their goods and use lots of price promotions in marketing, to boost market share and value, and thereby improve profits. (This is shown in Procter & Gamble's balancing loop, the small circle at upper left).

But price promotions created extra costs and difficulties for distributors (like Wal-Mart), which coped by "stocking up", also known as "forward-buying" - buying large quantities of the product during the discount period, selling it at regular price when the promotion

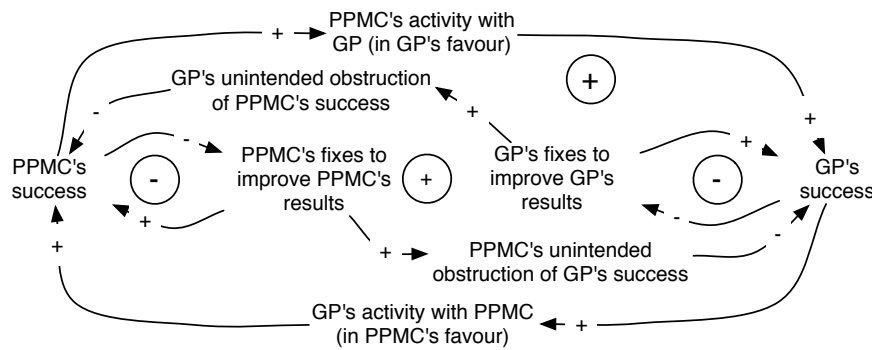
ended, and using that extra income to improve their margins. (This strategy is shown in Wal-Mart's balancing loop at lower right). This of course, deeply undermined the manufacturer's profitability, because the retailer discounted many times the manufacturer's intended amount of product. Worse still, it created swings in manufacturing volume, adding to cost, because distributors (being already stocked up) wouldn't order more product for months. To improve their results, the manufacturers pushed even more heavily on promotions, blaming the distributors for their troubles; and the distributors, blaming manufacturers, stocked up even more.

Eventually, consumer products companies found themselves putting effort into promotions at the expense of new product development, while distributors concentrated on buying and storing promoted products instead of basic operations. Much of the short-term profits from promotions were drained away in long-term costs. A reinforcing loop had formed in the middle, causing a death spiral of mutually detrimental actions.

Each of the partners recognises that they could mutually support each other's success - as shown by the large outer loop. However, as they take independent action to improve results, they respond more attentively to their local needs than their partner's. Each partner's "solution" turns out to be unintentionally obstructive to their counterpart's success. Often widely separated, the two partners do not communicate well. They tend to be unconscious of their effects on each other. One partner feels it is merely pulling an opportunity closer, but the other partner feels as if it is being flung through the air recklessly, flailing around at the end of the first partner's scope.

Later, as the unintended obstructions are felt more strongly, each remains confident that the solution is to convince the other partner that its strategy is the correct way to improve results. In general, at this stage, each partner has almost forgotten its original purpose in collaboration. It is much more aware of the things its purported partner - that traitor! - has done to block it. This makes the partner even more unlikely to talk, and it becomes even more unlikely that either side will ever learn the effect it is having on the other."

PPMC and Group Practice



Number of loops	4
Pattern	Accidental Adversaries
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	Collaboration
Reference	[25]

"In the early 1990's the Physician Practice Management Corporation industry emerged. PPMCs purchased the hard assets of a practice in return for a percent of revenue for operational services rendered. Initially the relationships fared well. Eventually however, when performance and growth lagged, physicians became uneasy with the relationships and began to interpret every move by the PPMC as potentially (or actually) injurious to their interests. The result was the downward spiral of both parties interests.

The lesson of Accidental Adversaries lies in the power of mental models to supply all too ready explanations of situations. Unless judgement is suspended these mental models can drive one, both or all parties to conclusions that bear remote resemblance to the underlying reason the breach in the relationship occurred in the first place, if indeed any breach actually took place.

There is also a lesson on Shared Vision in this archetype. The degree to which the parties hold a vision in common and have articulated their deep needs and expectations is a significant contributor to tempering reactions of the parties when breaches are perceived.

Breaches in the agreement(s) may happen; the probability of deteriorating into Accidental Adversaries is decidedly lower when the parties believe there are overarching values and objectives that unite them in Shared Vision.

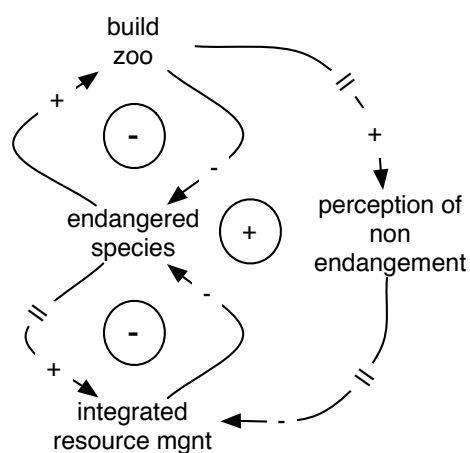
Shared Vision will contribute insight to the extent that partners actually engage in helping fix problems (or problem symptoms) in their partners organization because of their understanding of the long-term impact their efforts will have on their own firms success. This suggests that Shared Vision is connected to a sense of mission higher than money, that a

sense of purpose to customers and an underlying, shared sense of organizational values and culture must be the bedrock of the partnership in the first place.

The archetype also draws attention to Team Learning. If the partners in the venture adopt a principle of continuous joint improvement and learning, the probability that breaches to the partnership will happen in the first place is diminished, as well as a higher probability that if and when misunderstandings, unrealistic expectations or performance problems do occur, the parties will have mechanisms in place to meet each other half way and work them out."

Shifting the Burden

Newton's Zoo



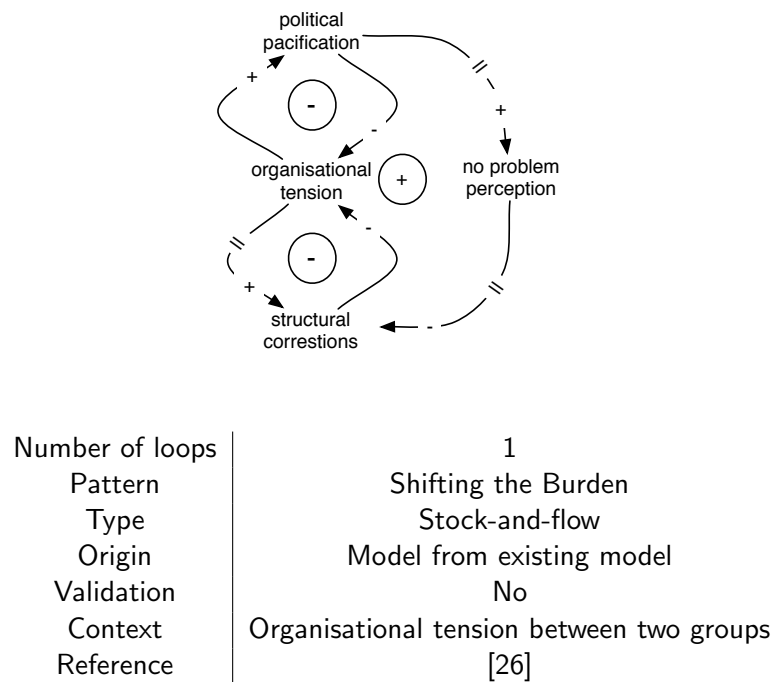
Number of loops	3
Pattern	Shifting the Burden
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	Building zoos for endangered species
Reference	[26]

"It seems that on Earthday 1996 there was a newspaper article in which Newt Gingrich was claiming he was in favor of the environment, after all he had helped raise funds for a zoo!

If the problem is perceived as endangered species then building zoos is an apparent quick fix because it will provide an environment which protects the endangered species. This solution is much more timely than the longer term solution of doing integrated resource management wherein the habitat of the species is protected, thus enabling the species to survive.

The side effect of this approach is that people will visit the zoos, see the animals, and develop a perception of nonendangerment, thus reducing the support for doing integrated resource management."

Perpetuation by Self-Deception



”The following example of a Shifting the Burden structure is based on the existence of an organizational tension between two individuals, groups, or organizations. By organizational tension it is simply meant that the two entities are constantly at odds with each other over lots of things, all related to their day to day operation.

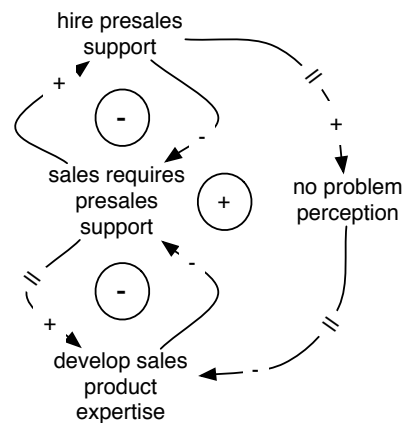
The problem is perceived in terms of the visible organizational tension. There are two approaches to resolving this organizational tension.

The short term expedient approach is for one group to work extra hard to smooth the tensions that exist with the other group. This reduces the visible symptoms of the organizational tension.

The longer term more difficult approach is to perform the structural corrections that are the foundation of the organizational tension in the first place. This approach is probably more difficult because the structure creating the tension is probably created by some other part of the organization and neither group involved is in a position to effect the appropriate structural corrections. This situation actually promotes the adoption of the first approach.

The side effect of the political pacification is that the diminished visibility of the organizational tension promotes the no problem perception on the part of the organization that is in a position to effect the appropriate structural corrections that would dissolve the organizational tension. As it turns out the political pacification action only serves to ensure the appropriate structural corrections will never happen.”

Rewarded with Limitations

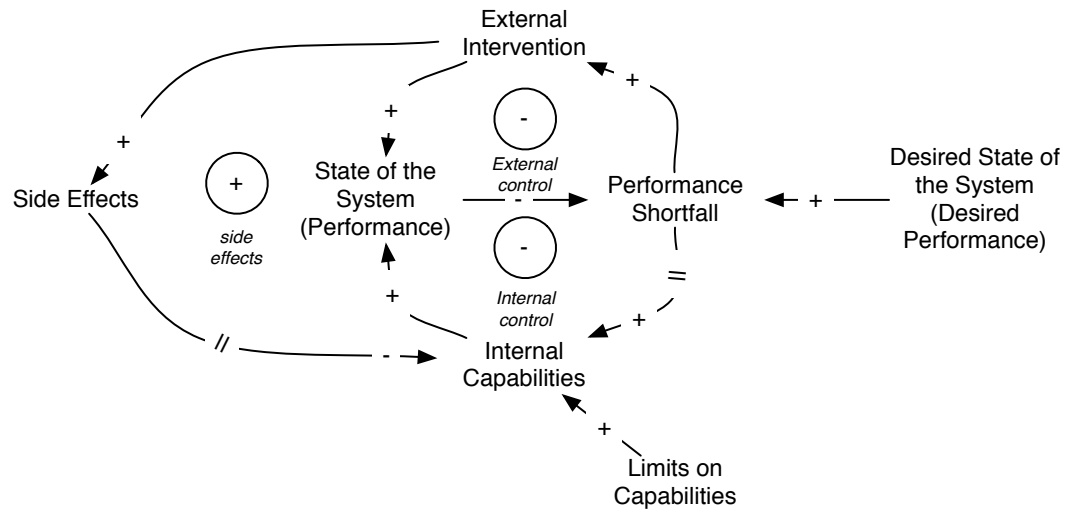


Number of loops	1
Pattern	Shifting the Burden
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	Sales environment
Reference	[26]

"Consider the following Shifting the Burden structure with a sales environment where it is perceived that interaction with the customer and closing the sale is very contingent on pre-sales support personnel that are very familiar with the product being sold. The problem being perceived as sales requires pre-sales support then offers two approaches for a solution. One approach is to hire pre-sales support personnel which can be trained to perform the pre-sales support function. This approach most definitely resolves the perceived problem. The alternative approach is to develop sales product expertise so sales is able to perform their own pre-sales support. This approach would also solve the problem.

What actually makes the hire pre-sales support more expedient than develop sales product expertise is pretty much wrapped up in a reinforcing Belief and Choice structure, that is, the structure considered to be the foundation of paradigms (interaction between our choices and our beliefs). As sales believes its responsibility is sales, as it has been for years, then it looks very unfavorably on the idea that sales should do pre-sales support. pre-sales support is the responsibility of an underling techno weenie, or so the thought goes. As such, the great resistance from sales to the develop sales product expertise idea promotes the hire pre-sales support approach. Once steps are taken to hire pre-sales support personnel it supports the no problem perception, which only serves to make the develop sales product expertise less likely. The real question is can the organization really afford to pay two people to do the job that one person should be doing?"

Shifting the Burden to the Intervener



Number of loops	3
Pattern	Shifting the Burden
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	Internal versus external control
Reference	[34]

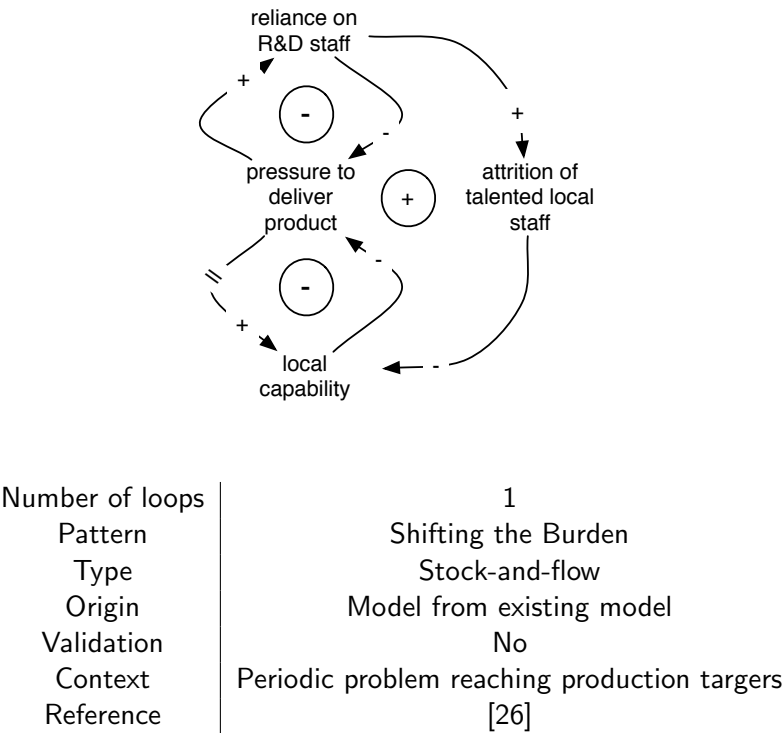
"This generic structure indicates that a decrease in the performance of the system being studied (which ever one it is) can lead to an external intervention aimed at preserving or maintaining the systems level of performance (External control) as quickly as possible. Prior to this intervention, the system maintained its function thanks to internal processes and its own capacities. This external intervention is effective only in the short term because it does not solve the fundamental problem which is causing the deficiency. In this case, in the medium term, this external intervention will progressively reduce the systems internal capacities to deal with its fundamental problem and to reorganise itself, and it will make the system increasingly dependant on the external intervention. For example, in an industrial organisation, this external intervention could be compared to emergency repairs which mobilise all the human resources. If these repairs need to be carried out too frequently, they will prevent maintenance and prevention operations from being organised and carried out, despite the fact that they could be the solution to these untimely failures. This reduction in preventive activities that were already lacking in effectiveness will only worsen the situation and will eventually lead to an increase in the number of breakdowns. But above all, the system becomes completely dependant on these emergency interventions and prevents any further resources from being allocated to proper preventive reorganisation.

The B1 loop does however indicate that an alternative to this negative spiral exists: work needs to be carried out on the fundamental problem at the origin of this decrease in performance levels. The delay in this loop means that the effects of this work on the improvement of the system may be delayed. Simulations of this type of archetype may lead to solutions that, because of this delay, will make the system worse before any improvements can be made. But when different hypotheses are simulated and solutions to this negative spiral are found, knowing and understanding why a situation might get worse before it gets better is essential. Obviously, an archetype of this kind would always take place in larger structures and would constitute loops amongst many others, but the system simulations can then be used to understand how the system in its entirety functions and evolves."

Comment from [19] on the same problem: "this is a very common variation of "Shifting the Burden" found in many circumstances. An outside entity is called in to help solve a difficult problem: a quality consultant to an organisation, a technical trainer to a rural village, a welfare program to a poor family, or a price subsidy to farmers of a particular crop. The "intervener's" role is meant to be temporary, but gradually the people with the problem become dependent on the intervention, and never learn to solve the problems themselves. This is not simply a matter of passing the buck. If the outsider could genuinely solve the problem, that would be acceptable. But the insiders, in the long run, are the only people who can make the fundamental changes necessary to solve the problem.

The intervener need not be a literal outsider. A quality consultant, for example, might be an internal expert who may indeed produce some clear gains in quality. But because the "fires were quickly put out", there is no incentive for the non experts to struggle with the quality problems from arising in the first place. The next time quality issues arise, everyone in the organisation knows they will once again depend on expert help."

Manufacturing Facility



"A manufacturing facility experiences periodic problems reaching production targets as a result of difficulties making adjustments to changing production requirements. Each time the R&D people, who know the product very well, are called upon to fix the problem. When the problem symptoms disappear, the incentive to fix the underlying problem likewise disappear. Additionally, since the production staff has received no training to improve their ability to respond to the problems, they feel disaffected and leave.

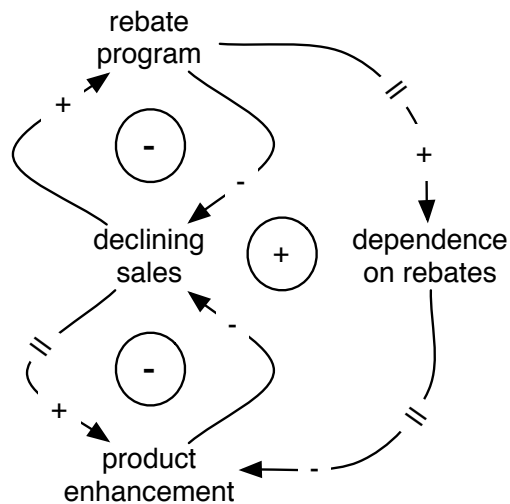
Shifting the Burden is an example of creative tension at work. The archetype draws attention to the gap between the pressures to perform in the short-term with the insights and long-term sustaining decisions to which systems managers seek to respond.

It also points to the critical importance of developing patience as one of the skills that systems managers include in their Personal Mastery of competencies. It illustrates the challenge and difficulty of demonstrating forward-thinking leadership in the face of mounting pressure to fix it and get on to the next problem.

Without a clear and convincing picture in the managers minds eye (Personal Vision) as well as in the collective minds eye of everyone (Shared Vision), the pressure to go for the quick fix may overwhelm the manager, condemning her/him to a recurring pattern of interventions that aim to solve the same set of problem symptoms."

Addiction

Dependence on Rebates



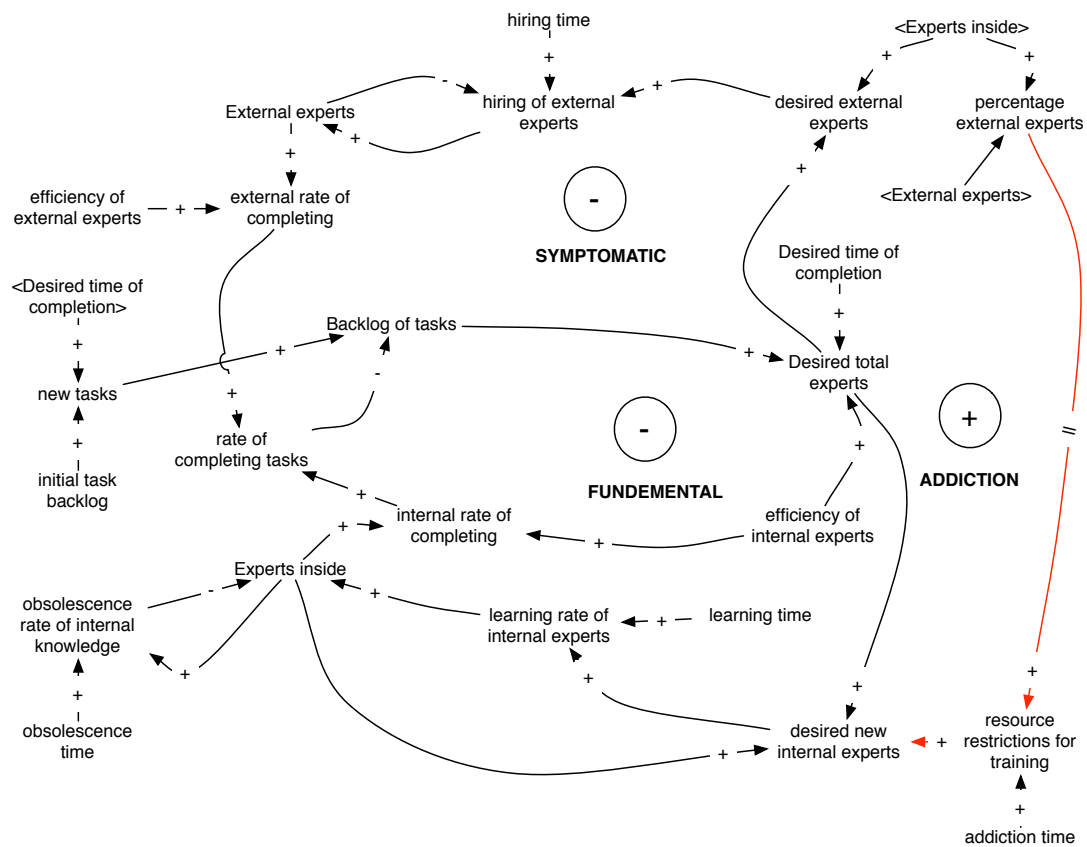
Number of loops	3
Pattern	Addiction
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	How an apparent right answer can be doubly incorrect
Reference	[26]

"We begin with an organization that is facing declining sales. This decline is very apparent and warrants immediate action. As a solution the organization implements a rebate program to boost sales, which it in fact does increase sales. When declining sales have been stemmed the rebate program is terminated and it is considered that there is no action warranted in the area of product enhancement.

After some period of time the declining sales problem reappears. Now, since a rebate program solved the problems last time, it is jumped on as the obvious answer. Because the rebate program worked last time it is figured that it will work this time, so there is still no need to consider product enhancement.

As it turns out the organization is developing a dependency on rebates as the standard answer to a declining sales situation. Yet, each time it takes a little bit more of a rebate and it has to be in effect a little longer to stem declining sales. The situation will eventually reach a point where a rebate program will no longer resolve the situation. At that time it may be too late to approach the situation from a product enhancement perspective."

Addiction



Number of loops	3
Pattern	Addiction
Type	Stock-and-flow
Origin	Model from existing model
Validation	Yes - behavior analysis (external vs internal experts)
Context	Short term fixes versus long term fixes
Reference	[24]

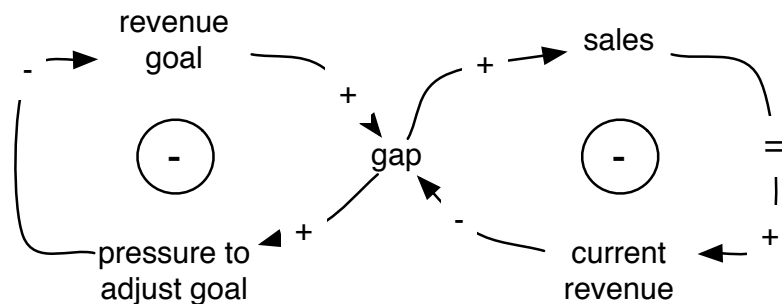
"Here a company is confronted to sophisticated commitments to its clients. A short-term fix, hiring outside experts, is shifting the problem away: it eliminates for sometime the symptoms, but at the same time, it diverts attention away from fundamental and sustainable decisions.

This policy results in a vicious addiction loop: it increases the level of external dependency while decreasing the available in-house competence. Educating internal experts would fortify the fundamental solution counteracting this evolution, and allowing the company to survive in the long term.

NB: <External experts>, <Desired time of completion> and '<Experts inside>' are there to simplify the diagram. There are just the same variables as External experts..."

Drifting Goals

Declining Sales Goals



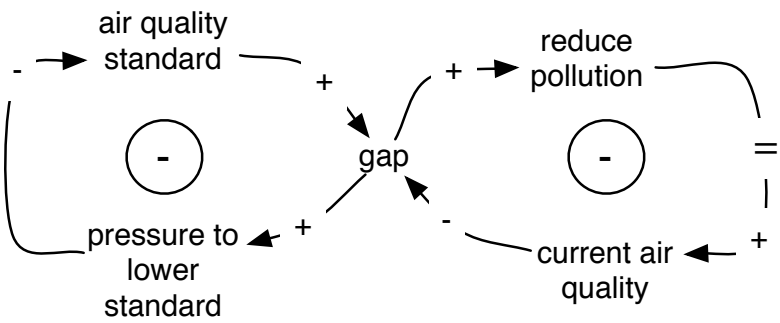
Number of loops	2
Pattern	Drifting Goals
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	How accomplishments can decline over time due to drifting goals, possibly without much awareness of what is happening
Reference	[26]

"In this situation a company establishes a revenue goal at the beginning of the year. This immediately produces a gap because of the difference between the revenue goal and the current revenue. Sales activity during the year produces a level of current revenue at the end of the year, which just happens to be less than the revenue goal.

For the next year the company establishes a revenue goal which is somewhat less than the previous year. When sales for this year produce current revenue less than the revenue goal the goal is again reduced for the next year.

In this manner, over time, the company experiences continually declining revenue, most likely with little awareness of what is happening over the long term. Organizations seem to have even shorter memories than people."

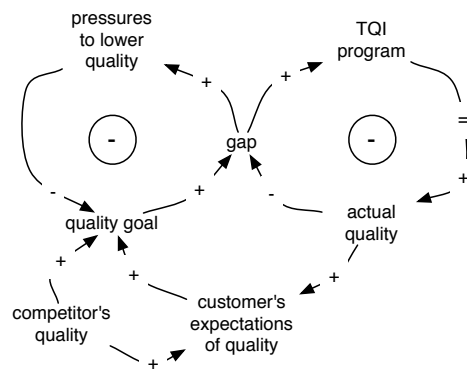
Air quality



Number of loops	2
Pattern	Drifting Goals
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	Air quality
Reference	[26]

"Goal: Air Quality
Gap: A region is in non-attainment
Pressures to adjust goal: businesses and local governments want the goals lowered – In Las Vegas, we have a problem with Carbon Monoxide being over the standard at one of the stations. The pressure is to move the station to another location in the Valley – CO is heavier than air, so if all the stations are in the higher levels of the Valley – there won't be a problem."

Internal Quality Standards



Number of loops	2
Pattern	Drifting Goals
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	Quality standards
Reference	[25]

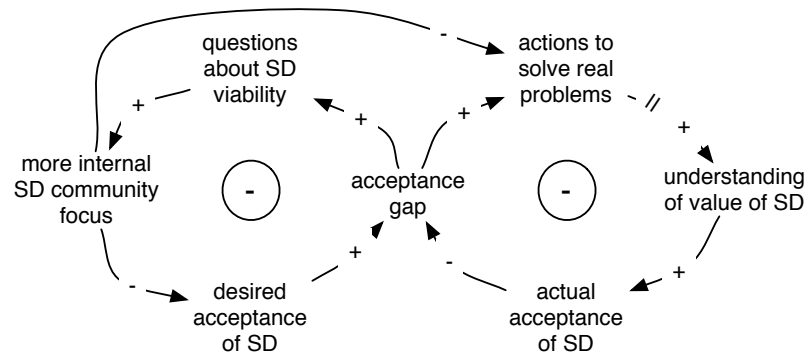
"Quality standards are common in organizations. If a gap occurs between what the organization targeted and its actual performance, a tension develops between pressure to live up to standards and the pressure to roll the standards back to something achievable. If the quality standard is anchored to an internal perception of customer expectations rather than an industry standard (what the competition is doing) there is the risk that the pressure to scale back the standard will prevail.

Eroding Goals has two important ramifications for systems managers. First, the immediate short-term effect is the failure to critically examine the underlying causes that explain why 1) performance is lacking and 2) managers feel pressure to revise goals to match what the organization is currently capable of achieving. Second, repeatedly falling into the trap of Eroding Goals eventually becomes embedded in the organizations culture as a justifiable and even reasonable thing to do. Over time, the organization falls farther and farther behind the expectations of its customers and eventually fails altogether.

On the other hand, how do managers assess whether the original goals were attainable? What about managers who repeatedly set goals that everyone knows are unattainable and uses them as catalysts to prod people into higher and higher levels of performance?

What about events in the external environment that could not have been predicted and that may be legitimate grounds for revising goals downward? What about goals that turn out to be mistakes in judgement or weaknesses in the forecasting process? Since there are (potentially) legitimate reasons to adjust goals downward, systems managers must take extreme caution when considering an adjustment to goals. The two most important considerations are 1) an honest and rigorous examination of the organization itself and 2) an equally candid look at competitors and their performance, and at customers and their expectations."

Acceptance of System Dynamics



Number of loops	2
Pattern	Accidental Adversaries
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	Acceptance of System Dynamics
Reference	[26]

"Based on a dialogue on the System Dynamics e-mail list regarding the current level of acceptance after it has been promoted for over 40 years I dredged up the following set of influences as a thought exercise. This is an example of a drifting goals structure.

