Digital Business Ecosystem

Contract N° 507953

Workpackage 16
PIM Service Description

Deliverable 16.3
Report on Adaptive Service Generator

Information Society and Media

Project funded by the European Community under the "Information Society Technology" Programme

Contract number: 507953
Project acronym: DBE
Title: Digital Business Ecosystem

Deliverable Nº: D16.3
Due date: June 30, 2006
Delivery date: August 21, 2006

Short description:
This report presents the state of the art of research in adaptive service generation as an enabling step toward automatic code generation. The relevant scientific results, computational problems and infrastructural implications are presented with the aim of providing the basis for the foundation of a computational theory for Digital Ecosystems.

Author: STU (Giulio Marcon, Hisanaga Okada, Thomas J. Heistracher, and Thomas Kurz)
Partners contributed: STU, LSE, ISUFI, UBham, Soluta
Made available to: DBE Consortium and European Commission

Versioning		
Version	**Date**	**Author, Organisation**
1.2	21/08/2005 (second review)	Giulio Marcon, STU
1.1	28/07/2005 (first review)	Giulio Marcon, STU
1.0	23/06/2006 (first submission)	Giulio Marcon, STU

Quality check
1st internal reviewer: Paolo Dini, LSE
2nd internal reviewer: Paul Krause, UniS

Contents

Executive summary

This report explores the possibilities for automatic service adaptation and generation and finds its foundations on human-readable natural-language (SBVR) service notations and model-driven approaches. Adaptive services are presented as an intermediate step for code generation, a capability built on the evolutionary optimization and DBE infrastructure functionalities. The way from function-oriented Lambda calculus to communication-oriented Pi calculus is described for building the conceptual basis for further research in defining the computational basis for a Digital Ecosystem theory. In addition, real-world software engineering implications of Digital Ecosystems are covered, specifically with regard to natural-language based requirements specifications and its utilization to overcome the "digital divide in software engineering". Finally, the most difficult problems that are in the path to achieve adaptive service generation are presented, analyzing their complexity and possibilities to tackle them, proposing lines of research for future work. In the appendix a LaTeX template for SBVR is given to encourage uptake by the scientific community.

Chapter 1

Introduction

> *They are ill discoverers that think there is no land, when they can see nothing but sea.*
> (Francis Bacon, 1561–1626)

This report is part of WP16 – "Platform Independent Model Service Description" and represents the state-of-the-art research performed toward adaptive service generation in the context of the DBE project. This deliverable, together with those resulting from WP18 – "DBE UML Profile" (in particular D18.4 – "Report on self-organization from a dynamical systems and computer science viewpoint"), has to be considered as the outcome of the DBE project's efforts in the direction of adaptive service generation and the platform for future research in the subject.

Most of the work presented in this deliverable has been performed in task S24 – "Adaptive Service Generator and Gene Expression" by STU. Part of the work relates to other tasks by STU: S12 – "Mathematical Models of Fitness Landscape" and C37 – "Simulator Evolutionary Environment". The whole work reflects interaction with other partners' tasks, thanks to the several on-line and off-line discussions: S23 – "Self-organization, DNA, and formal systems" (LSE); B18 – "Business Modelling Language" (ISUFI); C59 – "BML Editor based on SBVR" (ISUFI); S2 – "Dynamics and clustering in hierarchical networks" (UBham); S4 – "Population dynamics in the Evolutionary Environment" (UBham).

This deliverable builds on the common understanding reached in D16.1 – "Service description models and language definition" (Soluta, with contributions by STU) and enhances the Service Manifest conceptual model to enable automatic adaptive capabilities.

1.1 Scientific and DBE-related objectives

The scientific aim pursued by the research presented in this deliverable is to build the basis for developing a uniform theory for one of the pillars of digital ecosystems: automatic code generation from human readable requirements analysis.

In the scope of the DBE the objective is twofold: to keep the theory in line with the current infrastructure and, symmetrically, to keep the infrastructure open for the future evolution of the system. These objectives can be considered *visionary* in the sense that adaptive service generation is not currently implemented anywhere in the DBE technical infrastructure[1] and we are not presenting *the* solution but the outcomes of several different lines of investigation. In reality the feasibility of this vision could have not been assessed without the knowledge of the capabilities of the current infrastructure and the infrastructure itself would not present such a potential without such s vision.

1.2 Aims of this paper – success criteria

This document aims at achieving the following objectives:

- to provide an extensive literature review for the foundations of a digital ecosystem unified theory, in particular for code generation;
- to make biological models used for code generation, including gene expression, accessible also to non professionals;
- to propose some lines of research that would lead to feasible implementations of automatic code generation on top of the DBE infrastructure;
- to isolate the code generation patterns that enable adaptive service generation; and
- to advance a new software engineering approach that overcomes the barriers of current methodologies, being predominantly controlled by the end user.

[1] Although conceptually rooted into the design, see for instance the service manifest conceptual model[MKHM05]