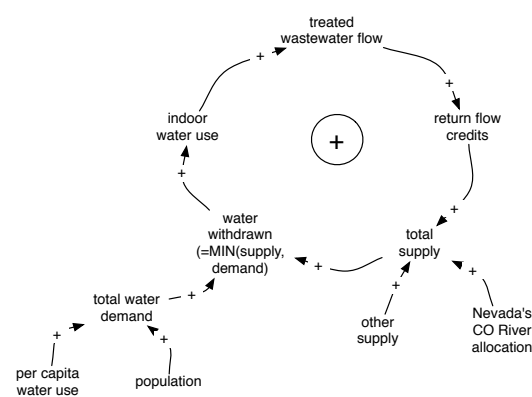
Given that there is some Actual Acceptance of SD, which is perceived to be somewhat less than the Desired, or Expected, Acceptance of SD, the two should interact to produce an Acceptance Gap. This Acceptance Gap should influence two results.

First, it should add to and increase Actions to Solve Real Problems, the kind that would add to an increased Understanding of the Value of SD. This increased Understanding of the Value of SD should then add to the Actual Acceptance of SD, thus serving to decrease the Acceptance Gap. Second, the Acceptance Gap would seem to influence SD practitioners to have some doubts, or raise Questions About SD Viability. These Questions About SD Viability should then tend to produce More Internal SD Community Focus attempting to strengthen the position and foundations of SD. Then, because there is More Internal SD Community Focus there is less focus on the Desired Acceptance of SD. And, what makes this situation even worse is that with energies being focused on More Internal SD Community Focus there is even less effort spent on Actions to Solve Real Problems that would add to the Understanding of Value of SD.

What we end up with is not only a drifting goals structure in terms of Desired SD Acceptance, but a situation that is additionally hindered from attaining the desired result by a viscous reinforcing loop which misdirects actions from what would actually move the community in the direction of its goal."

Appendix B – Other examples

Las Vegas water system

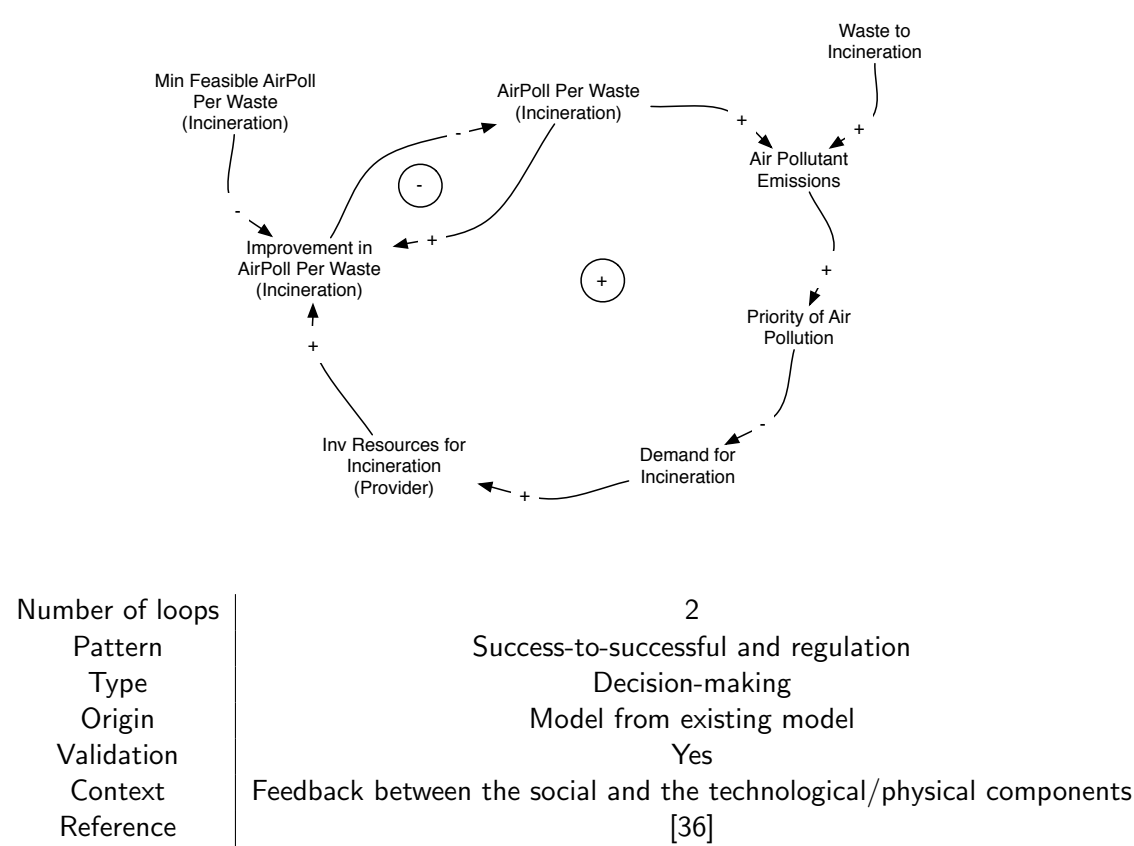


Number of loops	1
Pattern	Exponential growth
Type	Stock-and-flow
Origin	Model from existing model
Validation	Yes - behavior analysis
Context	Return flow credit mechanism
Reference	[35]

This diagram shows the major variables affecting supply and demand and their connections.

Supply changes in response to external sources, but also in response to changes in water use, through the mechanism of return flow credit. Demand increases as population increases. When water use increases, treated wastewater flow increases. Return flow credit also increases, increasing supply. But when demand increases faster than supply, demand eventually equals, the exceeds supply.

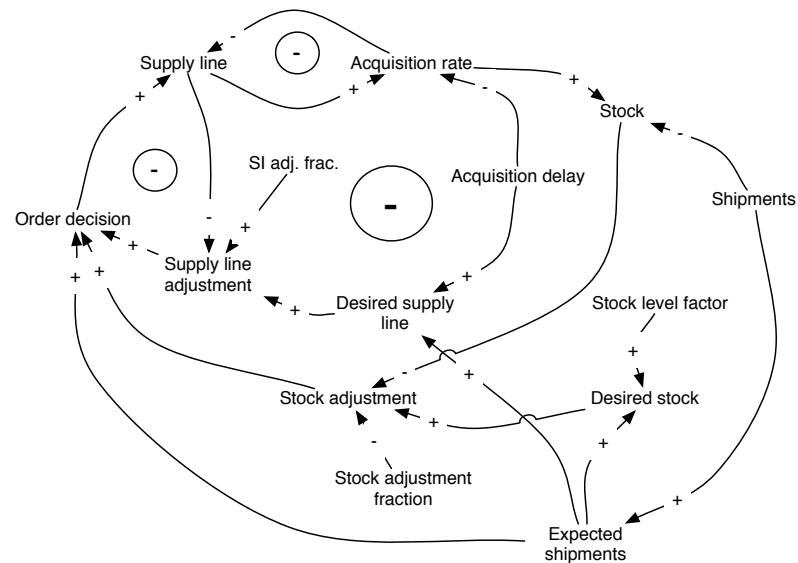
Technological development in Waste Management System



More investment in the incineration option triggers faster performance improvement. This, in turn, puts incineration to a cleaner position compared to other options. Increasing demand for cleaner incineration triggers further investment to the option. This feedback depicts the success-to-successful type of feedback loop in this specific context.

This feedback is coupled with another feedback loop that controls the development process and leads to decreasing returns in technological development.

General stock adjustment problem applied to inventory management



Number of loops	3
Pattern	Regulation
Type	Stock-and-flow
Origin	Model from existing model
Validation	Yes - behavior analysis
Context	General stock adjustment in the context of inventory order management
Reference	[9]

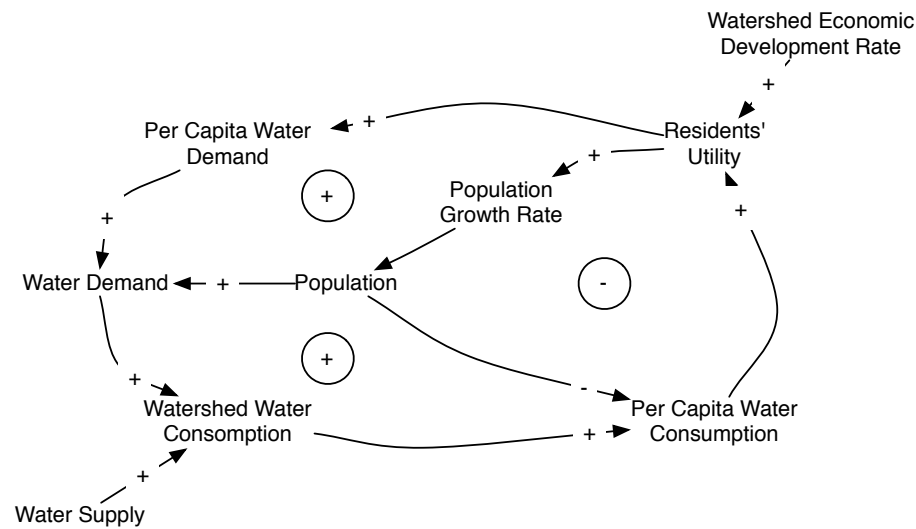
Depending on the policy used (goods in supply line are ignored in the order decision or not), the simulation of the model will show oscillations or not.

If they are ignored, the oscillations are caused by the existence of the material delay (Supply Line) between orders and the inventory.

If they are not, they prevent unnecessary over-ordering, hence yielding an improved, and more stable inventory system.

NB: the big - represents the loop Supply line \rightarrow Acquisition rate \rightarrow Stock \rightarrow Stock adjustment \rightarrow Order decision \rightarrow Supply line.

Socio-political and economics subsystem in river management



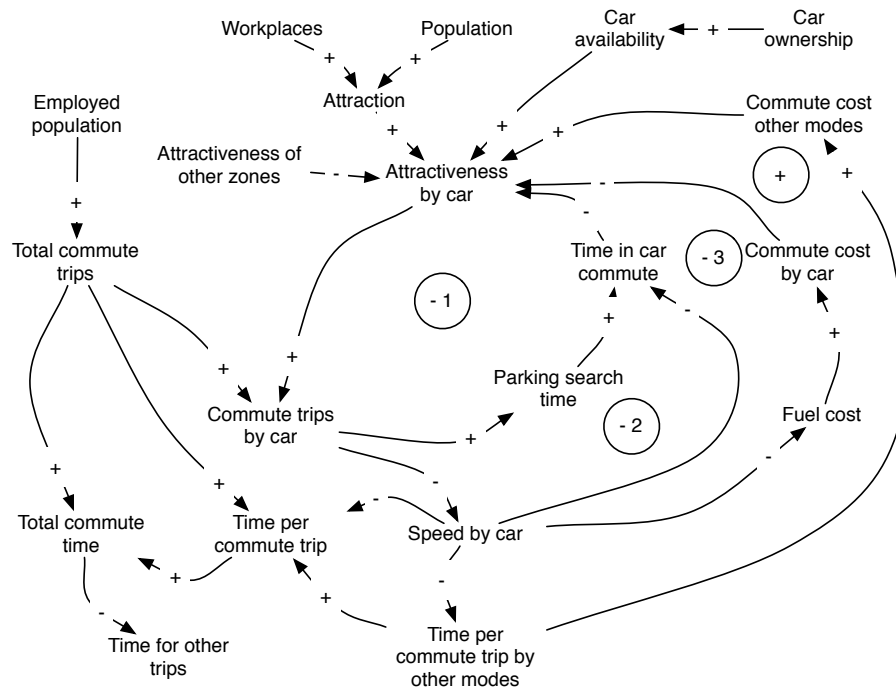
Number of loops	3
Pattern	Regulation
Type	Stock-and-flow
Origin	Model from existing model
Validation	Yes
Context	Problem for the Socio-Political and Economic Subsystem
Reference	[37]

If only the two reinforcing loops are considered, without interference of the balancing loop, residents utility, population, per capita water demand, watersheds water demand, water consumption and per capita water consumption grow or decline one after another as a result of rise or fall in the value of any of them.

However, the balancing loop interferes and changes the situation. An increase in population induces a decrease in per capita water consumption, which lower residents utility.

NB: In natural systems, balancing loops always seek neutralization of the effects of reinforcing loops, which bring an unsustainable behavior to the system and reinforcing loops rarely remain unchecked. However, in search of benefits, humans have changed the sustainable behavior of many natural system by weakening the functionality of natural-balancing systems in favor of humans without considering the disastrous effects of such actions on those natural systems. After a long period of unbalancing natural systems for various purposes, in today's world various environmental groups are struggling to bring back the original behavior of these systems by eliminating or minimizing the effects of human activities on the system and strengthening the functionality of natural balancing loops.

Transport model in the Metropolitan Activity Relocation Simulator (MARS)



Number of loops	4
Pattern	Regulation
Type	Stock-and-flow
Origin	Model from existing model
Validation	Yes
Context	Transport model
Reference	[38]

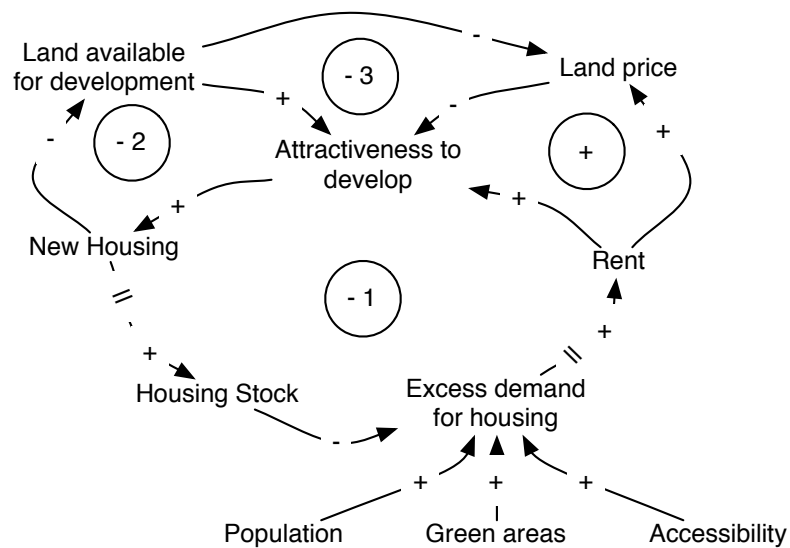
Commute trips by car increase as the attractiveness by car increases which in turn increases the search time for a parking space which then decreases the attractiveness of car use - hence balancing nature of the loop -1.

-2 represents the effect of congestion - as trips by car increase speeds decrease, times increase and so attractiveness is decreased.

-3 shows the impact on fuel costs, in our urban case as speeds increase fuel consumption is decreased.

The only reinforcing loop represents the effect of congestion on other modes. As trips by car increase, speeds by car and public transport decrease which increases costs by other modes and all other things equal would lead to a further increase in attractiveness by car.

Development of housing in the Metropolitan Activity Relocation Simulator (MARS)



Number of loops	4
Pattern	Regulation
Type	Stock-and-flow
Origin	Model from existing model
Validation	Yes
Context	Development of housing
Reference	[38]

Loop -1 shows that the attractiveness to the developer to develop in a given zone is determined by the rent which can be achieved. The level of the rent is driven by the excess demand for housing which in turn is related to the housing stock and new housing developments. As new houses are developed, the stock is increased which reduces the excess demand which then reduces the rent achievable which reduces the attractiveness to develop.

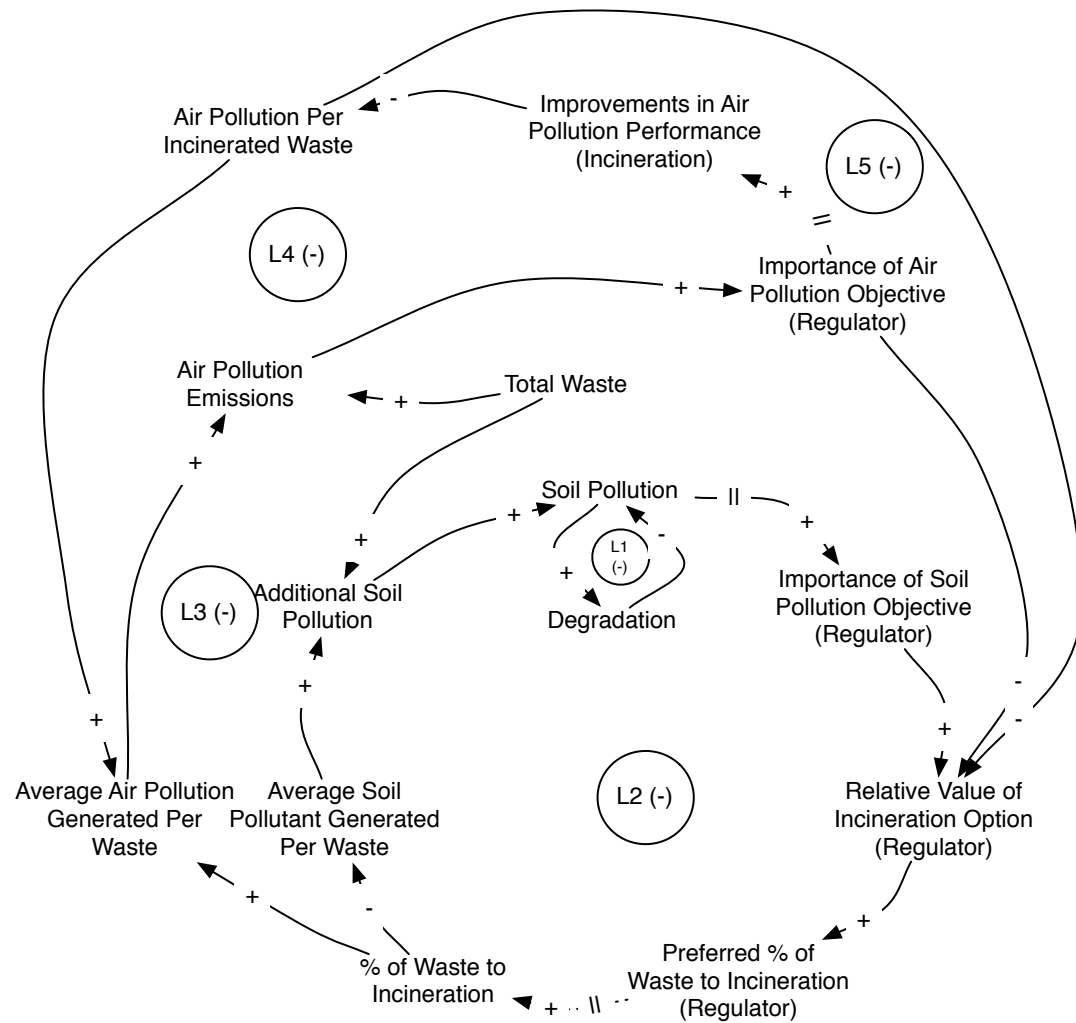
The only reinforcing loop shows that as new housing reduces the excess demand which reduces rent and hence land price which in turn makes development more attractive all other things being equal.

-2 represents the restriction of land available for development as land available is reduced then the attractiveness to develop is reduced.

-3 extends -2 to represent the effect of land availability on land price.

Finally the drivers of demand for housing are shown to be population, amount of green space and accessibility to activities from that zone.

Regulator's behavior in Waste Management System



Number of loops

5

Pattern

Regulation

Type

Information synthesis

Origin

Model from existing model

Validation

Yes

Context

Influence of the observed behavior of the regulator

Reference

[36]

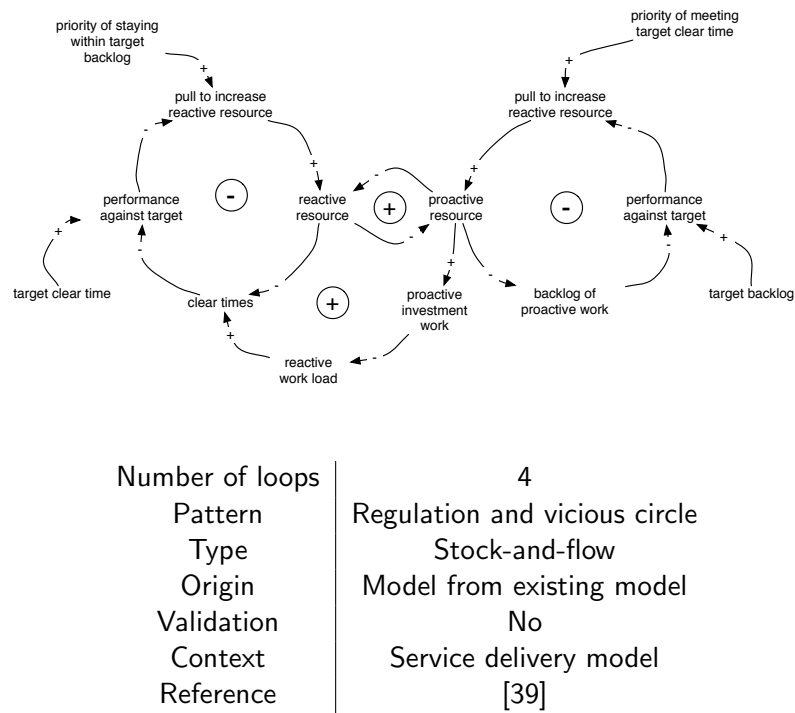
At the beginning, the loop L1 exerts some control over the level of soil pollution, which can be seen as a balancing act against the increase of the soil pollution level. However, as a consequence of the increase in the total waste to be handled, the growth in the soil pollution exceeds the levels that can be suppressed via degradation mechanism.

The observed increase in soil pollution triggers L2, the second balancing loop. According to this loop, an actor changes its priorities with a time delay following the recognition of the increase in the soil pollution. The actor therefore changes the assessment of the options, and incineration becomes more favorable. This induces further changes for other actors. In the absence of other loops, the expected consequence would be that L2 will drive the system to a point where the desired percentage of land-filling for the regulator is zero without losing any pace. However, before that point is reached other feedback loops are triggered in order to balance the shift to incineration driven by L2.

This shift results in a significant increase in the air pollution emissions. This triggers L3, which tries to balance the dynamics caused by L2 through increased priority of the regulator for air pollution. The consequence of this activation is worsening of the evaluation of the incineration option and a decrease in pace of the shift in the regulators preferences towards incineration.

However, the activation of L3 initiates other counteracting loops; L4 and L5. They are both related to the performance improvement mechanism, activated as a consequence of the air pollution issue that is gaining importance in the regulatory arena. These loops can also be interpreted as the defensive mechanism of the incineration niche to keep itself as a favorable option in the system. L4 balances the rise of air pollution level that was increasing due to the shift towards incineration. On the other hand, L5 attempts to balance the decrease in the assessment score of the option due to a gain in priority of the air pollution issue.

Service delivery

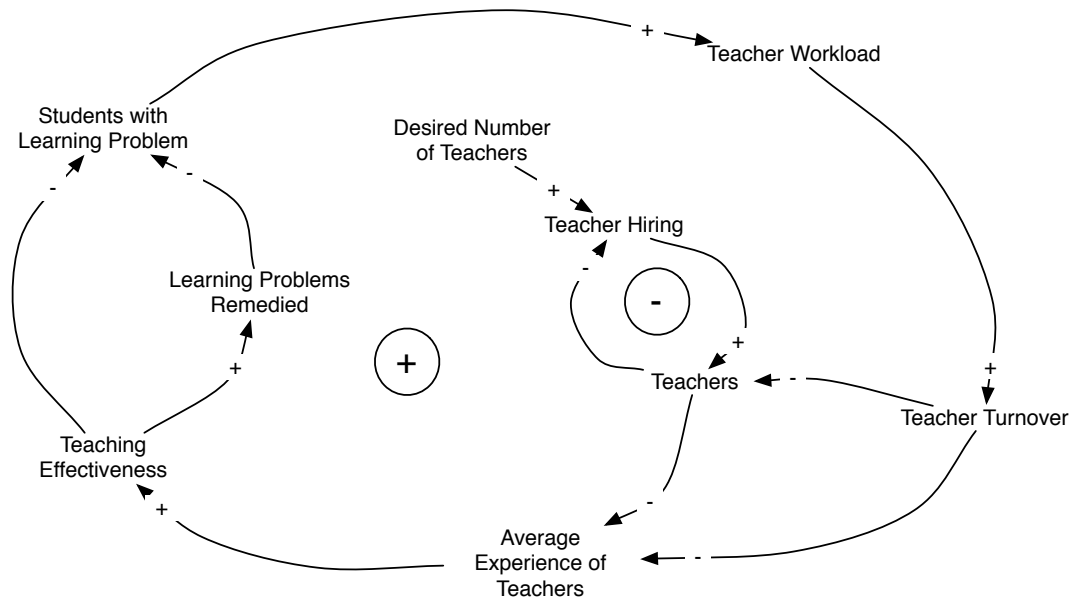


The loop on the left drives how resource levels associated with reactive work are adjusted to drive the clear time towards a target value. The clear time period is the period from when a task enters the system to when it is completed. The lower the resource level, the longer the clear times - hence poorer performance against target and an increased pull for more resources, on which in turn would shorten the clear times and so on.

On the right of the picture, there is a similar balancing loop related to proactive work. Because reactive and proactive share a lot of resources, the reactive and proactive loops are joined by a reinforcing loop in the middle of the figure. (More resource on reactive work means less resource on proactive work and vice versa).

However, the two loops are also connected by the reinforcing loops at the bottom of the figure related to the impact of proactive investment work. The lower the proactive resource, the less proactive maintenance work gets done. Since one of the major benefits of proactive work is to lower the network fault rate, less maintenance means higher faults rates and hence more reactive repair work to be done. And since more reactive work leads to a greater pull of resources away from proactive to reactive, this in turn means even less proactive work is possible. This can become a vicious circle of increasingly poor performance. However, it can be turned around to a virtuous circle if the direction of change is driven the other way.

Student learning problems and teacher turnover

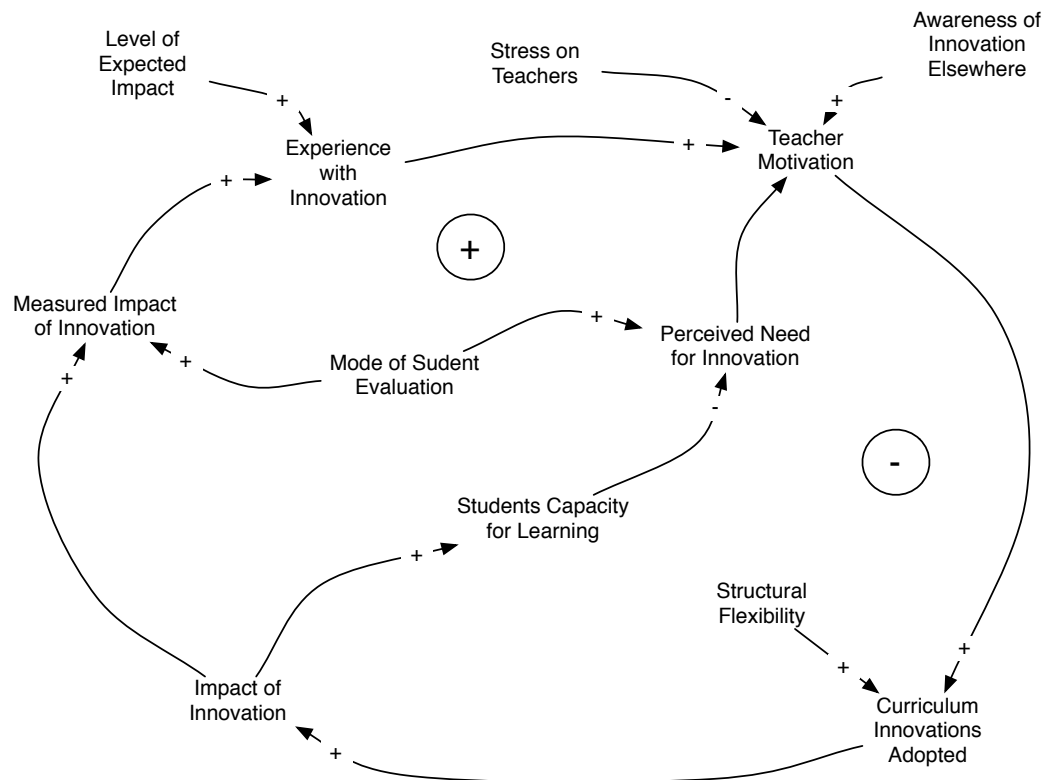


Number of loops	2
Pattern	Decision-making
Type	Model from existing model
Origin	No
Validation	Model of how to help students with learning problems while maintaining a supply of effective teachers
Context	[14]
Reference	

The negative loop is just a simple regulation of the number of teacher with a certain limit (desired number of teachers).

The positive loop shows how the system deal with students with learning problems and how this problem affects the teacher turnover.

Factors affecting teacher motivation

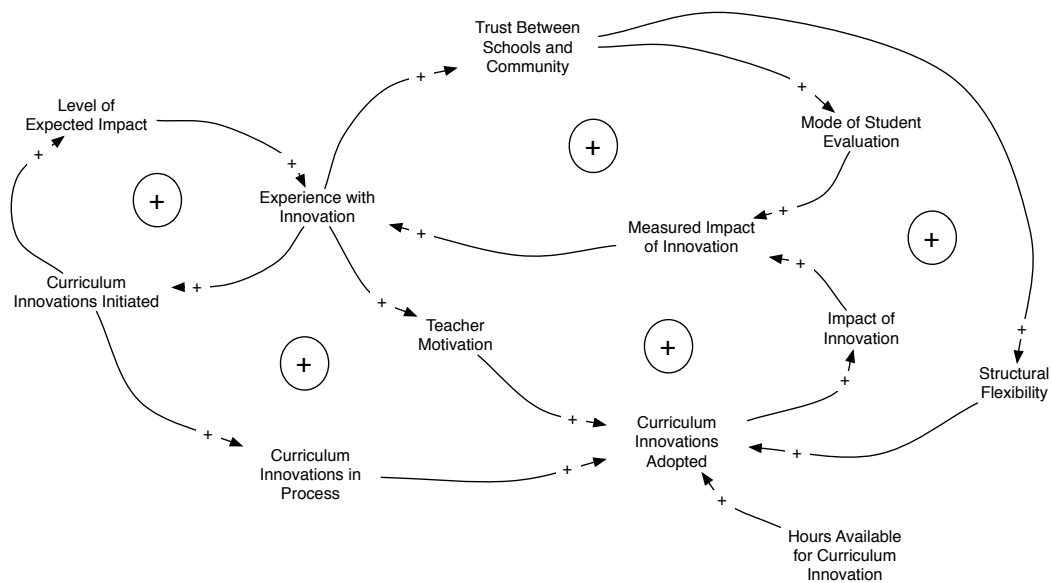


Number of loops	2
Pattern	Slow but efficient decision maker
Type	Decision-making
Origin	Model from existing model
Validation	Yes - see Factors affecting curriculum innovation
Context	Relationships among experience with innovation teacher motivation, and curriculum innovations are presented
Reference	[14]

The big loop form a reinforcing loop, in which a high level of teacher motivation will improve the chances of innovations being adopted and lead to a better experience with innovation and a continued high level of motivation and receptivity to innovations in the future.

Conversely, poor experience with innovation will reduce motivation and make it difficult to sustain effort on those innovations or have future innovations adopted.

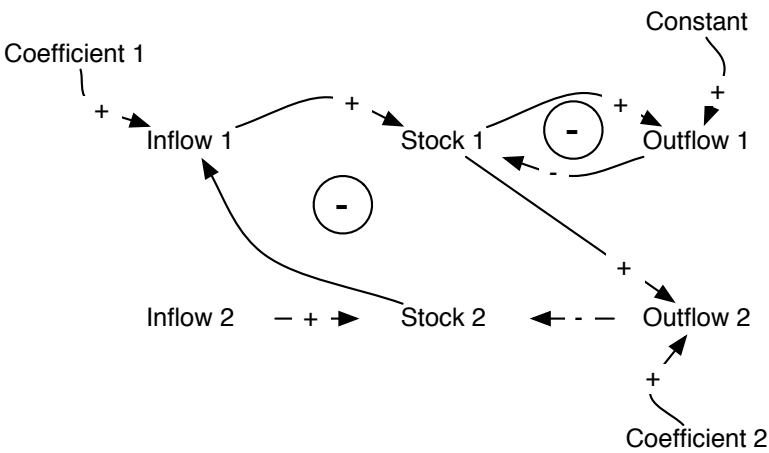
Factors affecting curriculum innovation



Number of loops	5
Pattern	Decision maker with amplified impact of failure
Type	Decision-making
Origin	Model from existing model
Validation	Yes - analysis of impact on innovation with other variables
Context	How trust between schools and the community interacts with the other variables
Reference	[14]

Additional reinforcing loops through this trust variable enable success to build on success and cause failure to initiate a downward spiral that becomes an impediment to future innovation.

Elementary 2-stock negative loop oscillator

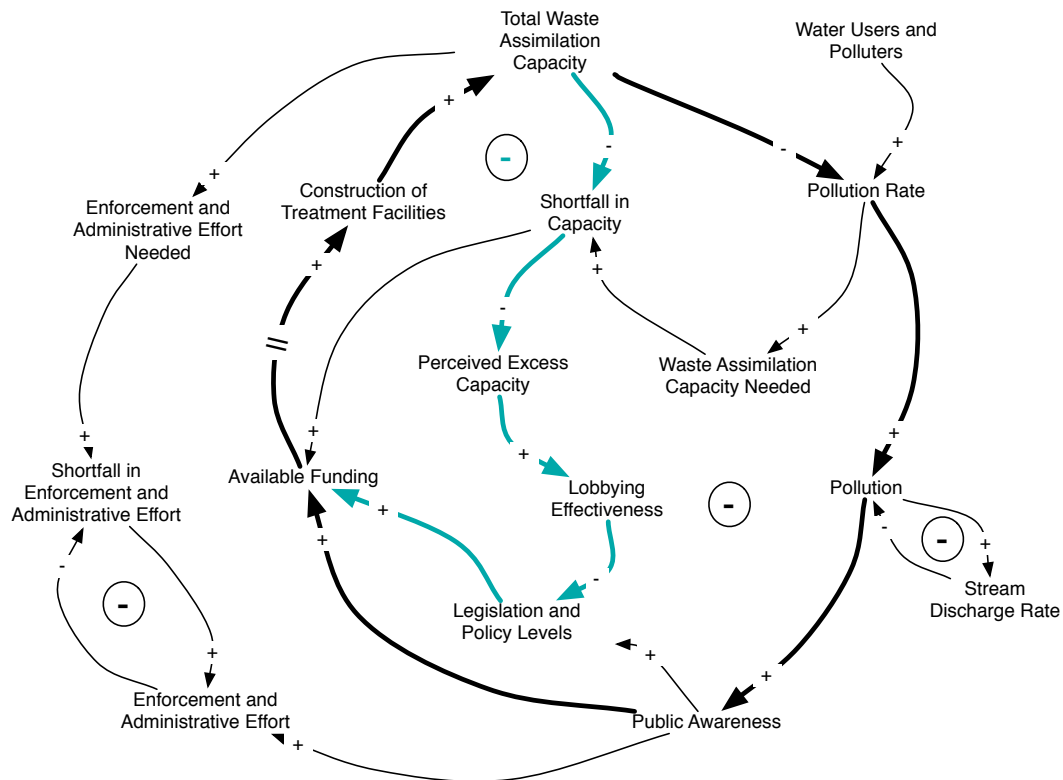


Number of loops	2
Pattern	Oscillator
Type	Stock-and-flow
Origin	Model from existing model
Validation	Yes - behavior analysis
Context	Consequences of having delays in structures
Reference	[9]

A very typical managerial application of delays is in the context of the standard goal-seeking structure.

The behavior of the model is pure exponential goal seeking. But if delays exist anywhere around the goal-seeking loop, then the system has the potential to oscillate.

Water quality dynamics

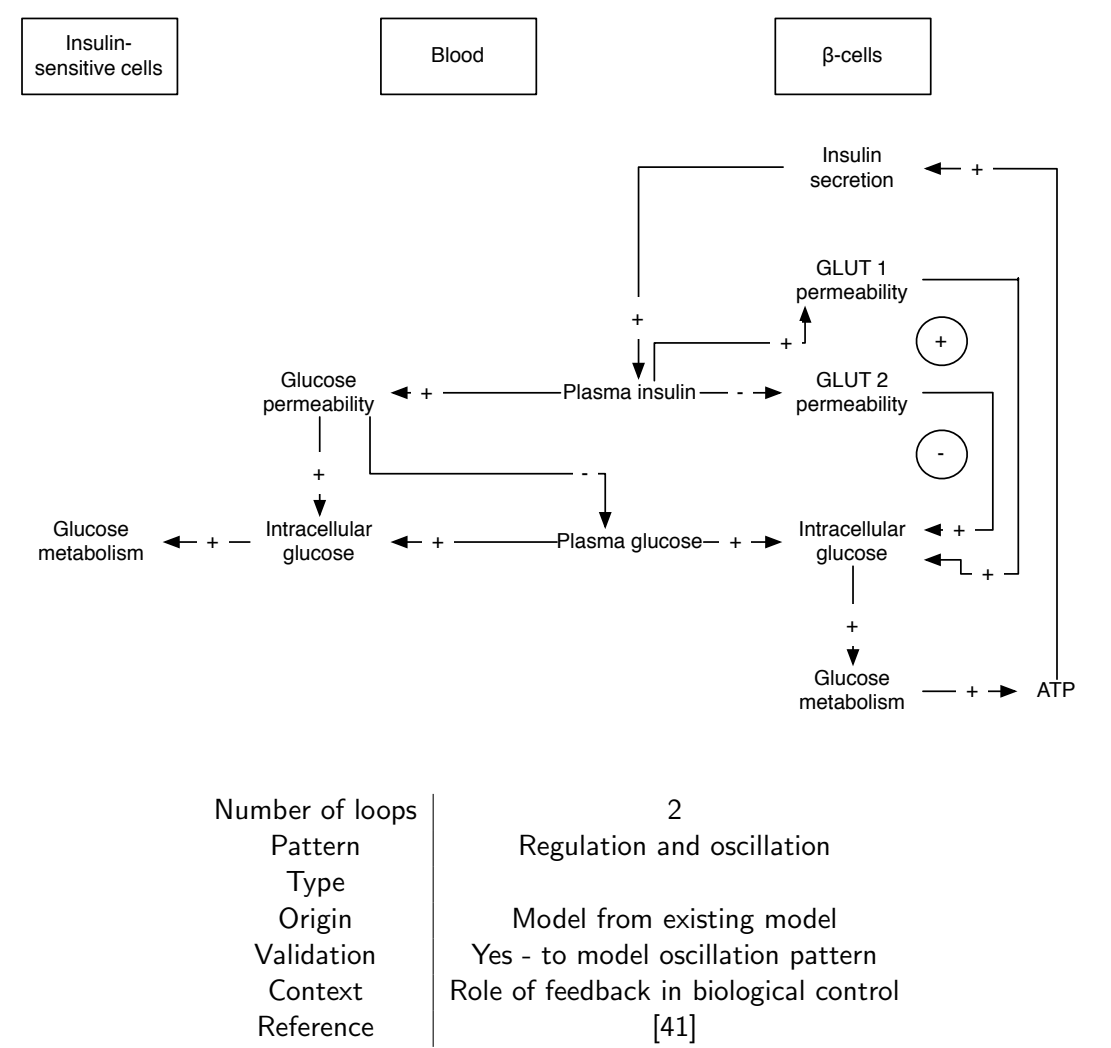


Number of loops	4
Pattern	Regulation and oscillation
Type	Stock-and-flow
Origin	Model from existing model
Validation	No
Context	Water resources management
Reference	[40]

The big loop in thick dark arrows explains how an increase in water pollution over time leads to an increase in water treatment. Due to system delays (double line), this balancing loop is likely to result in oscillating pollution levels over time.

The other big loop, in thick light arrows, shows how lobbyists (polluters) use the existing waste assimilation capacity to affect a change in public policy to their advantage.

Feedback in Transport Systems: Insulin



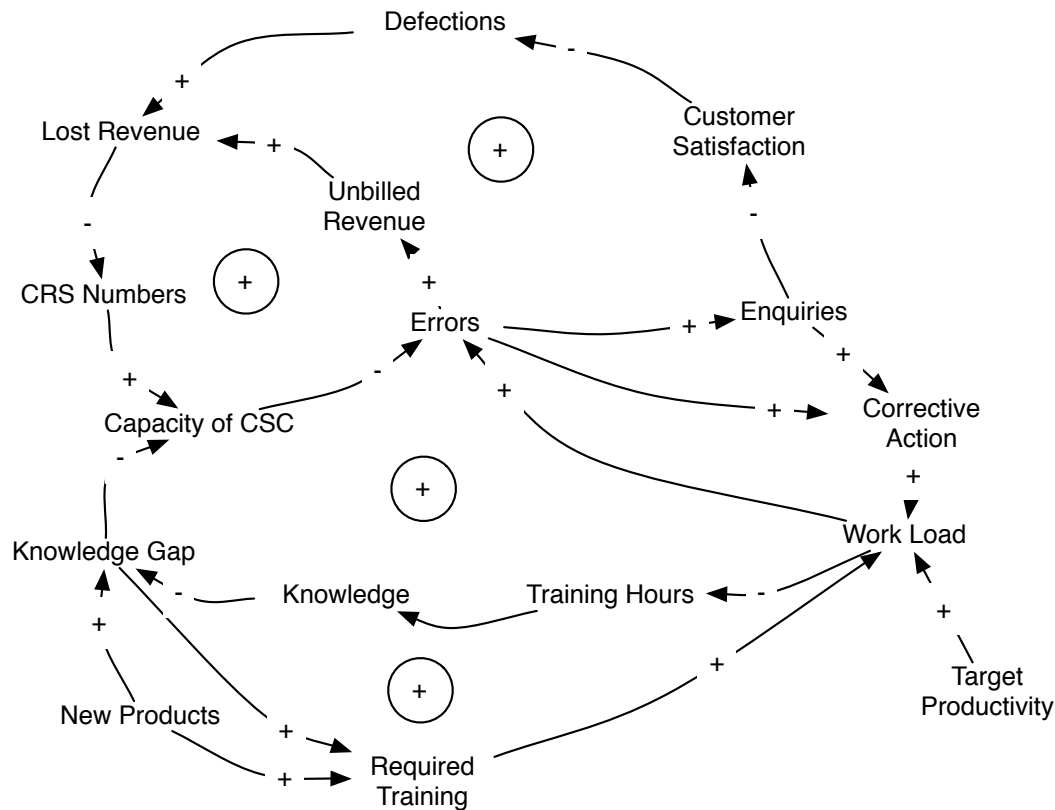
Transport processes can regulate the level of agonists that in turn can affect the regulatory process, giving rise to feedback effects. The regulation of insulin secretion is a good illustration of the role of feedback in biological control.

Insulin has the same effect on β -cells that it does on other cells in the body: it alters the permeability of the β -cell membrane to glucose. The β -cell uses two glucose transporters; one of them (GLUT 1) is activated by extracellular insulin, while the other (GLUT 2) is down-regulated.

The activation of GLUT 1 is a form of positive feedback, since the effect of the hormone on this carrier favors an increase in intracellular glucose, which promotes increased insulin secretion. Correspondingly, the down-regulation of GLUT 2 constitutes negative feedback.

The combination of positive and negative feedback leads to oscillations in insulin secretion.

Product Provisioning Process



Number of loops	4
Pattern	Vicious circles
Type	Stock-and-flow
Origin	Model from existing model
Validation	Yes
Context	Dynamics between Information process and organizational system
Reference	[42]

Focus on the effect of pre-existing policies and practices on an already implemented process. The interest is the dynamics between Information Services process and the organizational system into which it has been placed.

It is apparent that although the computer process works as designed a number of other softer, organizational issues are affecting the behavior of the entire process.

For example, an increase in errors leads to an increase in workload, which leads to a decrease in available training time, leading to a decrease in knowledge, which leads to an increase in the knowledge gap, which leads to a decrease in the capability of the CSC which in turn leads to an increase in errors,... This is an example of vicious circle.

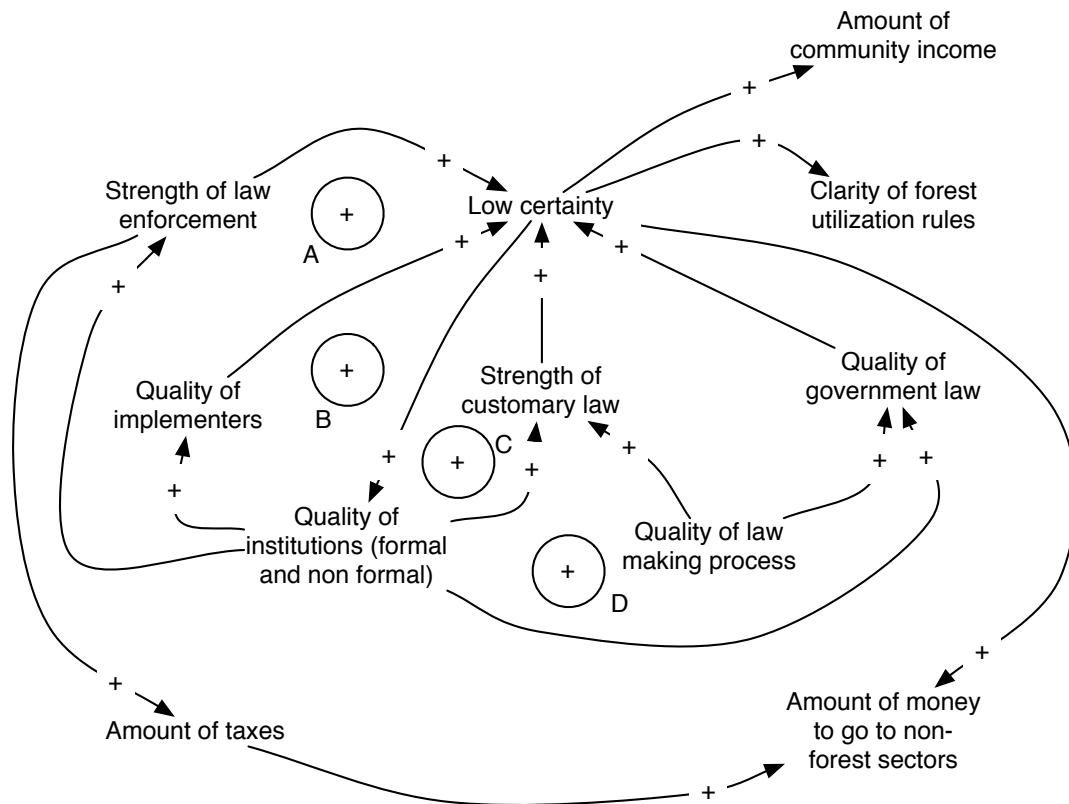
One management policy was to increase Target Productivity in the short term that will trigger the vicious cycle invalidating managements decision. The immediate effect is a reduction in

the backlog of work, but due to the time delay involved in the loop, the long term effect is an increase in errors and the size of the error bucket.

The validation of the system was used to determine which of the policies and practices are having the most effect on the provisioning process, and which offer the most effective leverage to gain an improvement in the overall process.

NB: CSC= customer service centers and CSR = customer services representatives

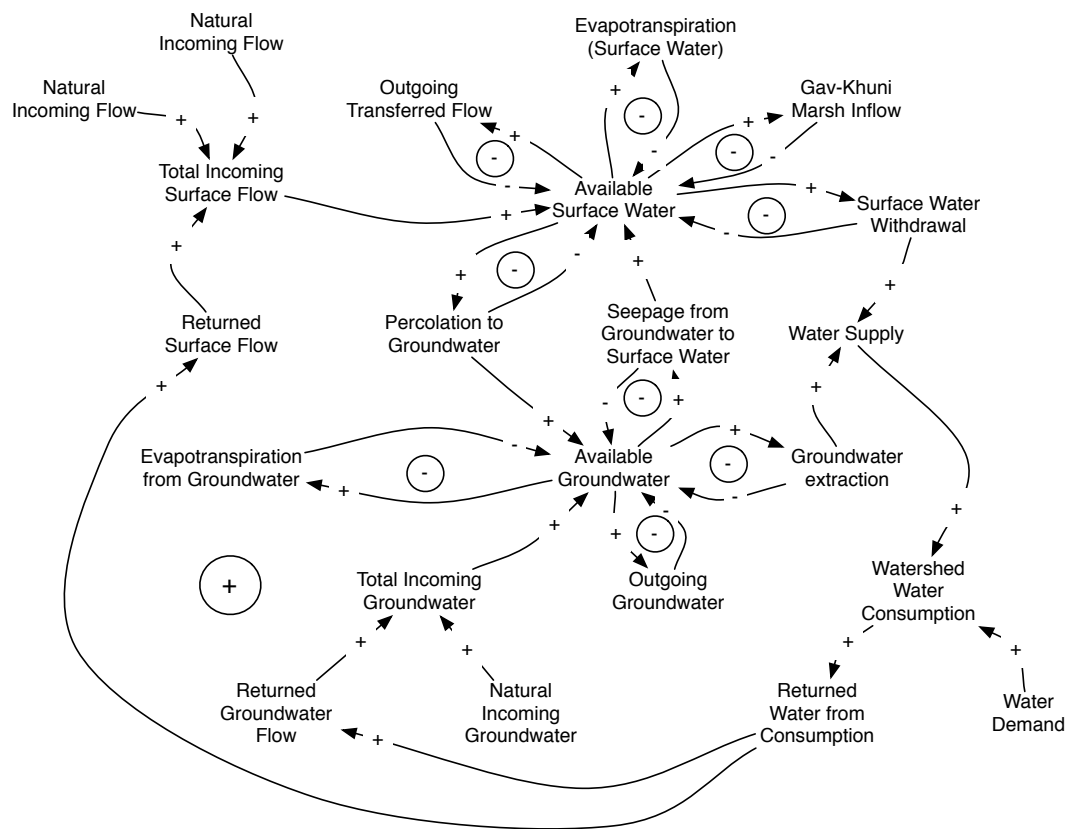
The inter-relationship between forest laws and rules



Number of loops	4
Pattern	Vicious circles
Type	Information synthesis
Origin	Model from existing model
Validation	No
Context	Forest management
Reference	[43]

Law certainty is the level of transparency, persistence and enforcement of law perceived by stakeholders. It influences quality of forest utilization rules, community income, and re-investment of forest taxes in other sectors. Quality of institutions and law-making processes need to be improved in order to improve the law certainty. Participants also perceived that improvements in law certainty would improve the quality of institutions. This leads to four feedback loops involving Quality of institutions, Strength of law enforcement, and Law certainty. Any improvement in the Quality of institutions is thought to be a self-sustaining investment, as any improvement will, after some lag time, tend to lead to further strengthening of the same institutions.

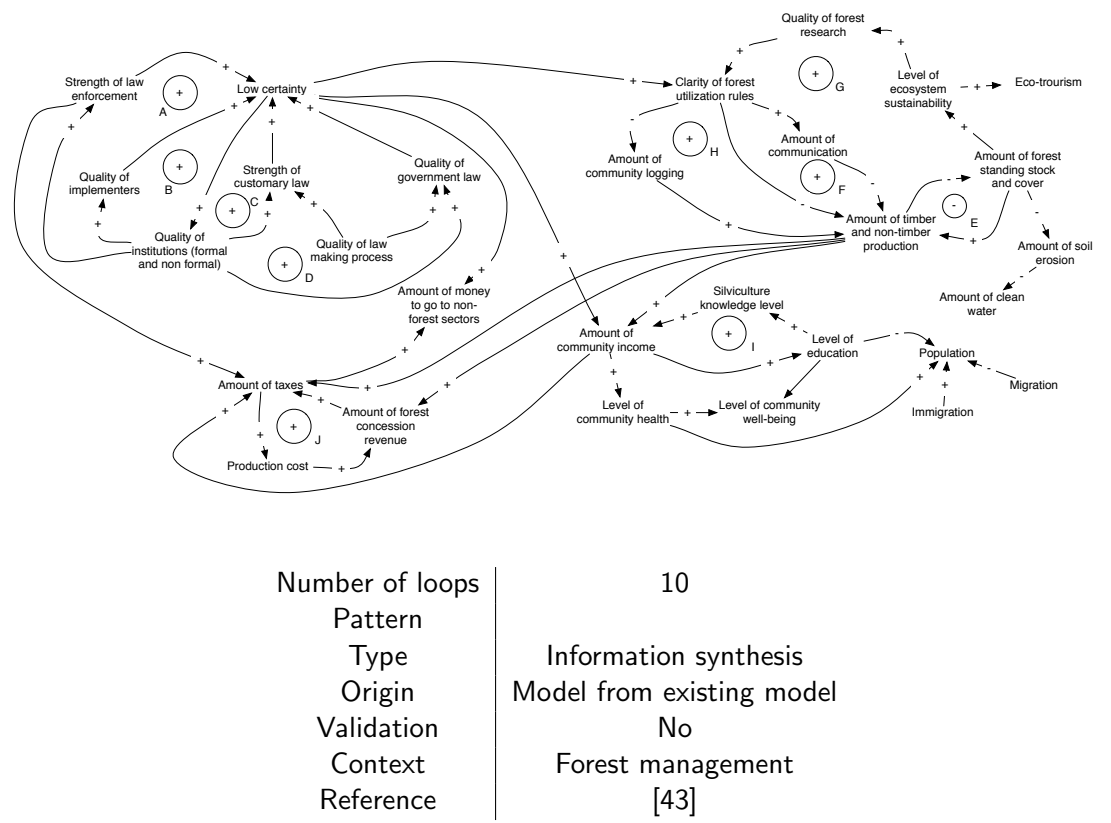
Hydrologic subsystem



Number of loops	10
Pattern	Regulation
Type	Stock-and-flow
Origin	Model from existing model
Validation	Yes
Context	Problem for the Hydrologic subsystem
Reference	[37]

There is one big reinforcing loop Available surface water → Surface Water Withdrawal → Water Supply → Watershed Water Consumption → Returned Water from Consumption → Returned Surface Flow → Total Incoming Surface Flow → Available Surface Water that is regulated with lots of small balancing loops.

Forest law and rules

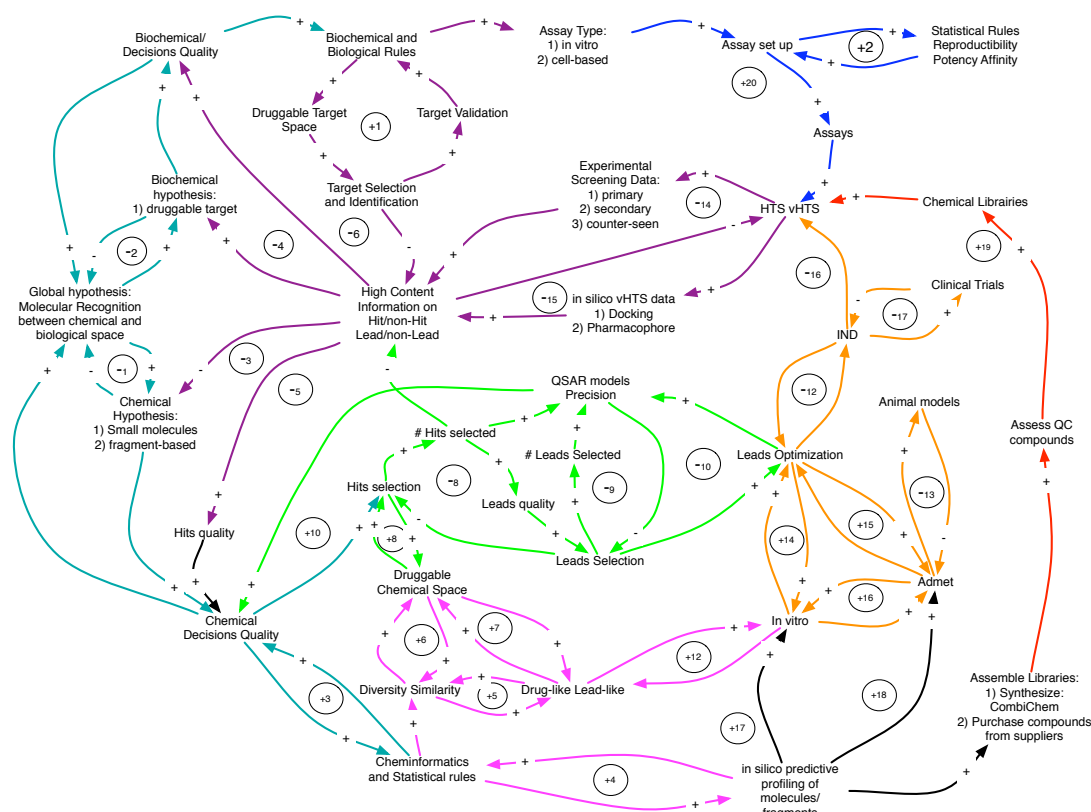


The Amount of forest standing stock and cover and Amount of community income were also selected by the participants as key factors within feedback loops. They identified a negative loop (E) between Amount of timber and non-timber production and Amount of forest standing stock and cover. They saw scope for this production to be tempered by Clarify of utilization rules (F) and Amount of communication (G), both of which involve positive feedbacks.

Participants also noted that community logging influenced the amount of timber and non-timber production and increased the Amount of community income.

Amount of community income and Amount of taxes were also identified as key indicators. The feedback loop (I) highlights the role of education in increasing community income. We can also see that the expectation that better education will foster small planned families rather than unplanned families, as a result of the governments family planning campaign in schools.

Drug discovery



Number of loops

32

Pattern

Type

Origin

Validation

Context

Reference

Model from existing model

No

System dynamic structure (SDS) from hit-to-lead
to optimization to investigational new drug in the HTS process
[45]

The drug discovery process comprises five main steps:

- building of libraries of small molecules
- optimization and validation of biochemical assays
- automation to perform HTS screening
- primary screening to generate data for use in hits identification and confirmation screening
- secondary screening and counter screening for the selection of leads series from the initial pool of hits.

This structure can be split into three categories:

- chemical and biochemical hypotheses feedback loops
- from H2L to IND and molecule attrition feedback loops
- lead series success on attrition rate feedback loops

More information in the article

Appendix C – Simulator.oz

```
1  functor
2  import
3      System
4      Application
5      OS
6
7  export
8      Start
9
10 define
11     {OS.srand 0}
12
13     %% Input: Loop is the record defined in Graph.oz of a project
14     %% Loop has to contain the following features: agents, graph, engine, monitor
15     proc {Start Loop}
16         ListOfAgents in
17             ListOfAgents = {CreateAgents Loop.agents Loop.engine Loop.monitor}
18             {CreateLinks Loop.graph}
19             {StartAgents ListOfAgents Loop.engine Loop.monitor}
20         end
21
22         %% Creates the topology sort organised in layers
23         %% Input: Agents is list of all the agents of the system
24         fun {Topology Agents}
25             Topo = {NewCell nil}
26             Dico = {NewDictionary}
27             %% Gives the layers of the 'instantaneous' agents
28             fun {Loop Agents Acc}
29                 %% Agents name are used to be stores in the dictionary
30                 LayerName = {NewCell nil}
31                 LayerAgent = {NewCell nil}
32             in
33                 for Agent in Agents do
34                     Stock = {Agent is_stock($)}
35                     Name = {Agent name($)}
36                     Ins = {Agent ins($)} Y
37                 in
38                     if {Not Stock} then
39                         Y = for In in Ins collect:C do
40                             InName = {In name($)} in
41                                 {C {Dictionary.member Dico InName}}
42                             end
43                             %% If the agent entries are in the dictionary
44                             %% and that the agent is not in the dictionary
45                             %% then it is added to the layer
46                             if {Not {Member false Y}} andthen
47                                 {Not {Dictionary.member Dico Name}} then
48                                 LayerName := Name|@LayerName
49                                 LayerAgent := Agent|@LayerAgent
50                             end
51                         end
52                     end
53                     %% If the layer is empty then all the agents are in the dictionary
54                     case @LayerName of nil then Acc
55                     else
56                         for Name in @LayerName do
```

```

57         {Dictionary.put Dico Name Name}
58     end
59     {Loop Agents @LayerAgent|Acc}
60 end
61 end
62 in
63     %% First find all the stocks...
64     for Agent in Agents do
65         Stock = {Agent is_stock($)}
66         Name = {Agent name($)}
67     in
68         if Stock then
69             Topo := Agent|@Topo
70             {Dictionary.put Dico Name Name}
71         end
72     end
73     %% ... then the other layers are added
74     Topo := @Topo|{Reverse {Loop Agents nil}}
75     @Topo
76 end
77
78 %% User has to define the agent as follows:
79 %% Var_name#Agents.class_name#init(name [attributes])#first_value
80 %% Input: Agents is the list of all the agents
81 %% Engine is sync(Delta) or async(Time Random Delta)
82 %% Monitor is info(list: [list of agent names]) times: int)
83 fun {CreateAgents Agents Engine Monitor}
84     List in List = {NewCell nil}
85     for Agent in Agents do
86         case Agent of Var#Class#Init#Out then
87             case Engine of sync(Delta) then
88                 Var = {CreateSync Class Init Out Monitor Delta}
89                 [] async(Time Random Delta) then
90                     Var = {CreateAsync Class Init Time Out Monitor Random Delta}
91                 end
92                 List := Var|@List
93             end
94         end
95     @List
96 end
97
98 %% Input: Graph is the list of Agents with their input and ouput agents
99 %% Var_name#[list of input agents]#[list of output agents]
100 proc {CreateLinks Graph}
101     for Agent in Graph do
102         case Agent of Name#In#Out then
103             {Name set_ins(In)}
104             {Name set_outs(Out)}
105         else {System.show 'Agent connection problem'}
106         end
107     end
108 end
109
110 %% Input: Agents is the list of all agents
111 %% Engine is sync(Delta) or async(Time Random Delta)
112 %% Monitor is info(list: [list of agent names]) times: int)
113 proc {StartAgents Agents Engine Monitor}
114     for Agent in Agents do
115         {Agent start}
116     end
117     case Engine
118     of sync(Delta) then
119         {System.show times#{Int.toFloat Monitor.times}}
120         Tri = {Topology Agents} in
121             %% Sends initial values to agents
122             for Agent in Agents do
123                 {Agent send_value(init)}
124             end
125             %% Execute each layer a certain number of times
126             for I in 1..{FloatToInt {IntToFloat Monitor.times}/Delta} do
127                 for Layer in Tri do

```

```

128         X Y in
129         Y = for Agent in Layer collect:C do
130             {C {Agent compute(X $)}}
131         end
132         X = unit
133         {ForAll Y Wait}
134     end
135 end
136 %% Stops the agents
137 for Agent in Agents do
138     End in
139     {Agent stop(End)}
140     {Wait End}
141 end
142 [] async(Time Boolean Delta) then
143     {System.show times#{Int.toFloat Monitor.times}}
144     %% Thread that waits before stopping the agents
145     for I in 1..{FloatToInt {IntToFloat Monitor.times}/Delta} do
146         if Boolean then {Delay Time + Time div 10}
147         else {Delay Time} end
148     end
149     %% Stops the agents
150     for Agent in Agents do
151         End in
152         {Agent stop(End)}
153         {Wait End}
154     end
155 end
156 {Application.exit 0}
157 end
158
159 %% Synchronous agent as an active object
160 fun {CreateSync Class Init Out Monitor Delta}
161     class Sync from Class
162         attr time ins_stream
163         meth init(Init Out Delta)
164             Class,Init
165             delta := Delta
166             out_stream := [Out]
167         end
168         meth start
169             @ins_stream = stream()
170             for Agent in @ins do
171                 if Agent \= none then
172                     Name in Name = {Agent name($)}
173                     ins_stream := {Adjoin @ins_stream {Record.make stream [Name]}}
174                     @ins_stream.Name = {NewCell nil}
175                 end
176             end
177         end
178         meth send_value(Value)
179             for Agent in @outs do
180                 if Agent \= none then
181                     if Value == init then
182                         {Agent new_value(@name Out)}
183                     else
184                         {Agent new_value(@name Value)}
185                     end
186                 end
187             end
188         end
189         meth new_value(Name Value)
190             {Assign @ins_stream.Name Value|{Access @ins_stream.Name}}
191         end
192         meth compute(X Y)
193             {Wait X}
194             In NewValue in In = {Record.make ins {Arity @ins_stream}}
195             for Name in {Arity @ins_stream} do
196                 List in List = {Access @ins_stream.Name}
197                 case List
198                 of X|nil then In.Name = X

```

```

199         [] X|_ then In.Name = X
200         else In.Name = 0.0 end
201     end
202     {self m(In NewValue)}
203     out_stream := NewValue|@out_stream
204     {self send_value(NewValue)}
205     Y = unit
206 end
207 end
208 Obj = {New Sync init(Init Out Delta)}
209 P
210 in
211 thread S in {NewPort S P}
212   for M in S do {Obj M} end
213 end
214 if {Member {Obj name($)} Monitor.list} then
215   {Obj monitored(true)}
216 else
217   {Obj monitored(false)}
218 end
219 proc {$ M} {Send P M} {Delay 1} end
220 end
221
222 %% Asynchronous agent as an active object
223 fun {CreateAsync Class Init Time Out Monitor Random Delta}
224   class Async from Class
225     attr time msg insName
226     meth init(Init Time Out Random Delta)
227       Class,Init
228       delta := Delta
229       if Random then
230         Ten in
231         if Time div 10 < 1 then Ten = 1
232         else Ten = Time div 10 end
233         if {OS.rand} mod 2 == 0 then
234           time := Time + ({OS.rand} mod Ten)
235         else
236           time := Time - ({OS.rand} mod Ten)
237         end
238       else
239         time := Time
240       end
241       out_stream := [Out]
242       insName := nil
243     end.
244     meth start
245       for Agent in @ins do
246         Name in {Agent name(Name)}
247         insName := Name|@insName
248       end
249       {self construct_msg}
250     end
251     meth construct_msg
252       msg := {Record.make state @insName}
253       msg := {Adjoin @msg {Record.make state [1]}}
254       @msg.1 = {NewCell 0}
255       {self send_requests}
256     end
257     meth send_requests
258       for Agent in @ins do
259         {Agent state_request(self)}
260       end
261     end
262     meth state_request(Agent)
263       {Agent state_answer(@name @out_stream.1)}
264     end
265     meth state_answer(Name Info)
266       @msg.Name = Info
267       {Assign @msg.1 ({Access @msg.1} + 1)}
268       if {Access @msg.1} == {Length @ins} then
269         Out in

```

```

270         if @stock andthen {OS.rand} mod 2 == 1 then
271             {Thread.preempt {Thread.this}}
272         end
273         {self m(@msg Out)}
274         out_stream := Out|@out_stream
275         if @stock then thread {Delay @time} {self construct_msg} end
276         else {self construct_msg} end
277         else skip end
278     end
279 end
280 Obj = {New Async init(Init Time Out Random Delta)}
281 P
282 in
283     thread S in {NewPort S P}
284         for M in S do {Obj M} end
285     end
286     if {Member {Obj name($)} Monitor.list} then
287         {Obj monitored(true)}
288     else
289         {Obj monitored(false)}
290     end
291     proc {$ M} {Send P M} {Delay 1} end
292 end
293 end

```

Appendix D – Simulation results

Population

```
1 functor
2 import
3     Agent at 'BaseAgent.ozf'
4
5 export
6     Births
7     Deaths
8     Population
9
10 define
11 class Births from Agent.baseAgent
12     attr rate
13     meth init(Name Rate)
14         Agent.baseAgent,init(Name)
15         rate := Rate
16     end
17     meth m(In ?Out)
18         Out = @rate * In.population
19     end
20 end
21 class Deaths from Agent.baseAgent
22     attr rate
23     meth init(Name Rate)
24         Agent.baseAgent,init(Name)
25         rate := Rate
26     end
27     meth m(In ?Out)
28         Out = @rate * In.population
29     end
30 end
31 class Population from Agent.baseAgent
32     attr popl
33     meth init(Name Popl)
34         Agent.baseAgent,init(Name)
35         popl := Popl
36         stock := true
37     end
38     meth m(In ?Out)
39         popl := @popl + (In.births - In.deaths) * @delta
40         Out = @popl
41     end
42 end
43 end
```

LISTING 1: Population: Agents.oz

```

1 functor
2 import
3     Agents at 'Agents.ozf'
4     Simulator at 'Simulator.ozf'
5
6 define
7 Graph
8 Births
9 Deaths
10 Population
11 in
12 Graph = loop(agents: [Population#Agents.population#init(population 100.0)#100.0
13                     Births#Agents.births#init(births 0.06)#6.0
14                     Deaths#Agents.deaths#init(deaths 0.03)#3.0]
15               graph: [Population#[Births Deaths]#[Births Deaths]
16                     Births#[Population]#[Population]
17                     Deaths#[Population]#[Population]]
18               engine: async(100 false 1.0)
19               monitor: info(list: [population] times: 100))
20
21     {Simulator.start Graph}
22 end

```

LISTING 2: Population: Graph.oz

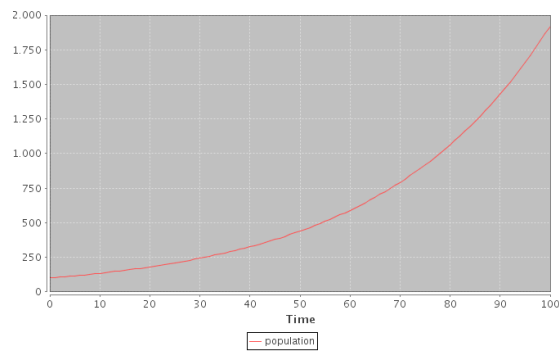
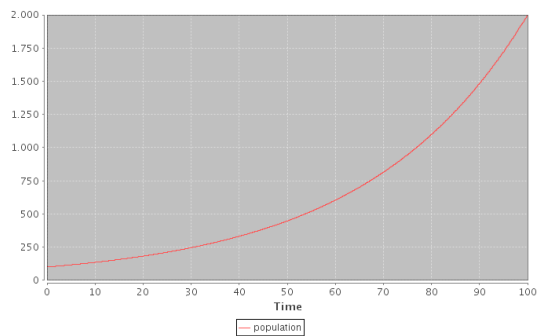
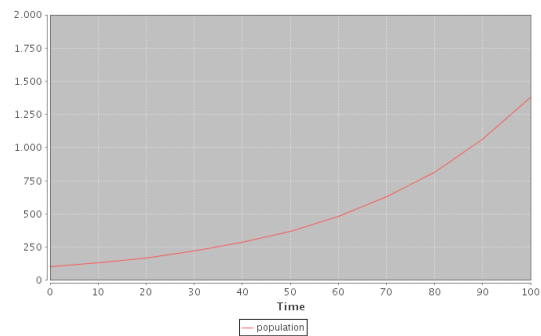
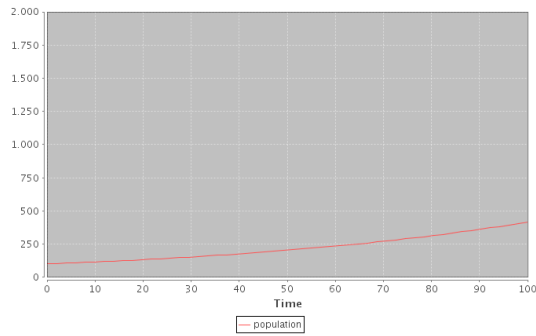
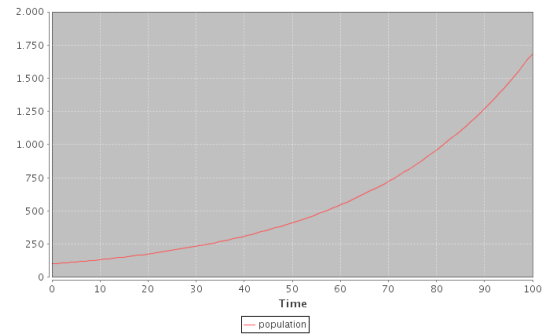
(a) $\delta = 1$ (b) $\delta = 0.1$ (c) $\delta = 10$

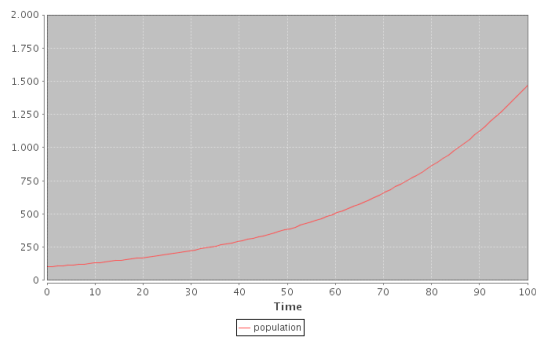
FIGURE 1: Simulator: the population model – synchronous engine



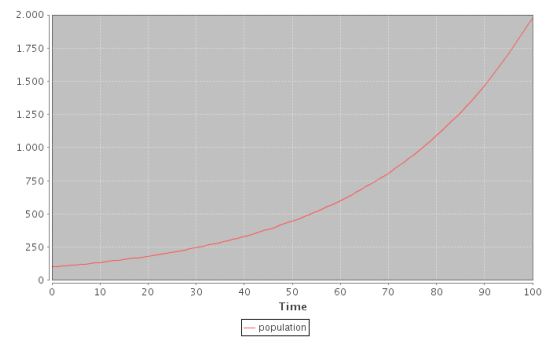
(a) time = 10 ; random time = false



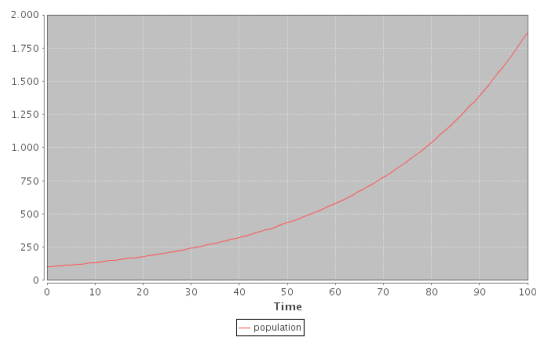
(b) time = 10 ; random time = true



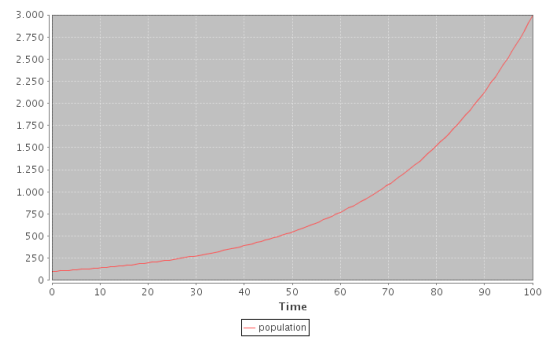
(c) time = 100 ; random time = false



(d) time = 100 ; random time = true

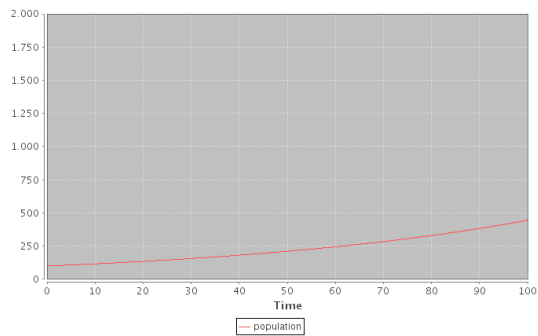


(e) time = 500 ; random time = false

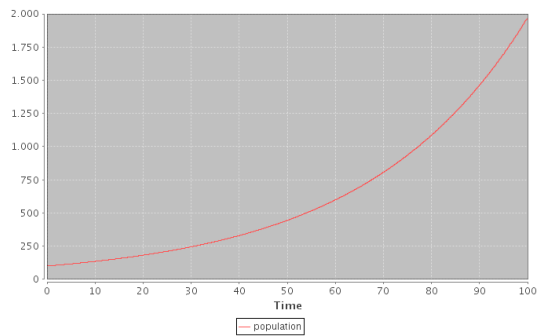


(f) time = 500 ; random time = true

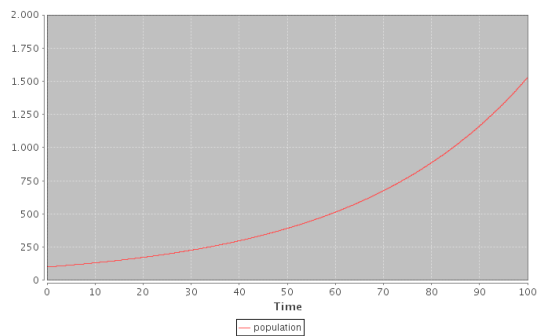
FIGURE 2: Simulator: the population model – asynchronous engine ($\delta = 1$)



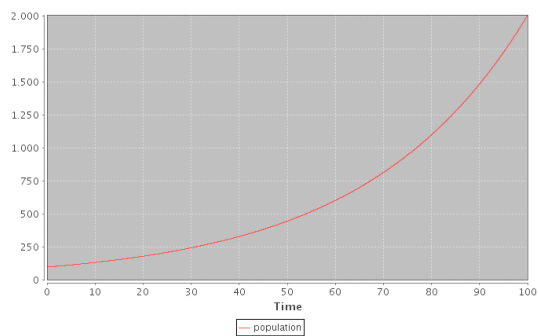
(a) time = 10 ; random time = false



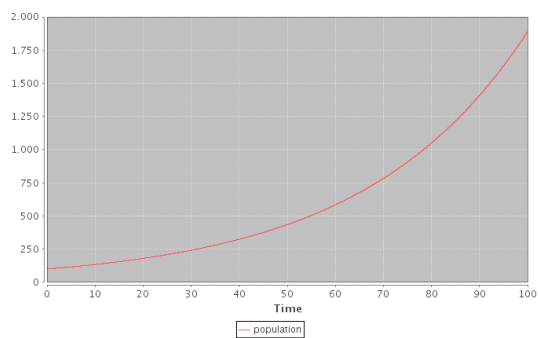
(b) time = 10 ; random time = true



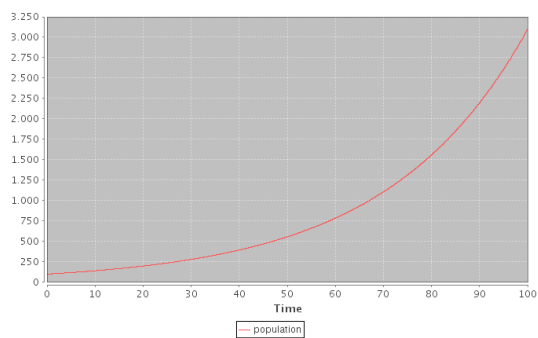
(c) time = 100 ; random time = false



(d) time = 100 ; random time = true

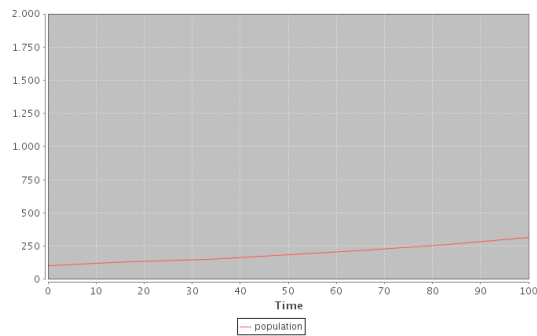


(e) time = 500 ; random time = false

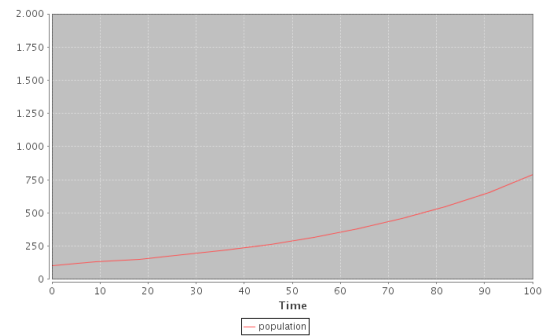


(f) time = 500 ; random time = true

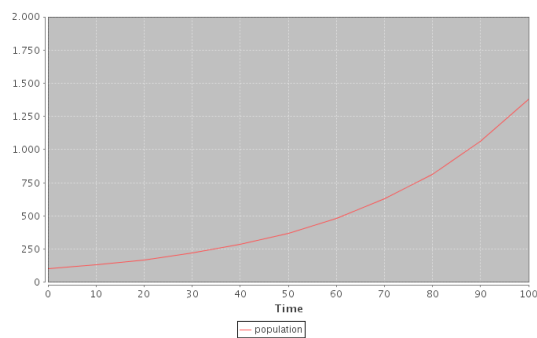
FIGURE 3: Simulator: the population model – asynchronous engine ($\delta = 0.1$)



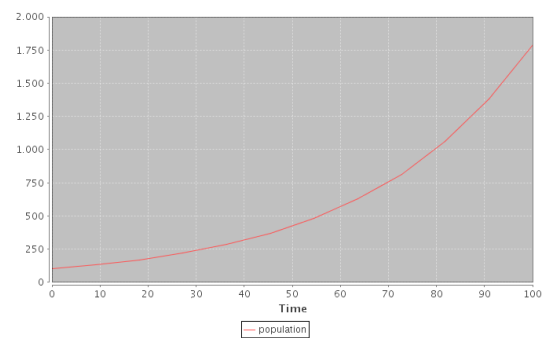
(a) time = 10 ; random time = false



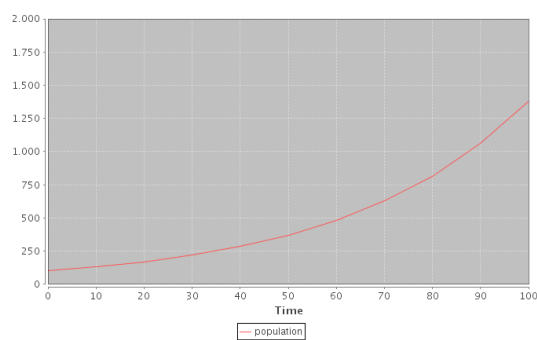
(b) time = 10 ; random time = true



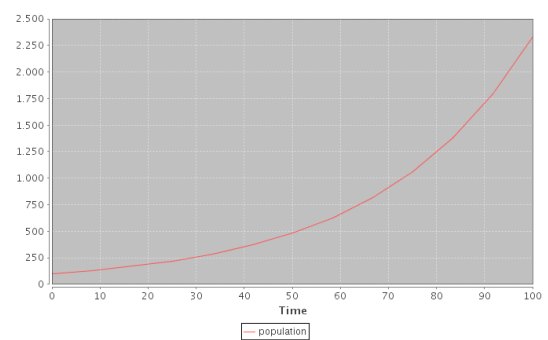
(c) time = 100 ; random time = false



(d) time = 100 ; random time = true



(e) time = 500 ; random time = false



(f) time = 500 ; random time = true

FIGURE 4: Simulator: the population model – asynchronous engine ($\delta = 10$)

Drifting goals

```

1  functor
2  import
3      Agent at 'BaseAgent.ozf'
4
5  export
6      Desired
7      Action
8      Current
9      Gap
10     Emotional
11
12  define
13  class Desired from Agent.baseAgent
14      attr attribute
15      meth init(Name Value)
16          Agent.baseAgent,init(Name)
17          attribute := Value
18          stock := true
19      end
20      meth m(In ?Out)
21          attribute := @attribute + (~ In.emotional) * @delta
22          Out = @attribute
23      end
24  end
25  class Action from Agent.baseAgent
26      meth m(In ?Out)
27          Out = 0.02 * In.gap
28      end
29  end
30  class Current from Agent.baseAgent
31      attr attribute
32      meth init(Name Value)
33          Agent.baseAgent,init(Name)
34          attribute := Value
35          stock := true
36      end
37      meth m(In ?Out)
38          attribute := @attribute + In.action * @delta
39          Out = @attribute
40      end
41  end
42  class Gap from Agent.baseAgent
43      meth m(In ?Out)
44          Out = In.desired - In.current
45      end
46  end
47  class Emotional from Agent.baseAgent
48      meth m(In ?Out)
49          Out = 0.1 * In.gap
50      end
51  end
52  end

```

LISTING 3: Drifting goals: Agents.oz

```

1  funcion
2  import
3      Agents at 'Agents.ozf'
4      Simulator at 'Simulator.ozf'
5
6  define
7  Graph
8  Desired
9  Action
10 Current
11 Gap
12 Emotional
13 in
14 Graph = loop(agents: [Current#Agents.current#init(current 10.0)#10.0
15                      Action#Agents.action#init(action)#1.8
16                      Desired#Agents.desired#init(desired 100.0)#100.0
17                      Emotional#Agents.emotional#init(emotional)#9.0
18                      Gap#Agents.gap#init(gap)#90.0]
19              graph: [Current#[Action]#[Gap]
20                      Action#[Gap]#[Current]
21                      Desired#[Emotional]#[Gap]
22                      Emotional#[Gap]#[Desired]
23                      Gap#[Desired Current]#[Emotional Action]]
24              engine: sync(4.0)
25              monitor: info(list: [desired current] times: 60))
26
27      {Simulator.start Graph}
28  end

```

LISTING 4: Drifting goals: Graph.oz

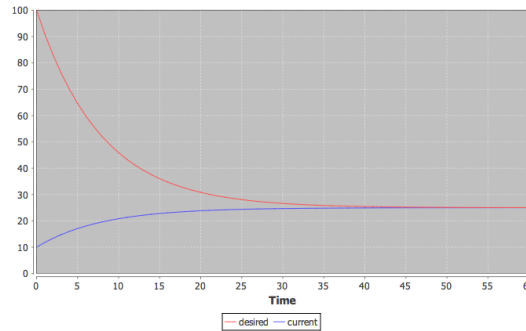
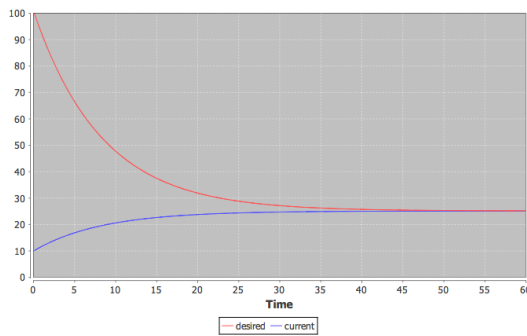
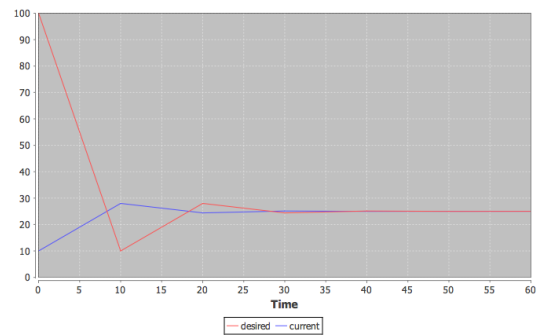
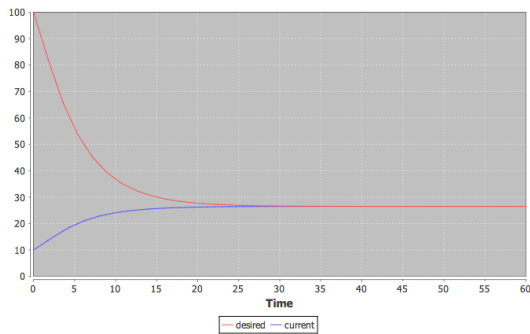
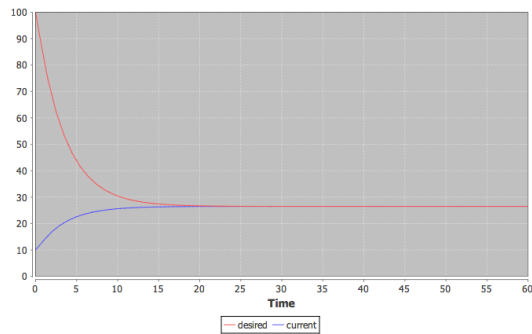
(a) $\delta = 1$ (b) $\delta = 0.1$ (c) $\delta = 10$

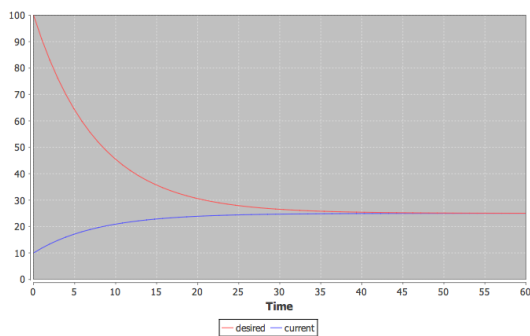
FIGURE 5: Simulator: the drifting goals model – synchronous engine



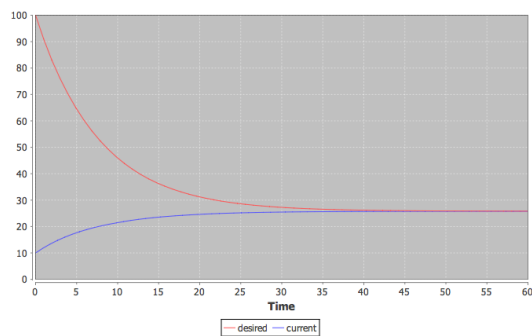
(a) time = 10 ; random time = false



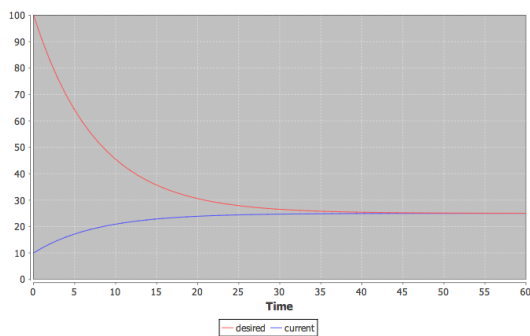
(b) time = 10 ; random time = true



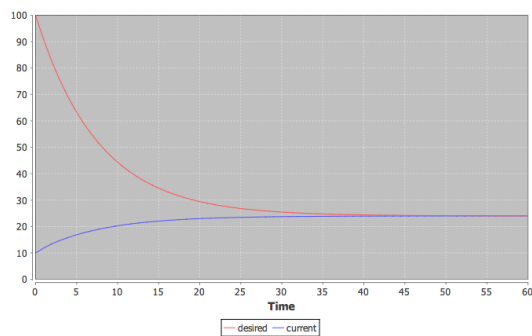
(c) time = 100 ; random time = false



(d) time = 100 ; random time = true

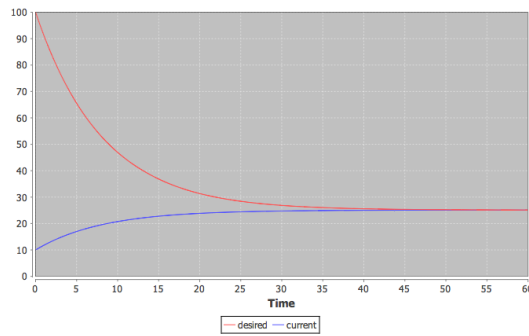


(e) time = 500 ; random time = false

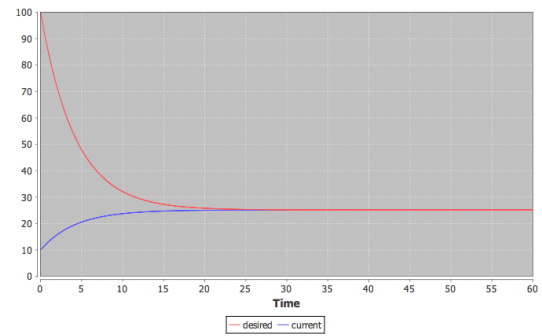


(f) time = 500 ; random time = true

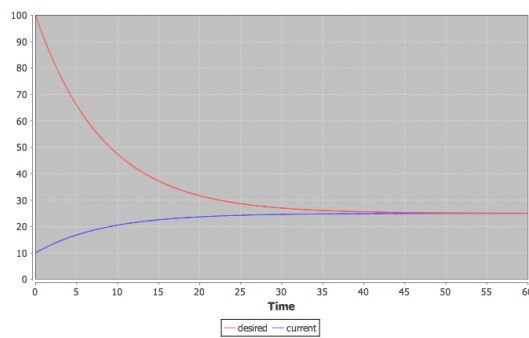
FIGURE 6: Simulator: the drifting goals model – asynchronous engine ($\delta = 1$)



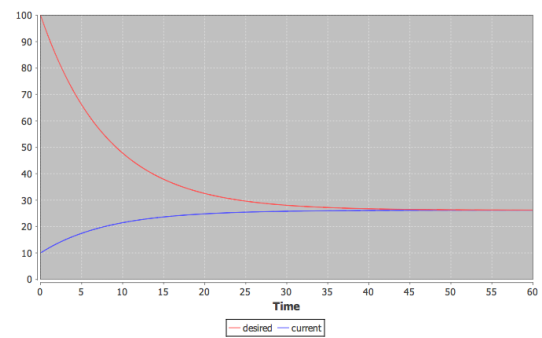
(a) time = 10 ; random time = false



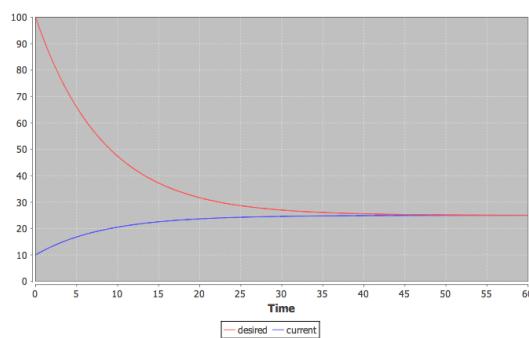
(b) time = 10 ; random time = true



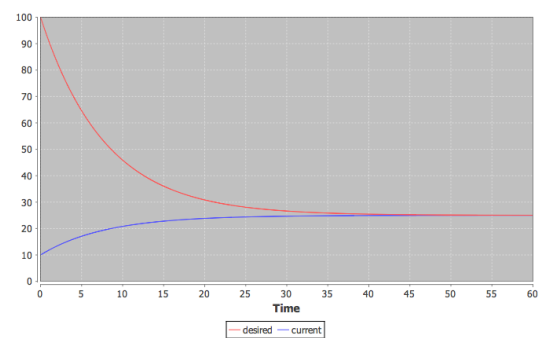
(c) time = 100 ; random time = false



(d) time = 100 ; random time = true

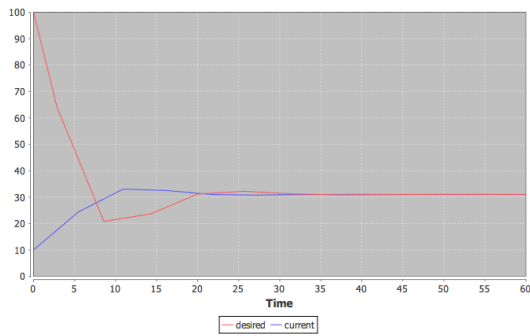


(e) time = 500 ; random time = false

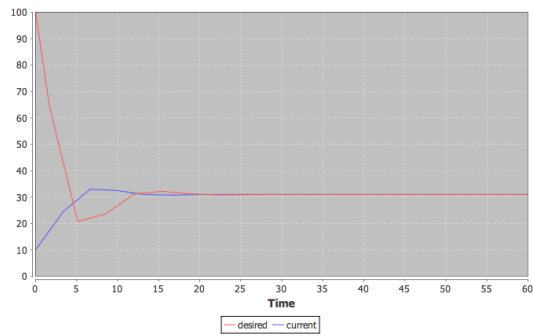


(f) time = 500 ; random time = true

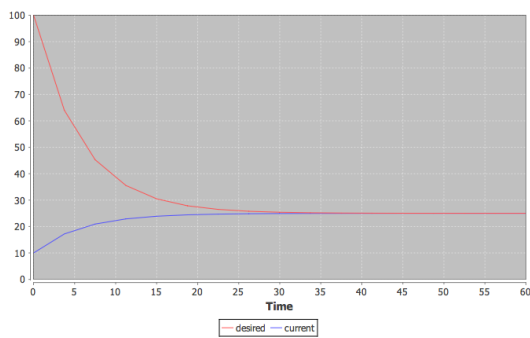
FIGURE 7: Simulator: the drifting goals model – asynchronous engine ($\delta = 0.1$)



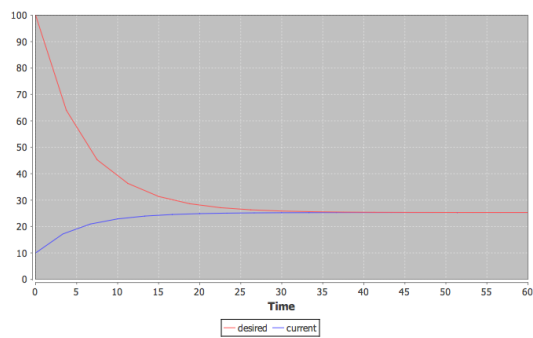
(a) time = 10 ; random time = false



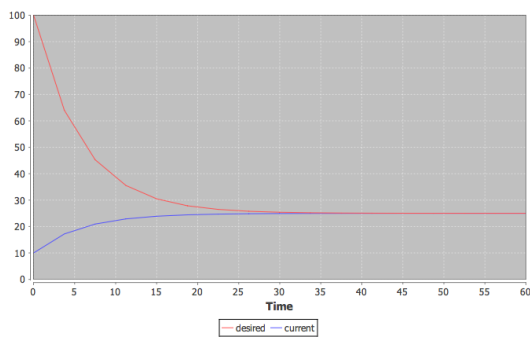
(b) time = 10 ; random time = true



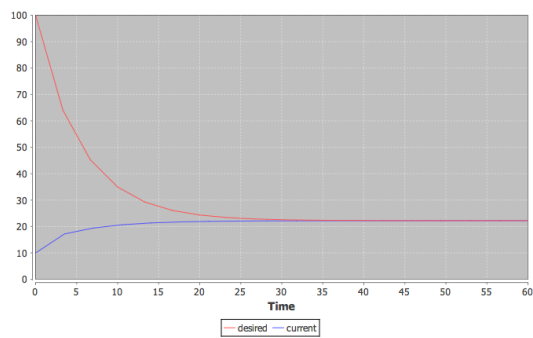
(c) time = 100 ; random time = false



(d) time = 100 ; random time = true



(e) time = 500 ; random time = false



(f) time = 500 ; random time = true

FIGURE 8: Simulator: the drifting goals model – asynchronous engine ($\delta = 4$)

Epidemic

```

1  functor
2  import
3      Agent at 'BaseAgent.ozf'
4
5  export
6      Removal
7      Infection
8      Susceptible
9      Contacts
10     Infected
11  define
12  class Removal from Agent.baseAgent
13      attr attribute
14      meth init(Name Value)
15          Agent.baseAgent,init(Name)
16          attribute := Value
17      end
18      meth m(In ?Out)
19          Out = @attribute * In.infected
20      end
21  end
22  class Infection from Agent.baseAgent
23      attr attribute
24      meth init(Name Value)
25          Agent.baseAgent,init(Name)
26          attribute := Value
27      end
28      meth m(In ?Out)
29          Out = @attribute * In.contacts
30      end
31  end
32  class Susceptible from Agent.baseAgent
33      attr popl incoming
34      meth init(Name Value In)
35          Agent.baseAgent,init(Name)
36          popl := Value
37          incoming := In
38          stock := true
39      end
40      meth m(In ?Out)
41          popl := @popl + (@incoming - In.infection) * @delta
42          Out = @popl
43      end
44  end
45  class Contacts from Agent.baseAgent
46      attr attribute
47      meth init(Name Value)
48          Agent.baseAgent,init(Name)
49          attribute := Value
50      end
51      meth m(In ?Out)
52          Out = @attribute * In.infected * In.susceptible
53      end
54  end
55  class Infected from Agent.baseAgent
56      attr attribute
57      meth init(Name Value)
58          Agent.baseAgent,init(Name)
59          attribute := Value
60          stock := true
61      end
62      meth m(In ?Out)
63          attribute := @attribute + (In.infection - In.removal) * @delta
64          Out = @attribute
65      end
66  end
67  end

```

LISTING 5: Epidemic: Agents.oz

```

1 functor
2 import
3     Agents at 'Agents.ozf'
4     Simulator at 'Simulator.ozf'
5
6 define
7 Graph
8 Removal
9 Infection
10 Susceptible
11 Contacts
12 Infected
13 in
14 Graph = loop(agents: [Infected#Agents.infected#init(infected 1.0)#1.0
15                     Susceptible#Agents.susceptible#init(susceptible 500.0 1.0)#500.0
16                     Infection#Agents.infection#init(infection 0.01)#5.0
17                     Removal#Agents.removal#init(removal 0.1)#0.1
18                     Contacts#Agents.contacts#init(contacts 0.1)#500.0]
19             graph: [Infected#[Infection Removal]#[Contacts Removal]
20                     Susceptible#[Infection]#[Contacts]
21                     Infection#[Contacts]#[Susceptible Infected]
22                     Removal#[Infected]#[Infected]
23                     Contacts#[Susceptible Infected]#[Infection]]
24             engine: async(500 true 0.1)
25             monitor: info(list: [susceptible infected] times: 100))
26
27 {Simulator.start Graph}
28 end

```

LISTING 6: Epidemic: Graph.oz

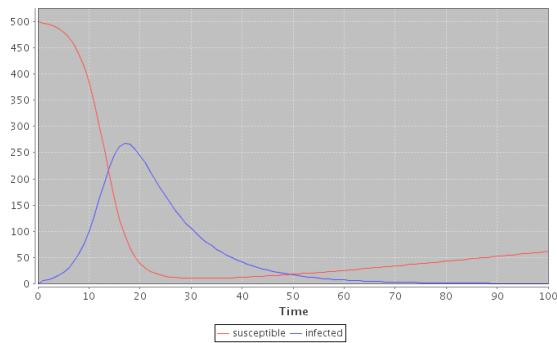
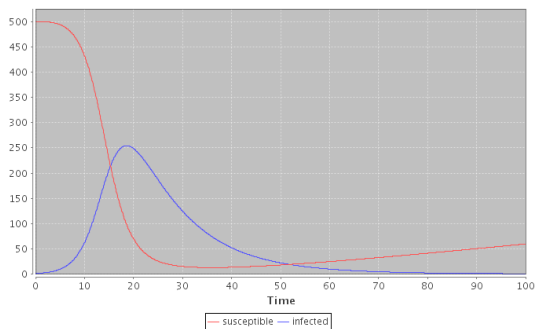
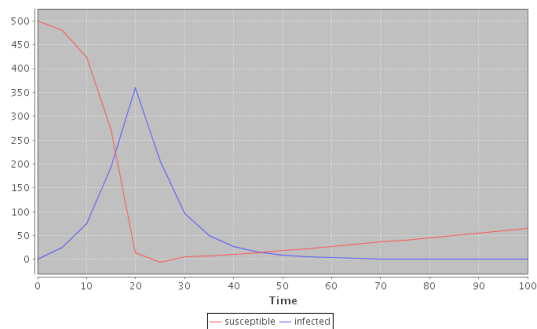
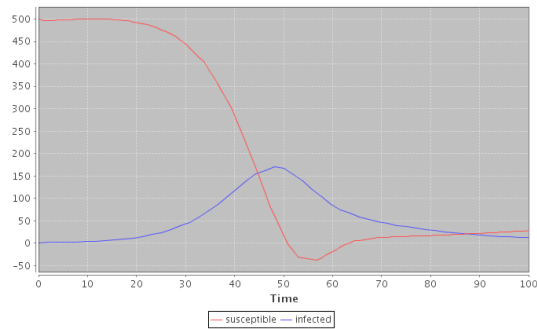
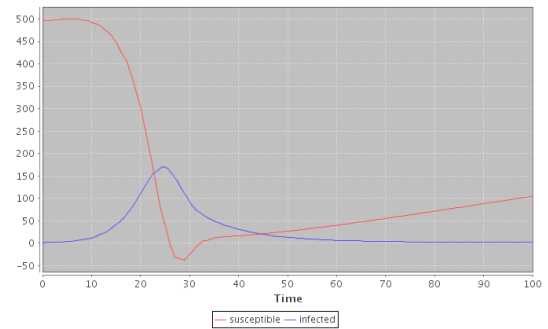
(a) $\delta = 1$ (b) $\delta = 0.1$ (c) $\delta = 5$

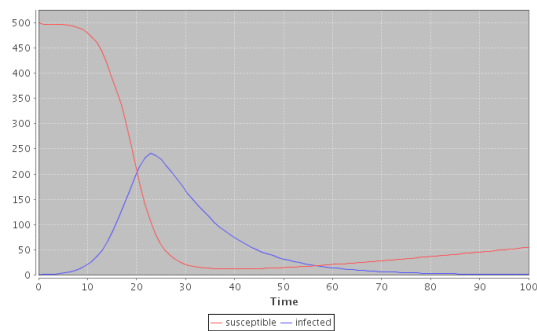
FIGURE 9: Simulator: the epidemic model – synchronous engine



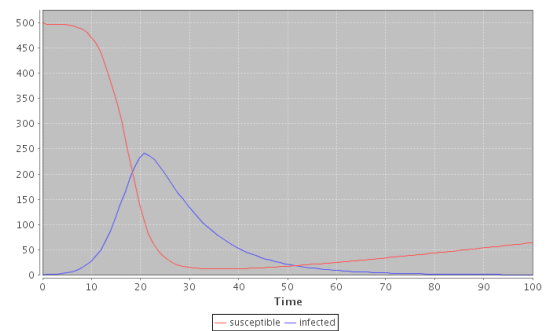
(a) time = 10 ; random time = false



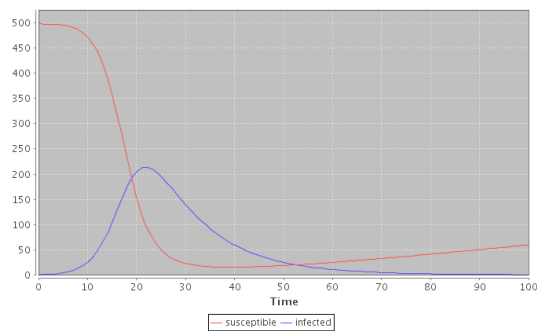
(b) time = 10 ; random time = true



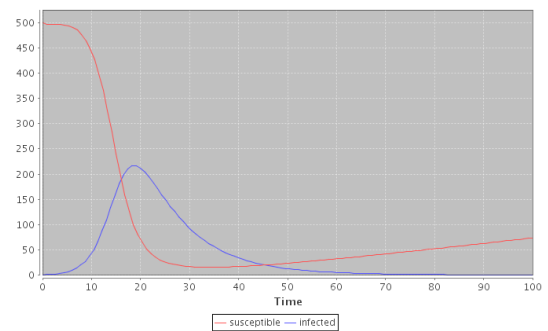
(c) time = 100 ; random time = false



(d) time = 100 ; random time = true

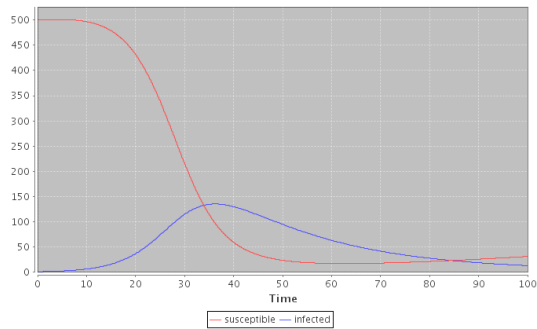


(e) time = 500 ; random time = false

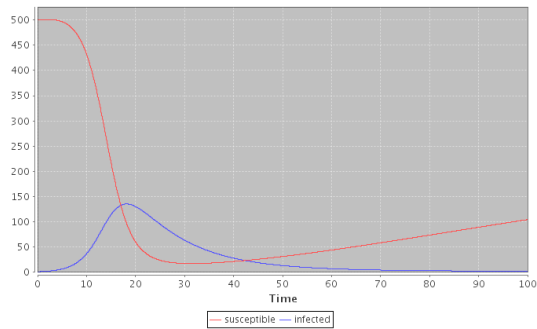


(f) time = 500 ; random time = true

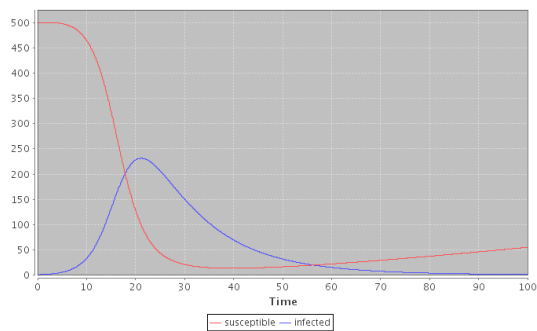
FIGURE 10: Simulator: the epidemic model – asynchronous engine ($\delta = 1$)



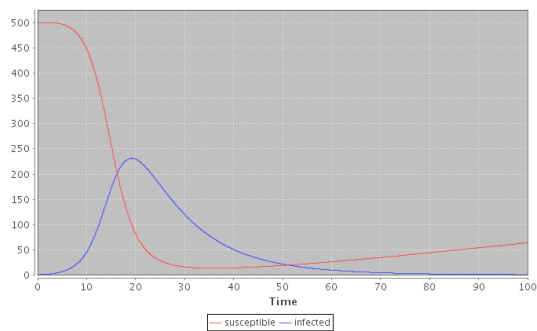
(a) time = 10 ; random time = false



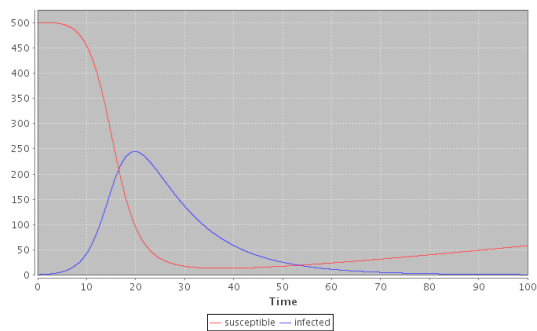
(b) time = 10 ; random time = true



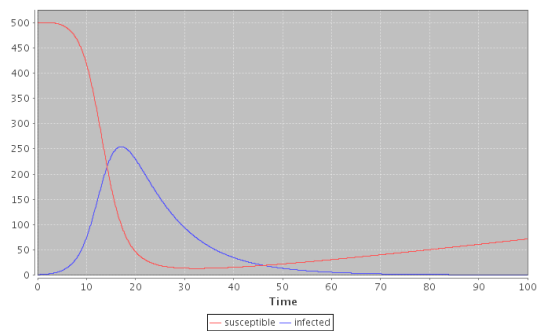
(c) time = 100 ; random time = false



(d) time = 100 ; random time = true

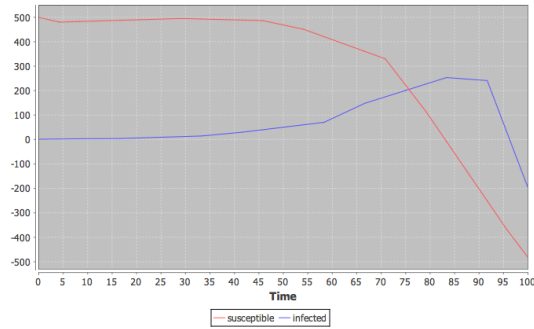


(e) time = 500 ; random time = false

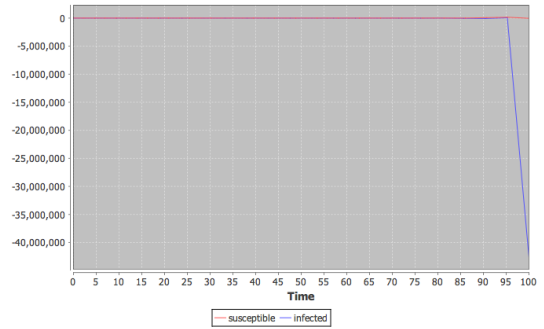


(f) time = 500 ; random time = true

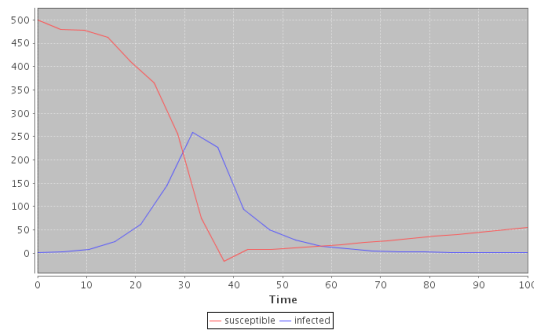
FIGURE 11: Simulator: the epidemic model – asynchronous engine ($\delta = 0.1$)



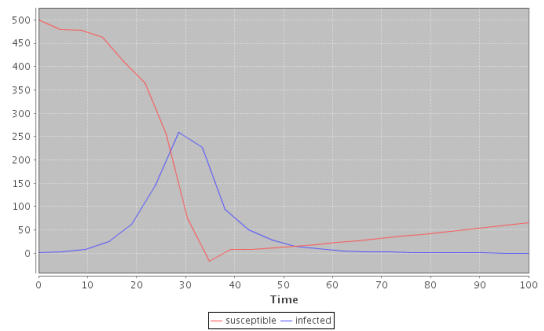
(a) time = 10 ; random time = false



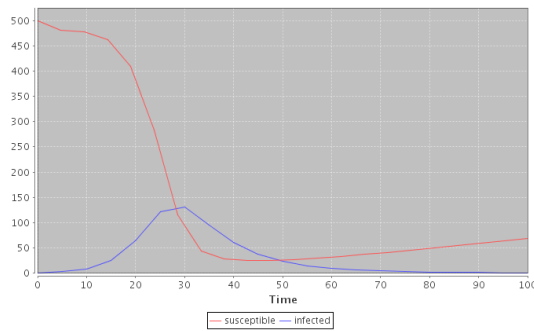
(b) time = 10 ; random time = true



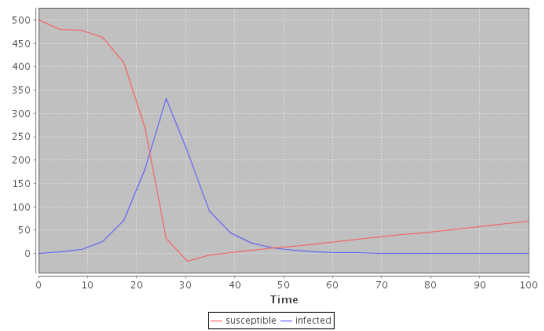
(c) time = 100 ; random time = false



(d) time = 100 ; random time = true



(e) time = 500 ; random time = false



(f) time = 500 ; random time = true

FIGURE 12: Simulator: the epidemic model – asynchronous engine ($\delta = 5$)

Crowding

```

1  functor
2  import
3      Agent at 'BaseAgent.ozf'
4
5  export
6      Crowding
7      Births
8      Deaths
9      Population
10     BirthFraction
11
12  define
13  class Crowding from Agent.baseAgent
14      attr capacity
15      meth init(Name Capacity)
16          Agent.baseAgent,init(Name)
17          capacity := Capacity
18      end
19      meth m(In ?Out)
20          Out = In.population / @capacity
21      end
22  end
23  class Births from Agent.baseAgent
24      meth m(In ?Out)
25          Out = In.birth_fraction * In.population
26      end
27  end
28  class Deaths from Agent.baseAgent
29      attr rate
30      meth init(Name Rate)
31          Agent.baseAgent,init(Name)
32          rate := Rate
33      end
34      meth m(In ?Out)
35          Out = @rate * In.population
36      end
37  end
38  class Population from Agent.baseAgent
39      attr popl
40      meth init(Name Popl)
41          Agent.baseAgent,init(Name)
42          popl := Popl
43          stock := true
44      end
45      meth m(In ?Out)
46          popl := @popl + (In.births - In.deaths) * @delta
47          Out = @popl
48      end
49  end
50  class BirthFraction from Agent.baseAgent
51      meth m(In ?Out)
52          Out = 0.08
53              - 0.000028561 * In.crowding
54              - 0.02 * {Pow In.crowding 2.0}
55      end
56  end
57  end

```

LISTING 7: Crowding: Agents.oz

```

1  functor
2  import
3      Agents at 'Agents.ozf'
4      Simulator at 'Simulator.ozf'
5
6  define
7  Graph
8  Crowding
9  Births
10 Deaths
11 Population
12 BirthFraction
13 in
14 Graph = loop(agents: [Population#Agents.population#init(population 10.0)#10.0
15                      Births#Agents.births#init(births)#0.5
16                      Crowding#Agents.crowding#init(crowding 200.0)#0.05
17                      BirthFraction#Agents.birthFraction#init(birth_fraction)#0.05
18                      Deaths#Agents.deaths#init(deaths 0.06)#0.6]
19      graph: [Population#[Births Deaths]#[Births Deaths Crowding]
20              Births#[BirthFraction Population]#[Population]
21              Crowding#[Population]#[BirthFraction]
22              BirthFraction#[Crowding]#[Births]
23              Deaths#[Population]#[Population]]
24      engine: async(100 false 0.25)
25      monitor: info(list: [population] times: 250))
26
27 {Simulator.start Graph}
28 end

```

LISTING 8: Crowding: Graph.oz

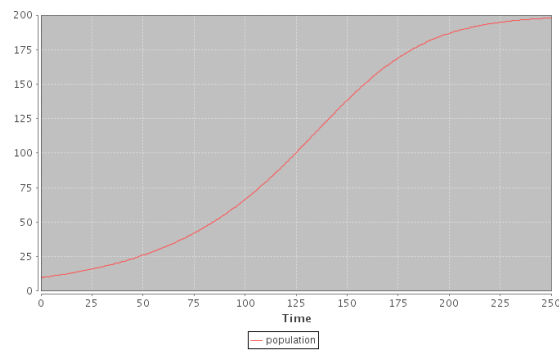
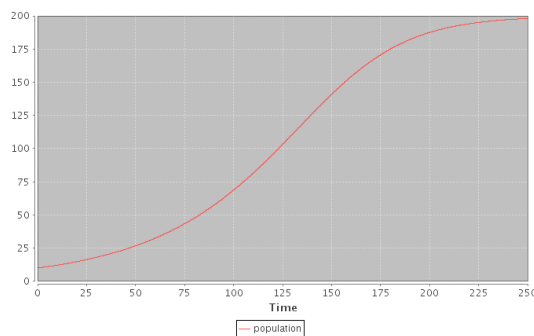
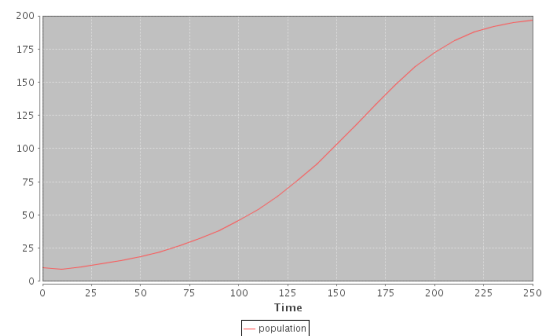
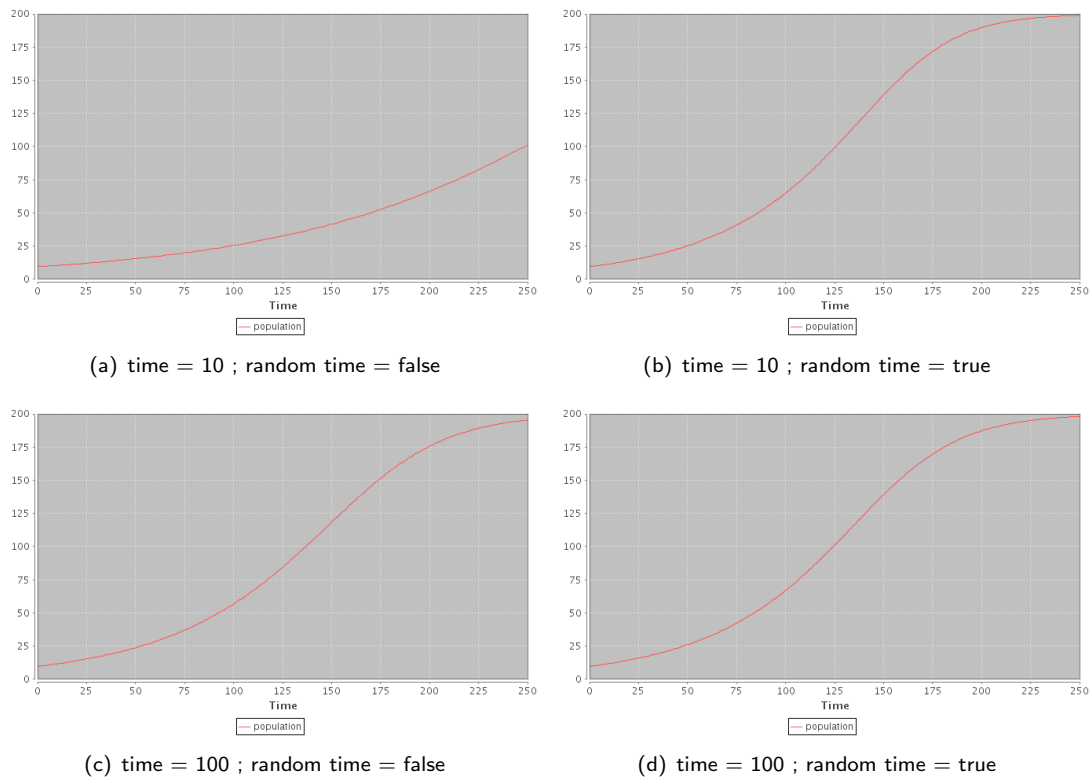
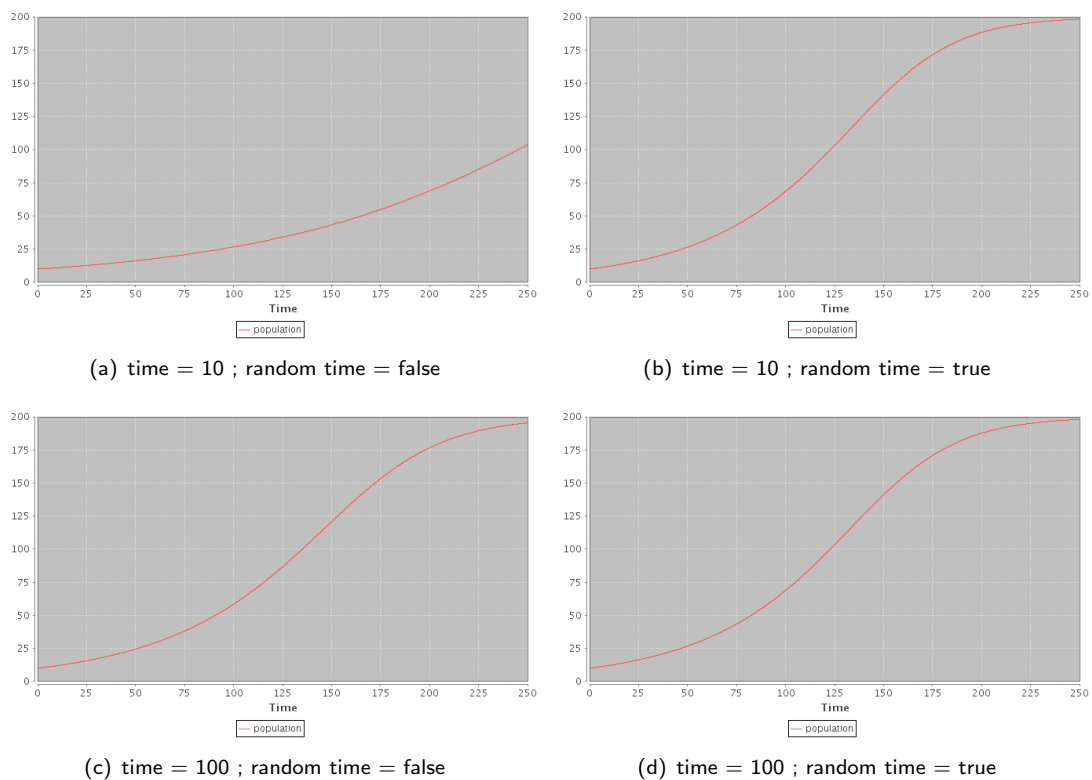
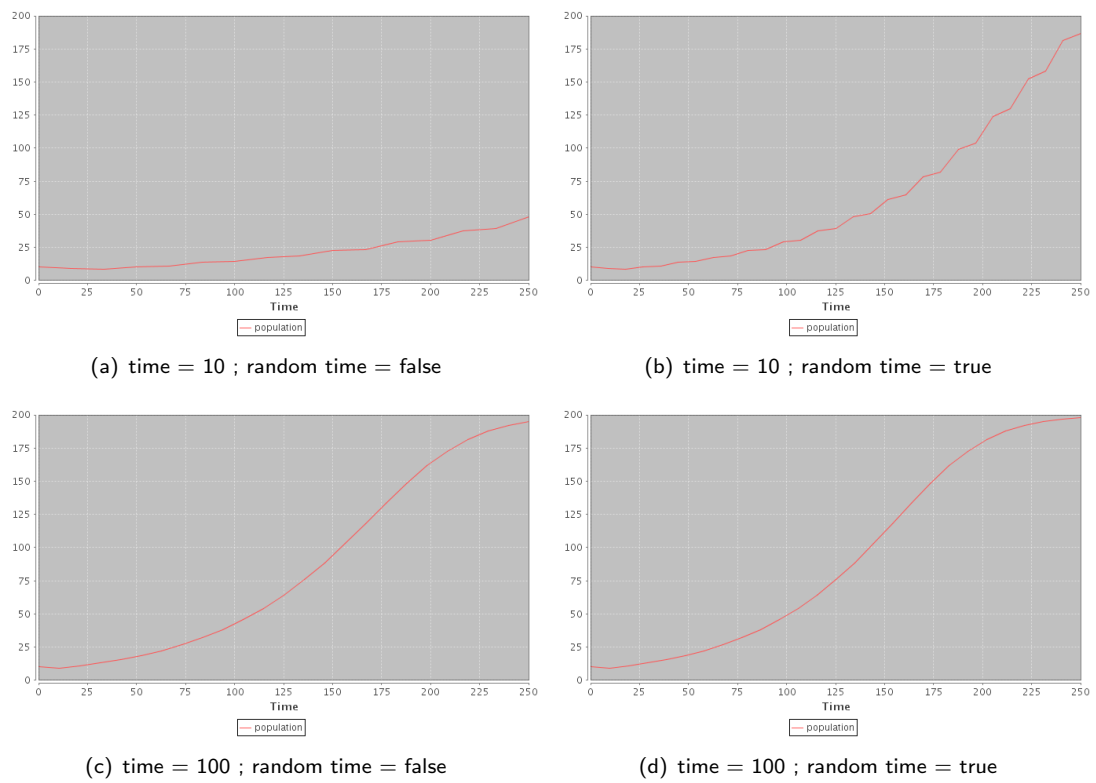
(a) $\delta = 1$ (b) $\delta = 0.25$ (c) $\delta = 10$

FIGURE 13: Simulator: the crowding model – synchronous engine

FIGURE 14: Simulator: the crowding model – asynchronous engine ($\delta = 1$)FIGURE 15: Simulator: the crowding model – asynchronous engine ($\delta = 0.25$)

FIGURE 16: Simulator: the crowding model – asynchronous engine ($\delta = 10$)

Flowers

```

1  functor
2  import
3      Agent at 'BaseAgent.ozf'
4
5  export
6      Area_flower
7      Decay
8      Multiplier
9      Growth
10     Growth_rate
11     Fraction_occupied
12
13  define
14  class Area_flower from Agent.baseAgent
15      attr area
16      meth init(Name Value)
17          Agent.baseAgent,init(Name)
18          area := Value
19          stock := true
20      end
21      meth m(In ?Out)
22          area := @area + (In.growth - In.decay) * @delta
23          Out = @area
24      end
25  end
26  class Decay from Agent.baseAgent
27      attr rate
28      meth init(Name Value)
29          Agent.baseAgent,init(Name)
30          rate := Value
31      end
32      meth m(In ?Out)
33          Out= In.area * @rate
34      end
35  end
36  class Multiplier from Agent.baseAgent
37      meth m(In ?Out)
38          Out = ~1.0*In.fraction_occupied + 1.0
39      end
40  end
41  class Growth from Agent.baseAgent
42      meth m(In ?Out)
43          Out = In.area* In.growth_rate
44      end
45  end
46  class Growth_rate from Agent.baseAgent
47      attr rate
48      meth init(Name Value)
49          Agent.baseAgent,init(Name)
50          rate := Value
51      end
52      meth m(In ?Out)
53          Out = @rate * In.multiplier
54      end
55  end
56  class Fraction_occupied from Agent.baseAgent
57      attr attribute
58      meth init(Name Value)
59          Agent.baseAgent,init(Name)
60          attribute := Value
61      end
62      meth m(In ?Out)
63          Out = In.area / @attribute
64      end
65  end
66  end

```

LISTING 9: Flowers: Agents.oz

```

1  functor
2  import
3      Agents at 'Agents.ozf'
4      Simulator at 'Simulator.ozf'
5
6  define
7  Graph
8  Area_flower
9  Decay
10 Multiplier
11 Growth
12 Growth_rate
13 Fraction_occupied
14 in
15 Graph = loop(agents: [Area_flower#Agents.area_flower#init(area 10.0)#10.0
16                      Growth#Agents.growth#init(growth)#9.9
17                      Decay#Agents.decay#init(decay 0.2)#2.0
18                      Growth_rate#Agents.growth_rate#init(growth_rate 1.0)#0.99
19                      Fraction_occupied#Agents.fraction_occupied#init(fraction_occupied 1000.0)#0.01
20                      Multiplier#Agents.multiplier#init(multiplier)#0.99]
21      graph: [Area_flower#[Growth Decay]#[Growth Decay Fraction_occupied]
22              Growth#[Growth_rate Area_flower]#[Area_flower]
23              Decay#[Area_flower]#[Area_flower]
24              Growth_rate#[Multiplier]#[Growth]
25              Fraction_occupied#[Area_flower]#[Multiplier]
26              Multiplier#[Fraction_occupied]#[Growth_rate]]
27      engine: async(10 true 2.0)
28      monitor: info(list: [area] times: 20))
29
30      {Simulator.start Graph}
31 end

```

LISTING 10: Flowers: Graph.oz

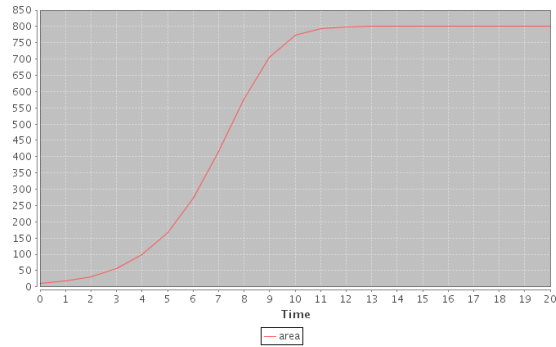
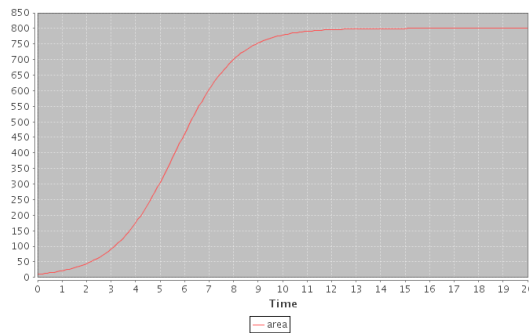
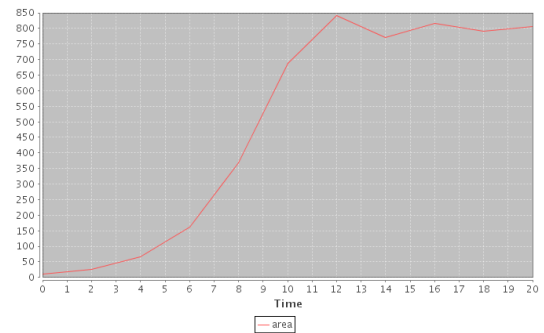
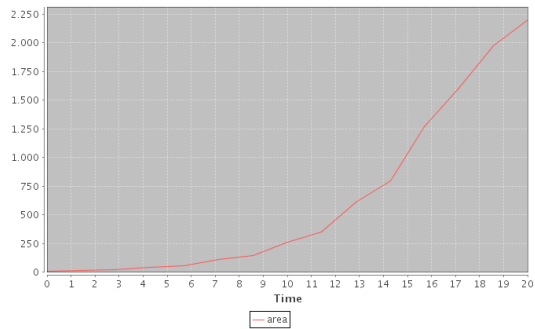
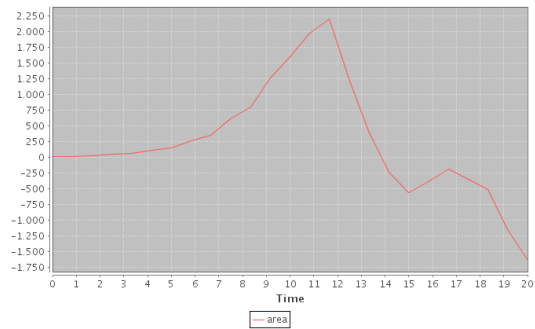
(a) $\delta = 1$ (b) $\delta = 0.1$ (c) $\delta = 2$

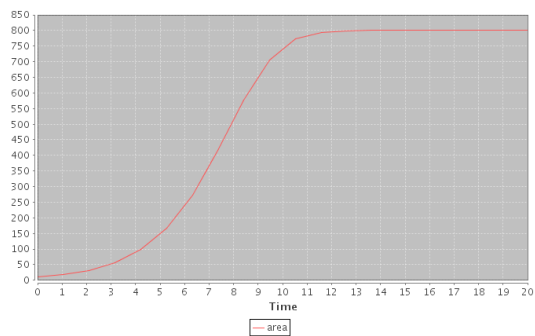
FIGURE 17: Simulator: the flowers model – synchronous engine



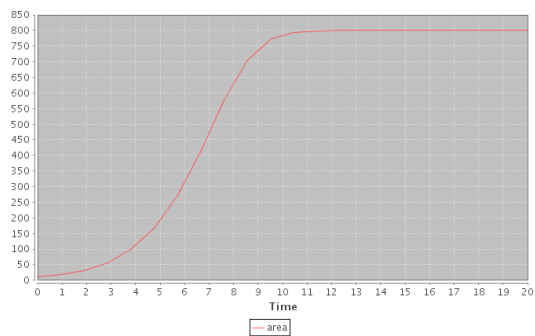
(a) time = 10 ; random time = false



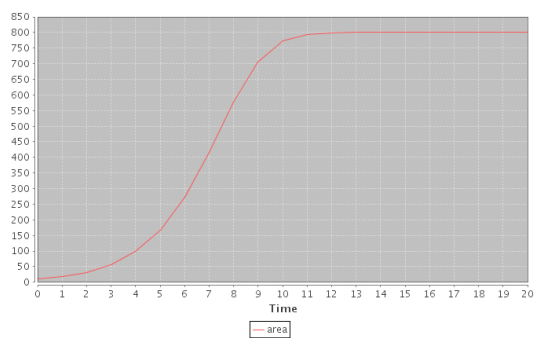
(b) time = 10 ; random time = true



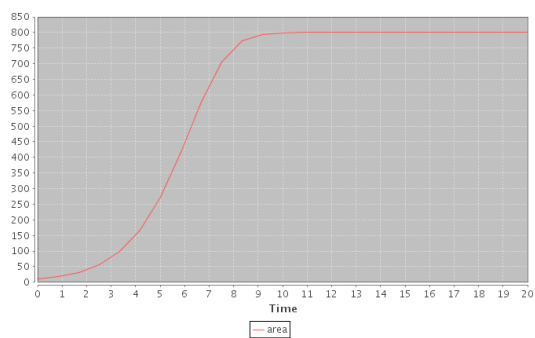
(c) time = 100 ; random time = false



(d) time = 100 ; random time = true

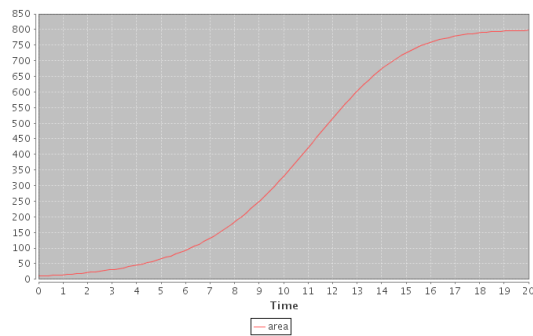


(e) time = 500 ; random time = false

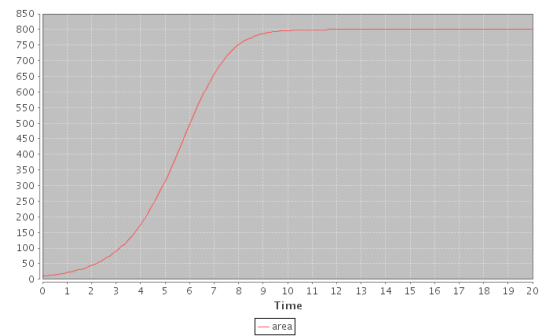


(f) time = 500 ; random time = true

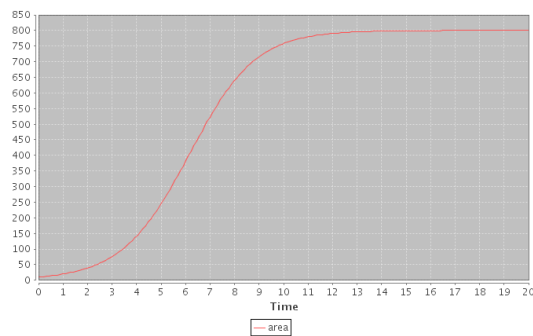
FIGURE 18: Simulator: the flowers model – asynchronous engine ($\delta = 1$)



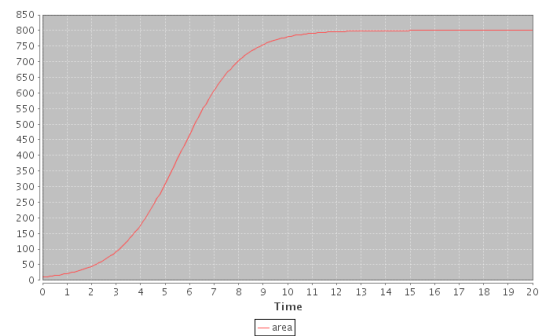
(a) time = 10 ; random time = false



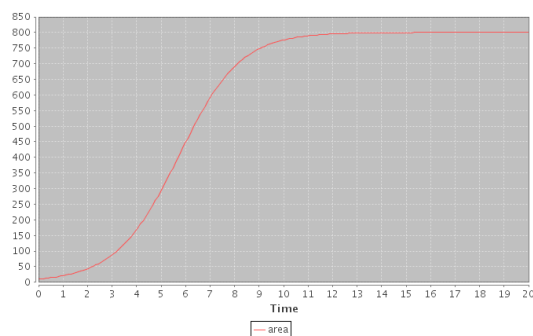
(b) time = 10 ; random time = true



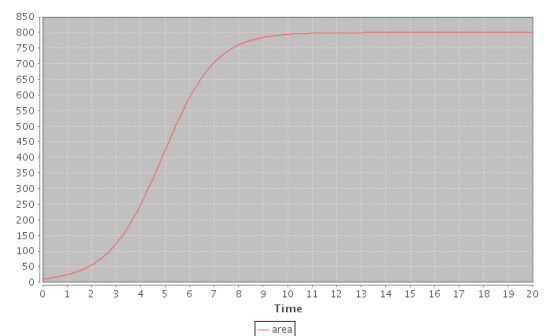
(c) time = 100 ; random time = false



(d) time = 100 ; random time = true

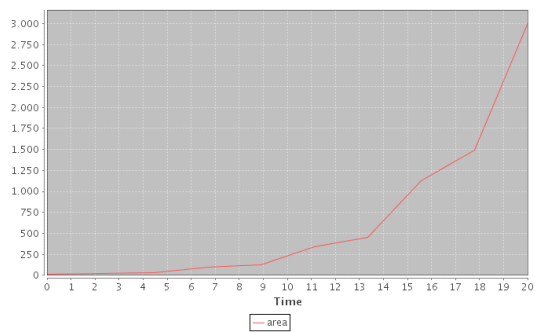


(e) time = 500 ; random time = false

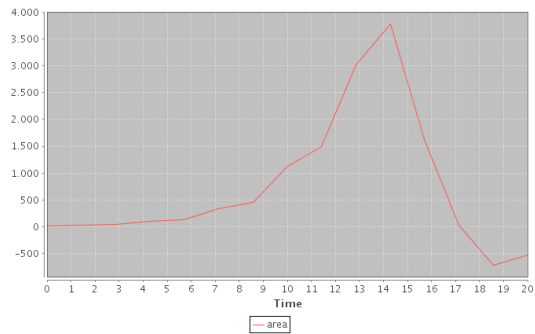


(f) time = 500 ; random time = true

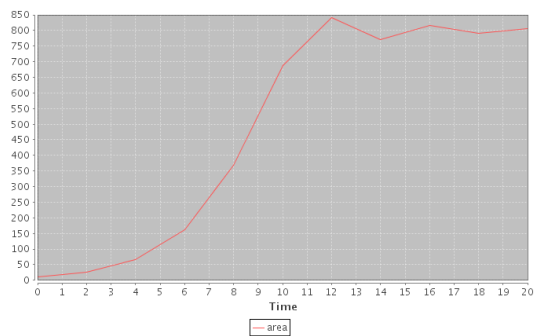
FIGURE 19: Simulator: the flowers model – asynchronous engine ($\delta = 0.1$)



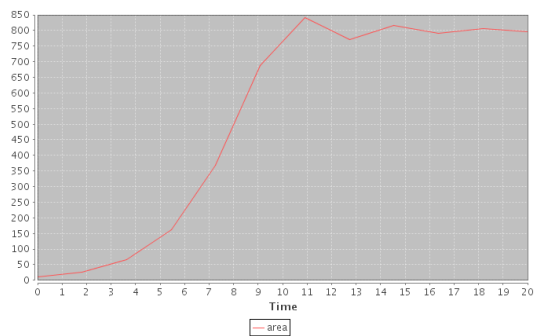
(a) time = 10 ; random time = false



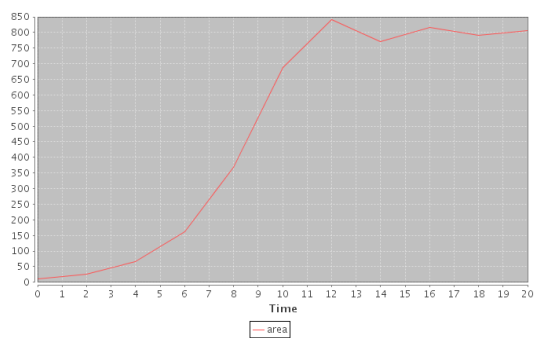
(b) time = 10 ; random time = true



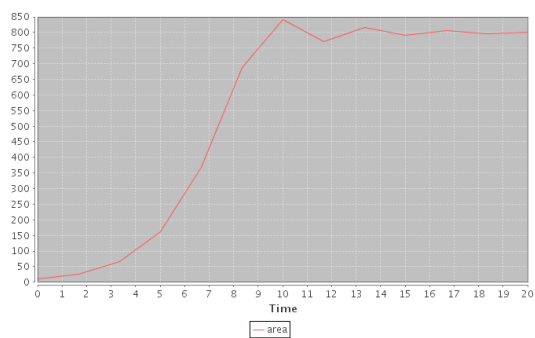
(c) time = 100 ; random time = false



(d) time = 100 ; random time = true



(e) time = 500 ; random time = false



(f) time = 500 ; random time = true

FIGURE 20: Simulator: the flowers model – asynchronous engine (delta = 2)

Balancing loop

```
1  functor
2  import
3      Agent at 'BaseAgent.ozf'
4
5  export
6      Action
7      Gap
8      Current_value
9
10 define
11 class Action from Agent.baseAgent
12     attr attribute
13     meth init(Name Value)
14         Agent.baseAgent,init(Name)
15         attribute := Value
16     end
17     meth m(In ?Out)
18         Out= In.gap / @attribute
19     end
20 end
21 class Gap from Agent.baseAgent
22     attr attribute dt
23     meth init(Name Value)
24         Agent.baseAgent,init(Name)
25         attribute := Value
26     end
27     meth m(In ?Out)
28         Out = @attribute - In.current_value
29     end
30 end
31 class Current_value from Agent.baseAgent
32     attr attribute
33     meth init(Name Value)
34         Agent.baseAgent,init(Name)
35         attribute := Value
36         stock := true
37     end
38     meth m(In ?Out)
39         attribute := @attribute + In.action * @delta
40         Out = @attribute
41     end
42 end
43 end
```

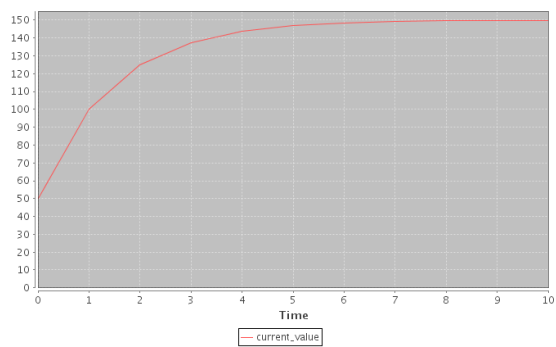
LISTING 11: Balancing loop: Agents.oz

```

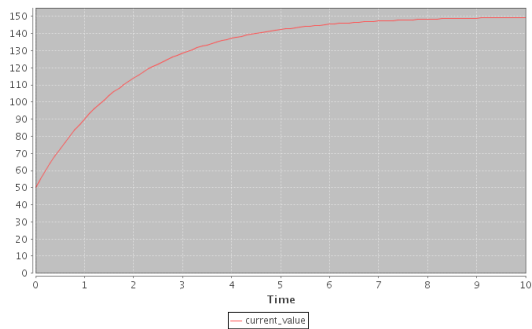
1  functor
2  import
3      Agents at 'Agents.ozf'
4      Simulator at 'Simulator.ozf'
5
6  define
7  Graph
8  Action
9  Gap
10 Current_value
11 in
12 Graph = loop(agents: [Current_value#Agents.current_value#init(current_value 50.0)#50.0
13                      Gap#Agents.gap#init(gap 150.0)#100.0
14                      Action#Agents.action#init(action 2.0)#50.0]
15              graph: [Current_value#[Action]#[Gap]
16                     Gap#[Current_value]#[Action]
17                     Action#[Gap]#[Current_value]]
18              engine: async(500 true 2.0)
19              monitor: info(list: [current_value] times: 10))
20
21      {Simulator.start Graph}
22 end

```

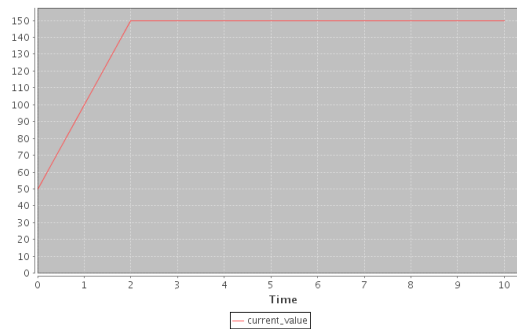
LISTING 12: Balancing loop: Graph.oz



(a) delta = 1

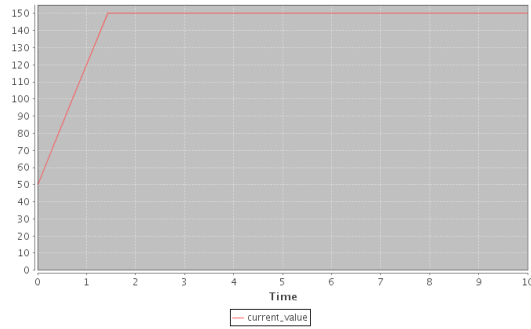


(b) delta = 0.1

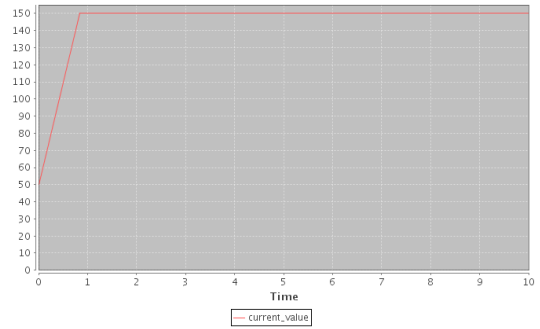


(c) delta = 2

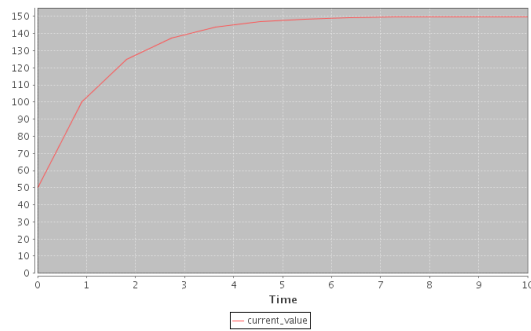
FIGURE 21: Simulator: the balancing loop model – synchronous engine



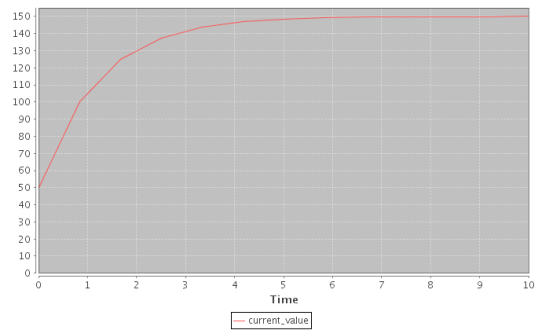
(a) time = 10 ; random time = false



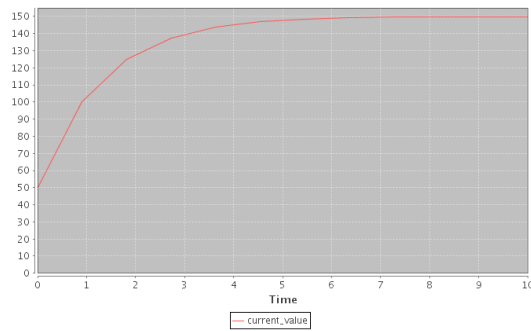
(b) time = 10 ; random time = true



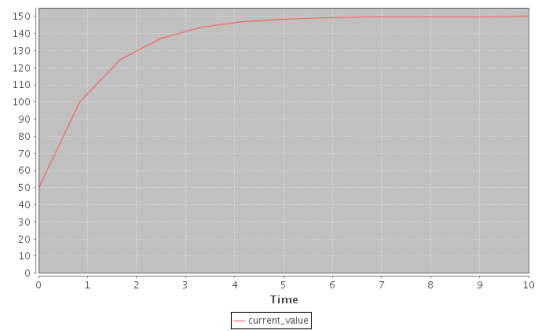
(c) time = 100 ; random time = false



(d) time = 100 ; random time = true

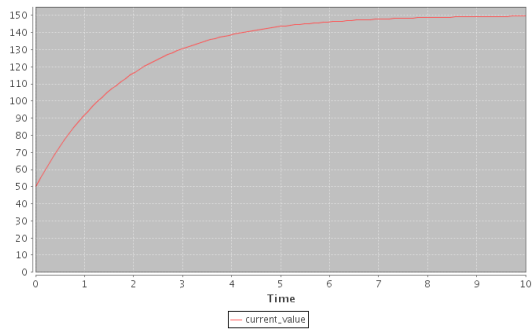


(e) time = 500 ; random time = false

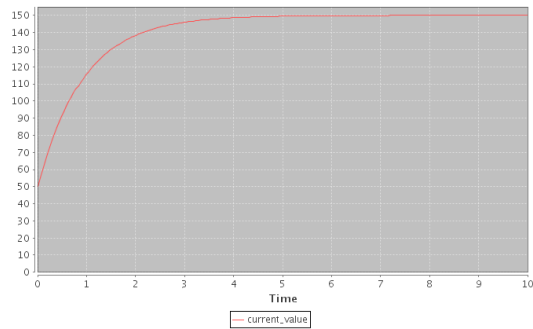


(f) time = 500 ; random time = true

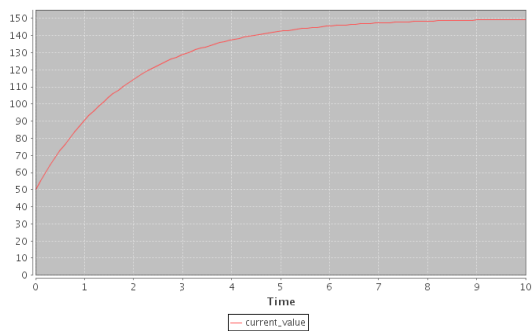
FIGURE 22: Simulator: the balancing loop model – asynchronous engine ($\delta = 1$)



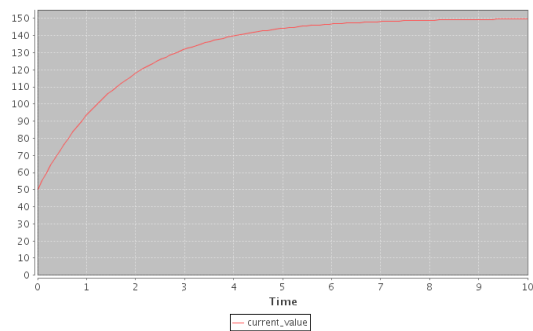
(a) time = 10 ; random time = false



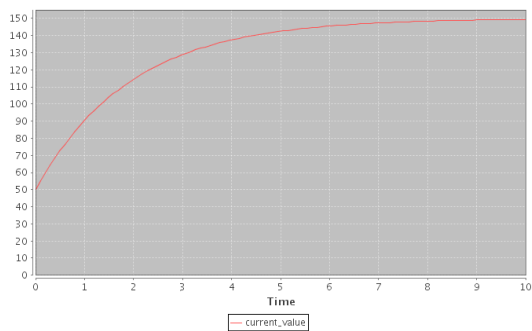
(b) time = 10 ; random time = true



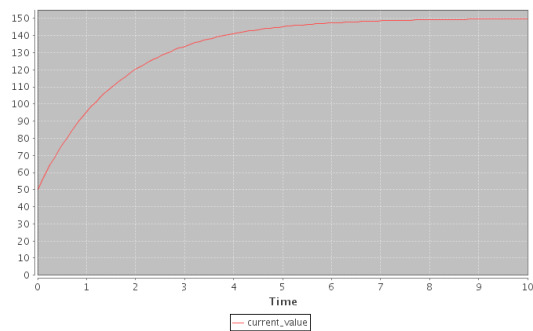
(c) time = 100 ; random time = false



(d) time = 100 ; random time = true

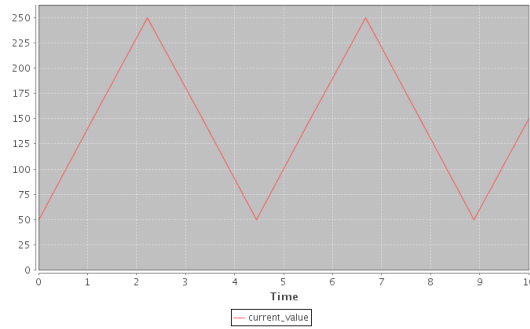


(e) time = 500 ; random time = false

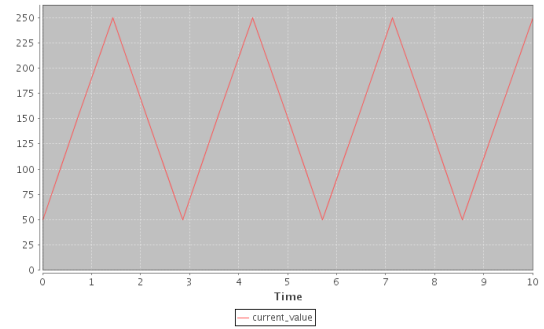


(f) time = 500 ; random time = true

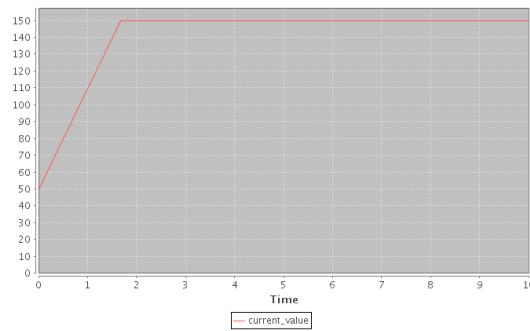
FIGURE 23: Simulator: the balancing loop model – asynchronous engine ($\delta = 0.1$)



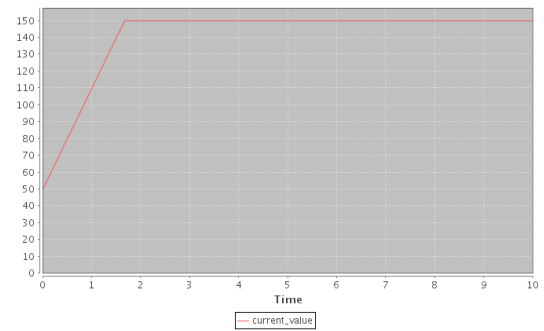
(a) time = 10 ; random time = false



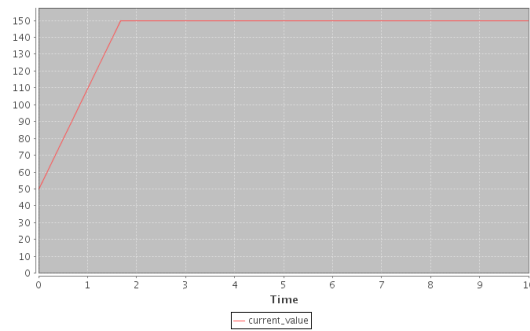
(b) time = 10 ; random time = true



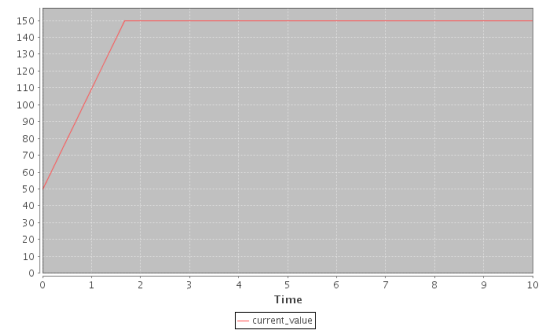
(c) time = 100 ; random time = false



(d) time = 100 ; random time = true



(e) time = 500 ; random time = false



(f) time = 500 ; random time = true

FIGURE 24: Simulator: the balancing loop model – asynchronous engine ($\delta = 2$)

Knowledge diffusion

```

1  functor
2  import
3      Agent at 'BaseAgent.ozf'
4
5  export
6      Gap
7      Diffusion
8      No_knowledge
9      Knowledge
10
11  define
12  class Gap from Agent.baseAgent
13      meth m(In ?Out)
14          Out = (In.knowledge + In.no_knowledge)
15                - In.knowledge
16      end
17  end
18  class Diffusion from Agent.baseAgent
19      attr attribute
20      meth init(Name Value)
21          Agent.baseAgent,init(Name)
22          attribute := Value
23      end
24      meth m(In ?Out)
25          Out = @attribute * In.knowledge * In.gap
26      end
27  end
28  class No_knowledge from Agent.baseAgent
29      attr attribute
30      meth init(Name Value)
31          Agent.baseAgent,init(Name)
32          stock := true
33          attribute := Value
34      end
35      meth m(In ?Out)
36          attribute := @attribute + (~In.diffusion) * @delta
37          Out = @attribute
38      end
39  end
40  class Knowledge from Agent.baseAgent
41      attr attribute dt
42      meth init(Name Value)
43          Agent.baseAgent,init(Name)
44          stock := true
45          attribute := Value
46      end
47      meth m(In ?Out)
48          attribute := @attribute + In.diffusion * @delta
49          Out = @attribute
50      end
51  end
52  end

```

LISTING 13: Knowledge diffusion: Agents.oz

```

1  functor
2  import
3      Agents at 'Agents.ozf'
4      Simulator at 'Simulator.ozf'
5
6  define
7  Graph
8  Gap
9  Diffusion
10 No_knowledge
11 Knowledge
12 in
13 Graph = loop(agents: [No_knowledge#Agents.no_knowledge#init(no_knowledge 100.0)#100.0
14                      Knowledge#Agents.knowledge#init(knowledge 1.0)#1.0
15                      Diffusion#Agents.diffusion#init(diffusion 0.02)#1.98
16                      Gap#Agents.gap#init(gap)#100.0]
17             graph: [No_knowledge#[Diffusion]#[Gap]
18                    Knowledge#[Diffusion]#[Gap Diffusion]
19                    Diffusion#[Knowledge Gap]#[No_knowledge Knowledge]
20                    Gap#[Knowledge No_knowledge]#[Diffusion]]
21             engine: async(100 true 0.6)
22             monitor: info(list: [knowledge no_knowledge] times: 10))
23
24     {Simulator.start Graph}
25 end

```

LISTING 14: Knowledge diffusion: Graph.oz

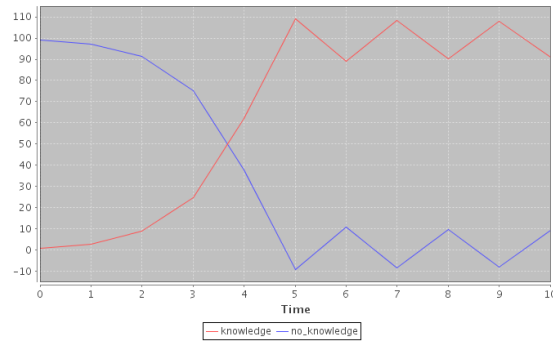
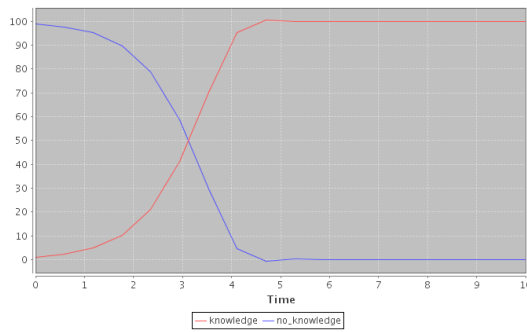
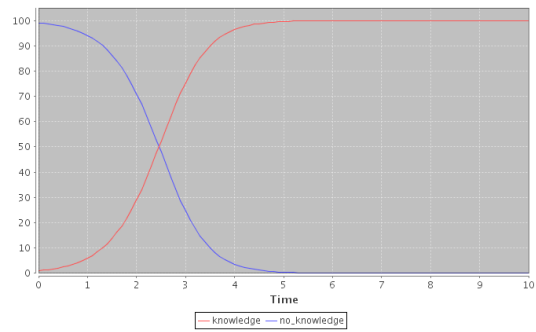
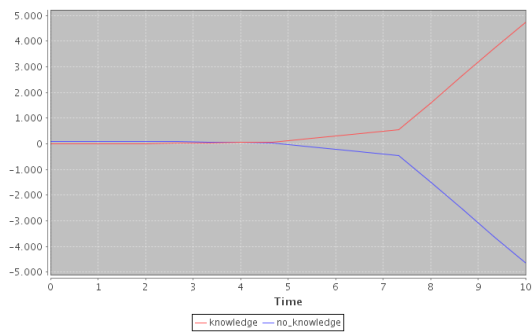
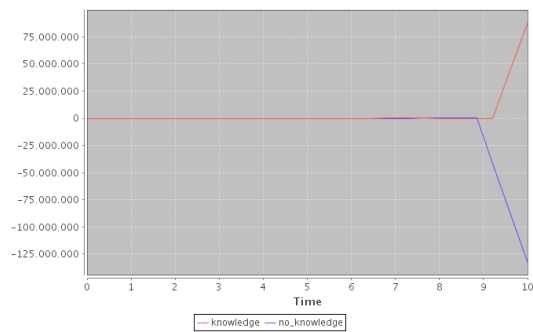
(a) $\delta = 1$ (b) $\delta = 0.6$ (c) $\delta = 0.1$

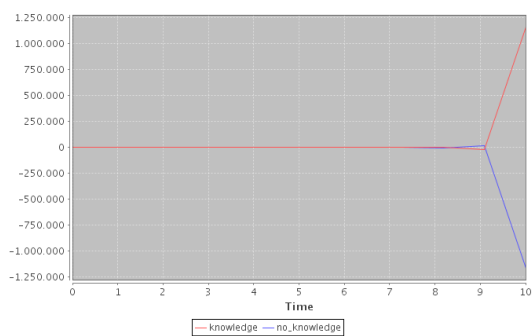
FIGURE 25: Simulator: the knowledge diffusion model – synchronous engine



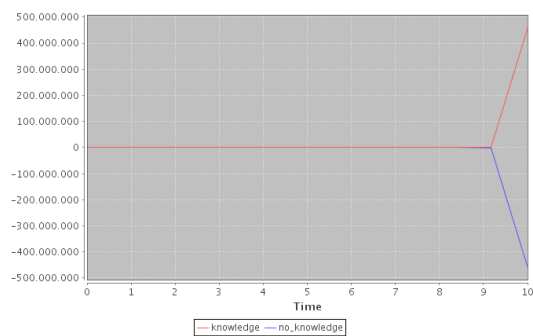
(a) time = 10 ; random time = false



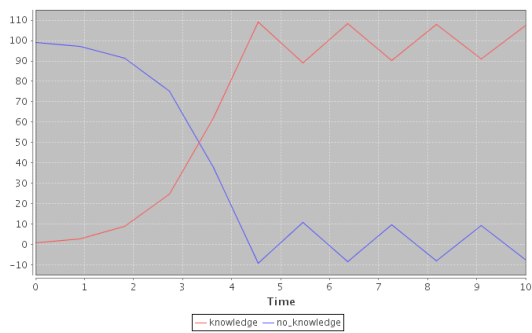
(b) time = 10 ; random time = true



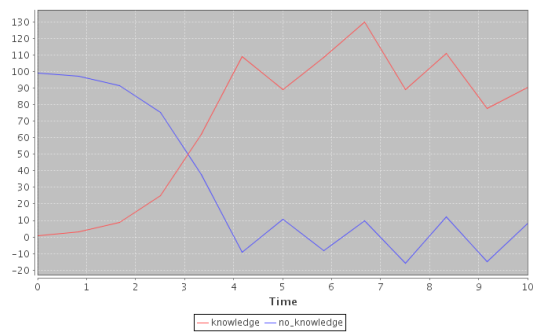
(c) time = 100 ; random time = false



(d) time = 100 ; random time = true

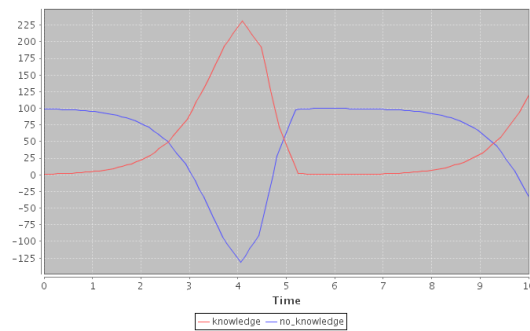


(e) time = 500 ; random time = false

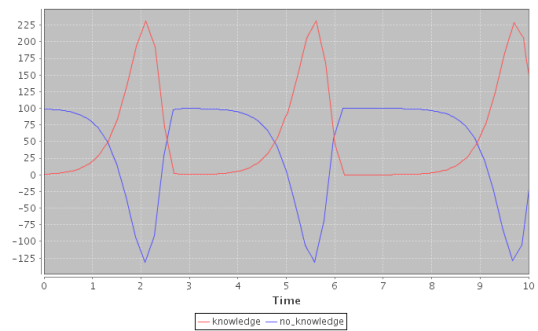


(f) time = 500 ; random time = true

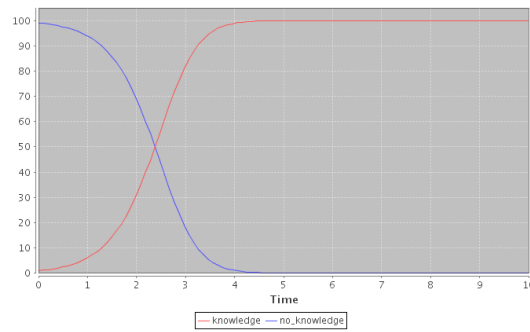
FIGURE 26: Simulator: the knowledge diffusion model – asynchronous engine ($\delta = 1$)



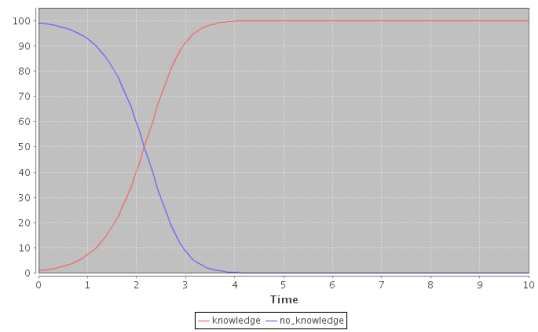
(a) time = 10 ; random time = false



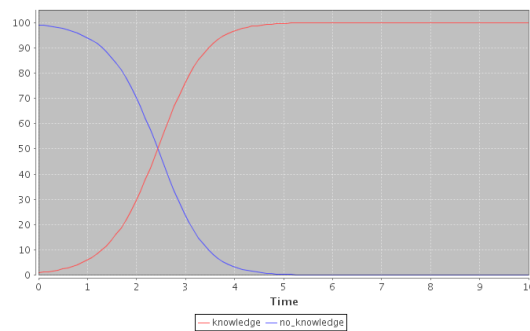
(b) time = 10 ; random time = true



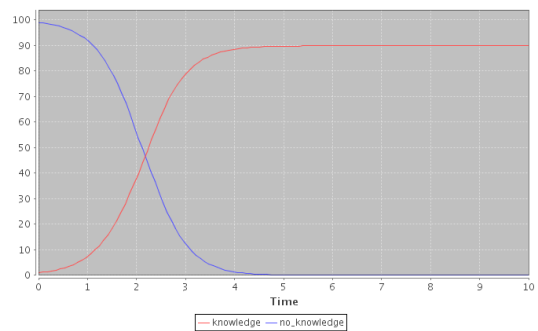
(c) time = 100 ; random time = false



(d) time = 100 ; random time = true

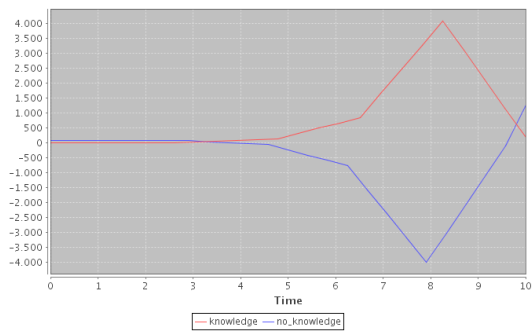


(e) time = 500 ; random time = false

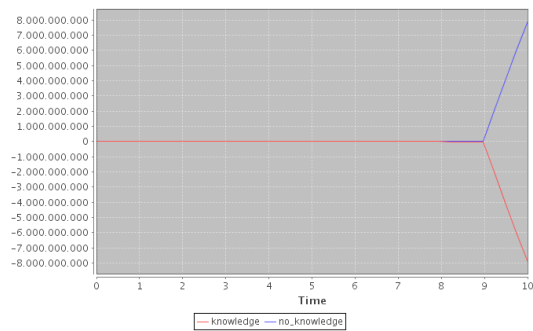


(f) time = 500 ; random time = true

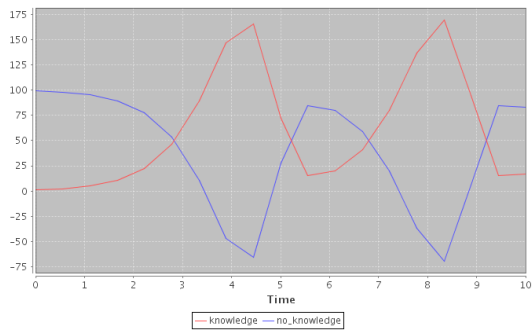
FIGURE 27: Simulator: the knowledge diffusion model – asynchronous engine ($\delta = 0.1$)



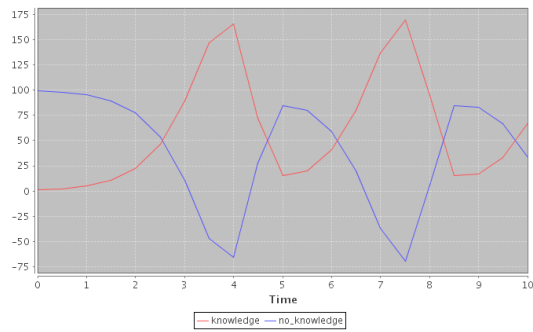
(a) time = 10 ; random time = false



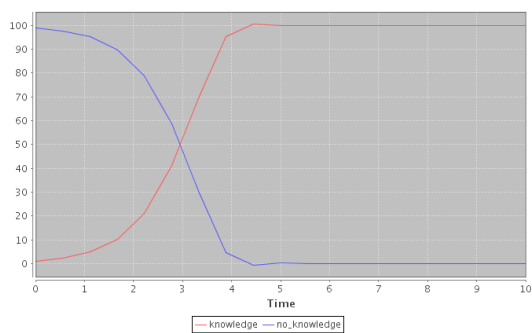
(b) time = 10 ; random time = true



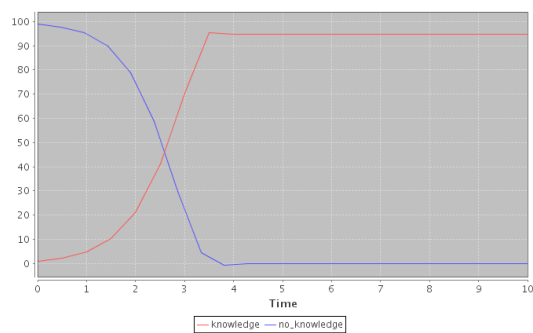
(c) time = 100 ; random time = false



(d) time = 100 ; random time = true



(e) time = 500 ; random time = false



(f) time = 500 ; random time = true

FIGURE 28: Simulator: the knowledge diffusion model – asynchronous engine ($\delta = 0.6$)

Escalation

```

1  functor
2  import
3      Agent at 'BaseAgent.ozf'
4
5  export
6      Activity_b
7      Activity_a
8      Results_b
9      Results_a
10     Results_a_rel_b
11
12  define
13  class Activity_b from Agent.baseAgent
14      meth m(In ?Out)
15          Out = In.results_a_rel_b
16      end
17  end
18  class Activity_a from Agent.baseAgent
19      attr attribute
20      meth init(Name Value)
21          Agent.baseAgent,init(Name)
22          attribute := Value
23      end
24      meth m(In ?Out)
25          Out = In.results_a_rel_b + @attribute
26      end
27  end
28  class Results_b from Agent.baseAgent
29      attr attribute
30      meth init(Name Value)
31          Agent.baseAgent,init(Name)
32          attribute := Value
33          stock := true
34      end
35      meth m(In ?Out)
36          attribute := @attribute + In.activity_b * @delta
37          Out = @attribute
38      end
39  end
40  class Results_a from Agent.baseAgent
41      attr attribute
42      meth init(Name Value)
43          Agent.baseAgent,init(Name)
44          attribute := Value
45          stock := true
46      end
47      meth m(In ?Out)
48          attribute := @attribute + In.activity_a * @delta
49          Out = @attribute
50      end
51  end
52  class Results_a_rel_b from Agent.baseAgent
53      meth m(In ?Out)
54          Out = In.results_a - In.results_b
55      end
56  end
57  end

```

LISTING 15: Escalation: Agents.oz

```

1 functor
2 import
3     Agents at 'Agents.ozf'
4     Simulator at 'Simulator.ozf'
5
6 define
7 Graph
8 Activity_b
9 Activity_a
10 Results_b
11 Results_a
12 Results_a_rel_b
13 in
14 Graph = loop(agents: [Results_a#Agents.results_a#init(results_a 101.0)#101.0
15                     Results_b#Agents.results_b#init(results_b 100.0)#100.0
16                     Activity_a#Agents.activity_a#init(activity_a 1.0)#1.0
17                     Activity_b#Agents.activity_b#init(activity_b)#0.0
18                     Results_a_rel_b#Agents.results_a_rel_b#init(results_a_rel_b)#1.0]
19             graph: [Results_a#[Activity_a]#[Results_a_rel_b]
20                    Results_b#[Activity_b]#[Results_a_rel_b]
21                    Activity_a#[Results_a_rel_b]#[Results_a]
22                    Activity_b#[Results_a_rel_b]#[Results_b]
23                    Results_a_rel_b#[Results_a Results_b]#[Activity_a Activity_b]]
24             engine: async(200 true 1.0)
25             monitor: info(list: [results_a results_b] times: 100))
26
27     {Simulator.start Graph}
28 end

```

LISTING 16: Escalation: Graph.oz

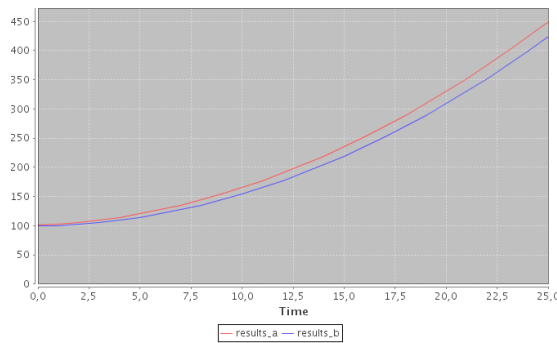
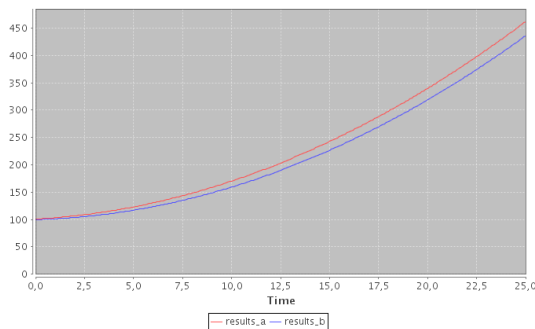
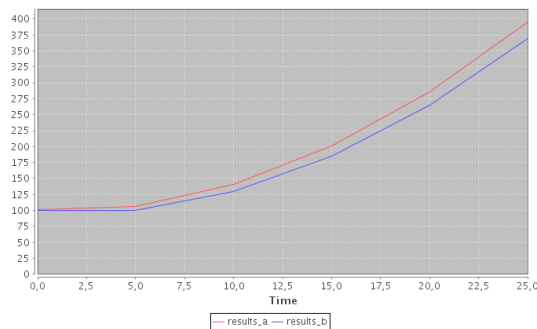
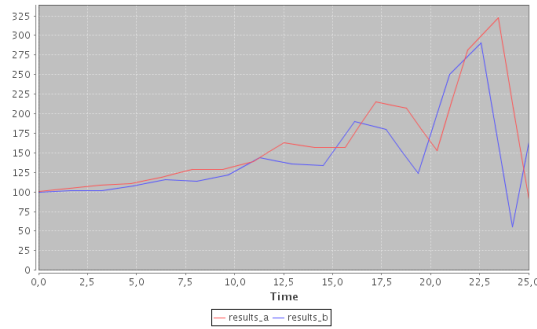
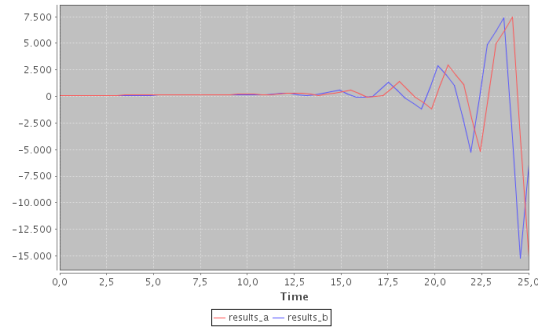
(a) $\delta = 1$ (b) $\delta = 0.1$ (c) $\delta = 5$

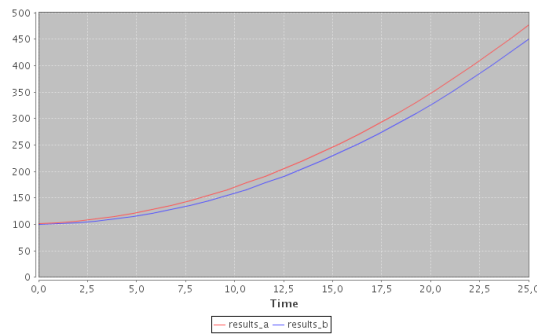
FIGURE 29: Simulator: the escalation model – synchronous engine



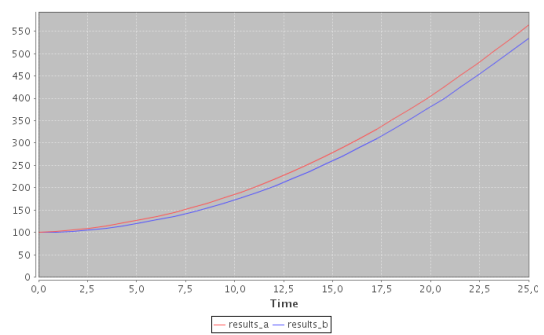
(a) time = 10 ; random time = false



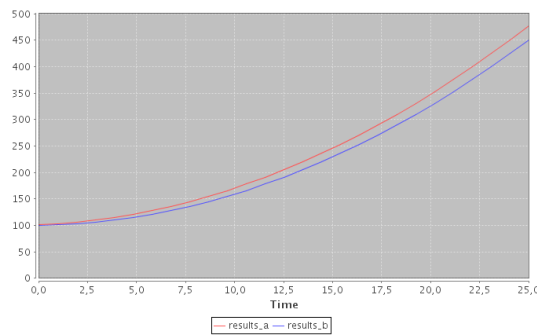
(b) time = 10 ; random time = true



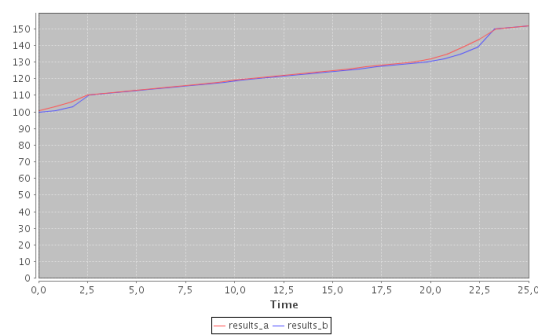
(c) time = 100 ; random time = false



(d) time = 100 ; random time = true

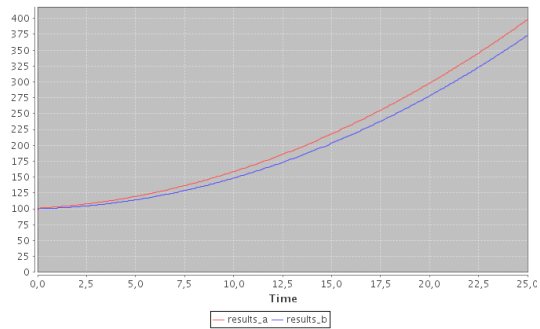


(e) time = 500 ; random time = false

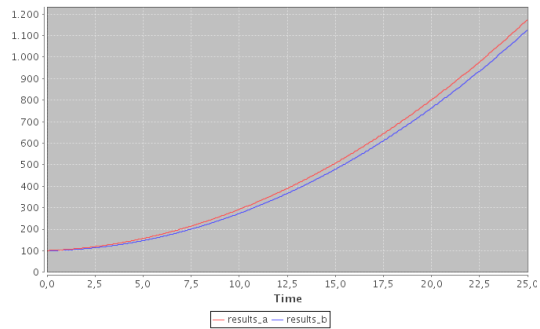


(f) time = 500 ; random time = true

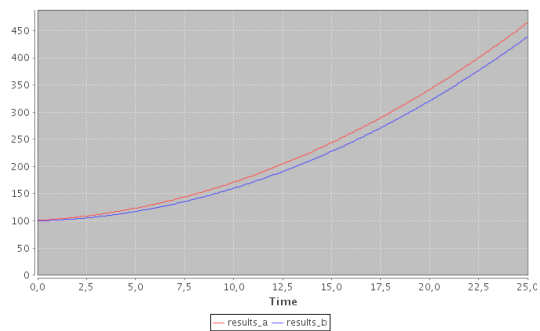
FIGURE 30: Simulator: the escalation model – asynchronous engine ($\delta = 1$)



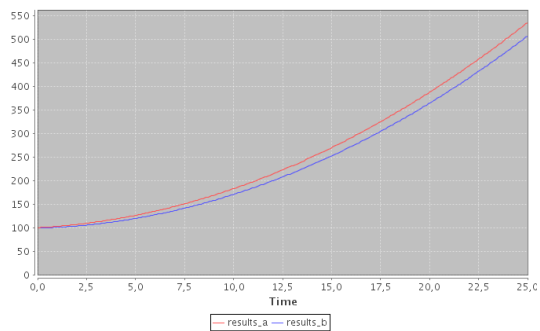
(a) time = 10 ; random time = false



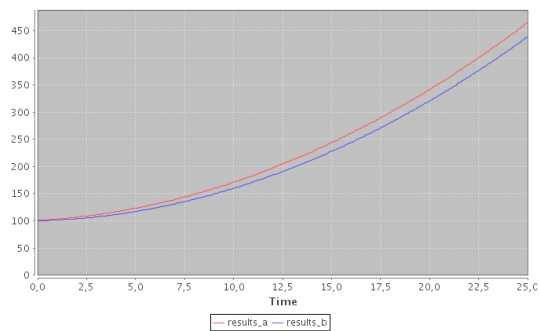
(b) time = 10 ; random time = true



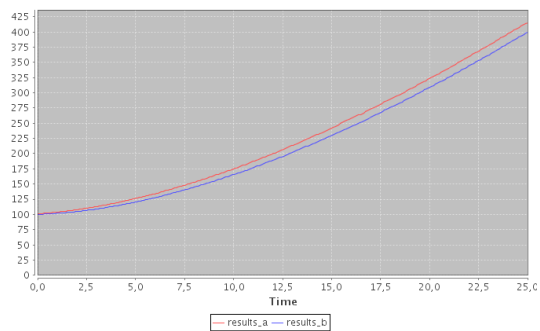
(c) time = 100 ; random time = false



(d) time = 100 ; random time = true

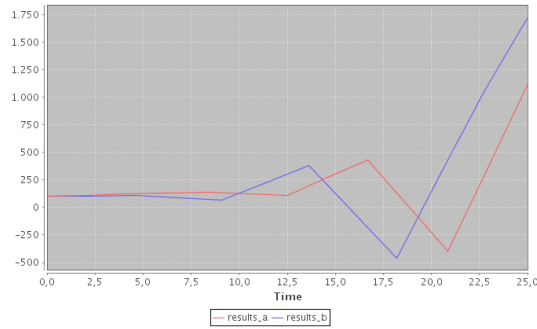


(e) time = 500 ; random time = false

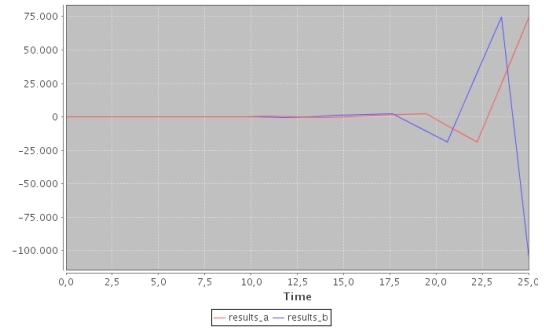


(f) time = 500 ; random time = true

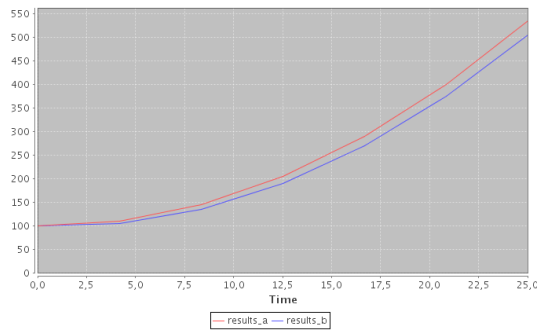
FIGURE 31: Simulator: the escalation model – asynchronous engine (delta = 0.1)



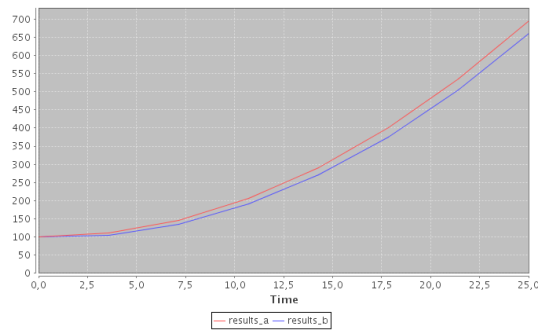
(a) time = 10 ; random time = false



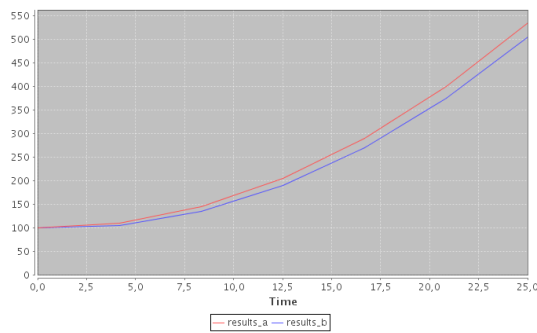
(b) time = 10 ; random time = true



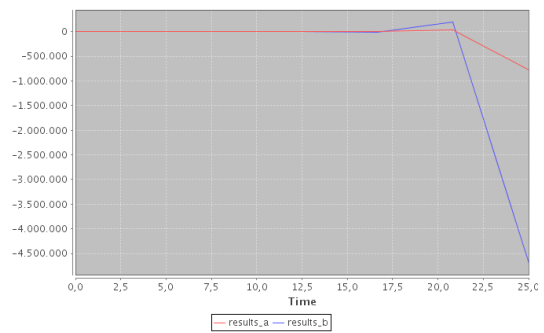
(c) time = 100 ; random time = false



(d) time = 100 ; random time = true



(e) time = 500 ; random time = false



(f) time = 500 ; random time = true

FIGURE 32: Simulator: the escalation model – asynchronous engine (delta = 5)

Success to the Successful

```

1  functor
2  import
3      Agent at 'BaseAgent.ozf'
4
5  export
6      Success_a_rel_b
7      Ress_alloc
8      B_success
9      A_success
10     A_resource
11     B_resource
12
13  define
14  class Success_a_rel_b from Agent.baseAgent
15      meth m(In ?Out)
16          Out = In.a_success - In.b_success
17      end
18  end
19  class Ress_alloc from Agent.baseAgent
20      meth m(In ?Out)
21          Out = 0.1 * In.success_a_rel_b
22      end
23  end
24  class B_success from Agent.baseAgent
25      meth m(In ?Out)
26          Out = In.b_res
27      end
28  end
29  class A_success from Agent.baseAgent
30      attr attribute
31      meth init(Name Value)
32          Agent.baseAgent,init(Name)
33          attribute := Value
34      end
35      meth m(In ?Out)
36          Out = In.a_res + @attribute
37      end
38  end
39  class A_resource from Agent.baseAgent
40      attr attribute
41      meth init(Name Value)
42          Agent.baseAgent,init(Name)
43          attribute := Value
44          stock := true
45      end
46      meth m(In ?Out)
47          attribute := @attribute + In.res_alloc * @delta
48          Out = @attribute
49      end
50  end
51  class B_resource from Agent.baseAgent
52      attr attribute
53      meth init(Name Value)
54          Agent.baseAgent,init(Name)
55          attribute := Value
56          stock := true
57      end
58      meth m(In ?Out)
59          attribute := @attribute - In.res_alloc * @delta
60          Out = @attribute
61      end
62  end
63  end

```

LISTING 17: Success to the Successful: Agents.oz

```

1  funcion
2  import
3      Agents at 'Agents.ozf'
4      Simulator at 'Simulator.ozf'
5
6  define
7  Graph
8  Success_a_rel_b
9  Ress_alloc
10 B_success
11 A_success
12 A_resource
13 B_resource
14 in
15 Graph = loop(agents: [A_resource#Agents.a_resource#init(a_res 50.0)#50.0
16                      B_resource#Agents.b_resource#init(b_res 50.0)#50.0
17                      A_success#Agents.a_success#init(a_success 1.0)#51.0
18                      B_success#Agents.b_success#init(b_success)#50.0
19                      Success_a_rel_b#Agents.success_a_rel_b#init(success_a_rel_b)#1.0
20                      Ress_alloc#Agents.ress_alloc#init(res_alloc)#0.1]
21          graph: [A_resource#[Ress_alloc]#[A_success]
22                 B_resource#[Ress_alloc]#[B_success]
23                 A_success#[A_resource]#[Success_a_rel_b]
24                 B_success#[B_resource]#[Success_a_rel_b]
25                 Success_a_rel_b#[A_success B_success]#[Ress_alloc]
26                 Ress_alloc#[Success_a_rel_b]#[A_resource B_resource]]
27          engine: async(10 false 5.0)
28          monitor: info(list: [a_res b_res] times: 25))
29
30      {Simulator.start Graph}
31 end

```

LISTING 18: Success to the Successful: Graph.oz

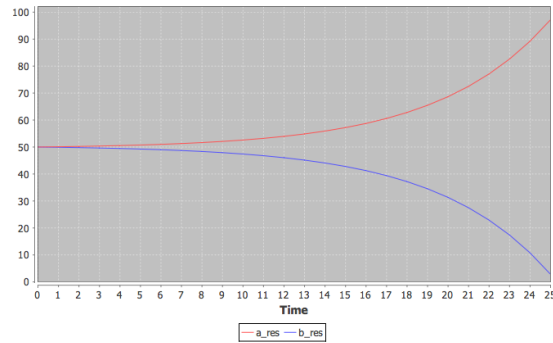
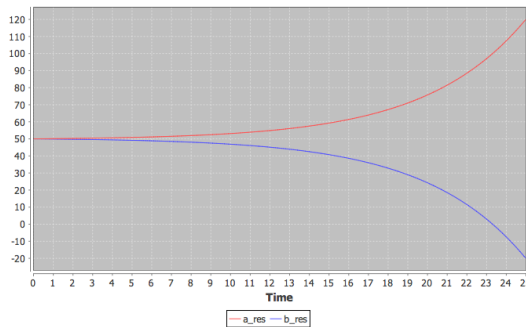
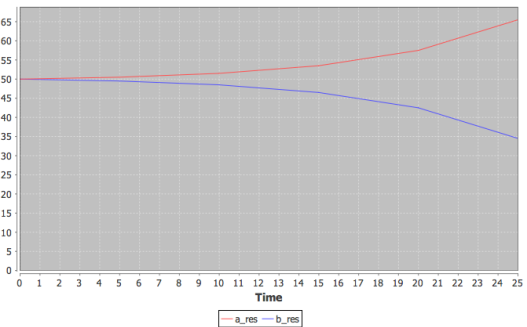
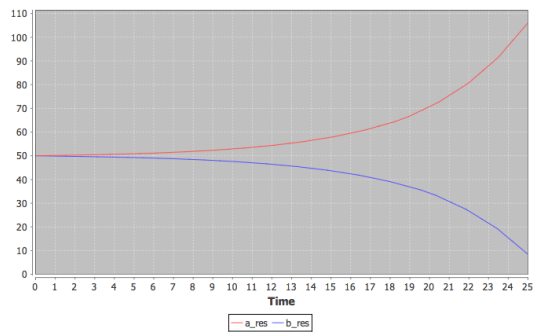
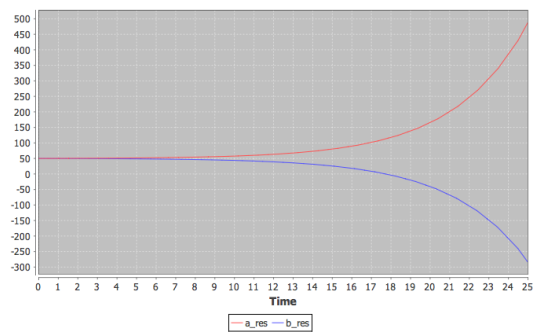
(a) $\delta = 1$ (b) $\delta = 0.1$ (c) $\delta = 5$

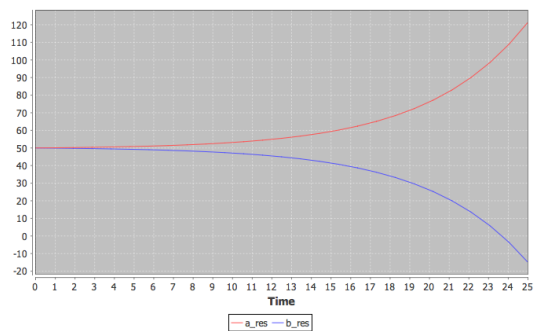
FIGURE 33: Simulator: the success to the successful model – synchronous engine



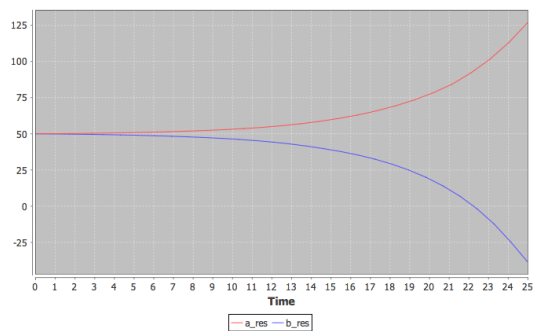
(a) time = 10 ; random time = false



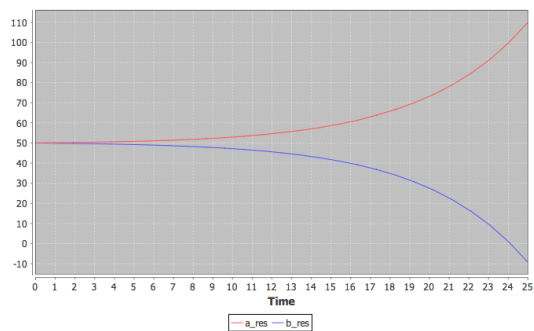
(b) time = 10 ; random time = true



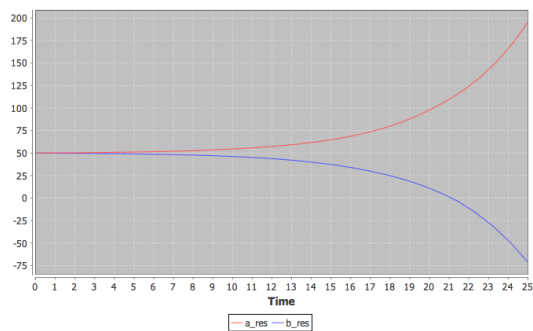
(c) time = 100 ; random time = false



(d) time = 100 ; random time = true

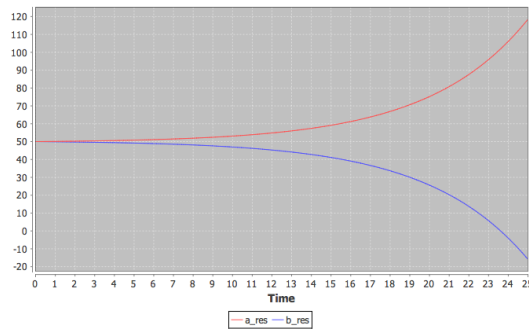


(e) time = 500 ; random time = false

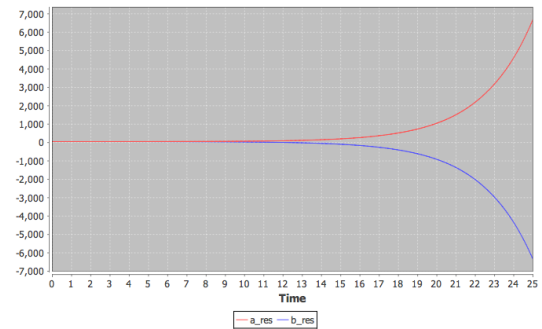


(f) time = 500 ; random time = true

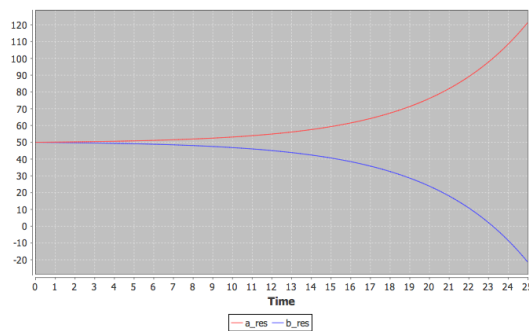
FIGURE 34: Simulator: the success model – asynchronous engine ($\delta = 1$)



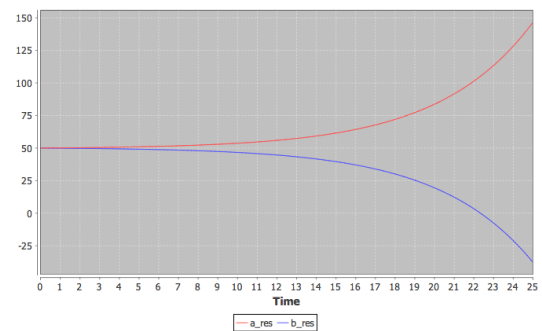
(a) time = 10 ; random time = false



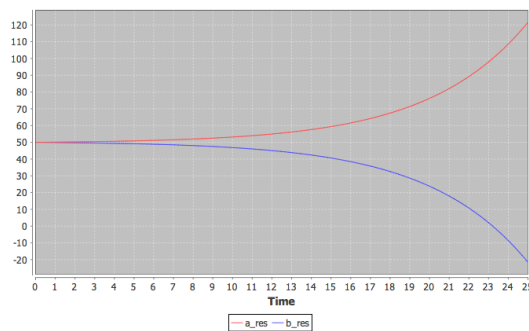
(b) time = 10 ; random time = true



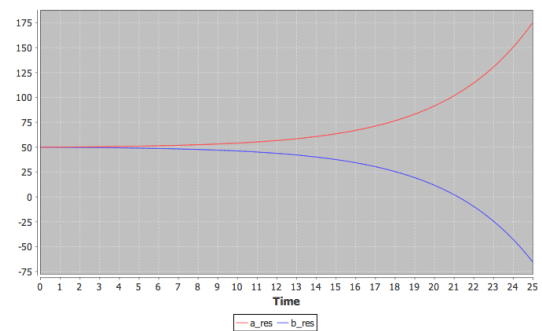
(c) time = 100 ; random time = false



(d) time = 100 ; random time = true

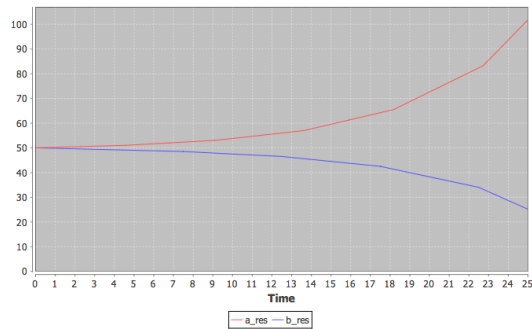


(e) time = 500 ; random time = false

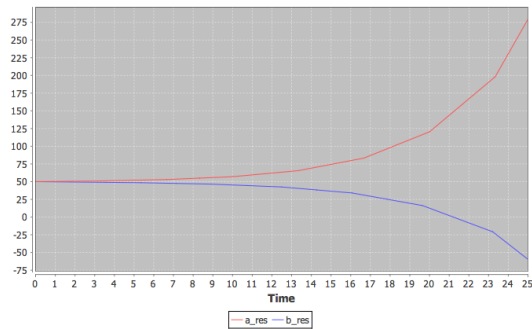


(f) time = 500 ; random time = true

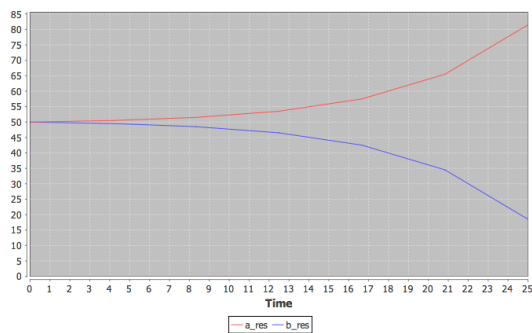
FIGURE 35: Simulator: the success model – asynchronous engine ($\delta = 0.1$)



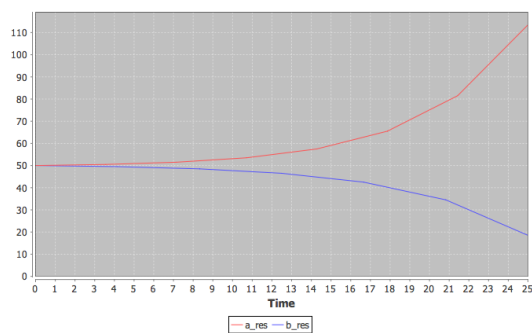
(a) time = 10 ; random time = false



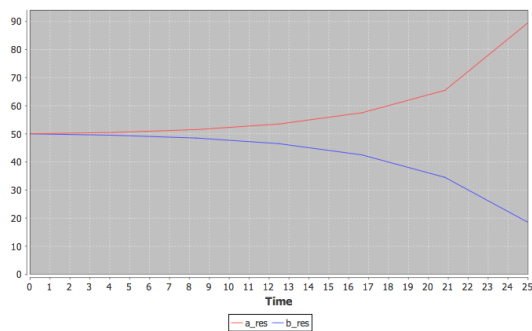
(b) time = 10 ; random time = true



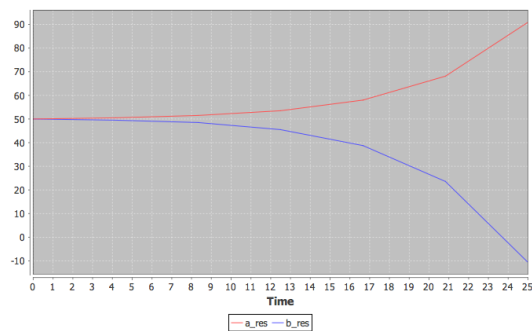
(c) time = 100 ; random time = false



(d) time = 100 ; random time = true



(e) time = 500 ; random time = false



(f) time = 500 ; random time = true

FIGURE 36: Simulator: the success model – asynchronous engine ($\delta = 5$)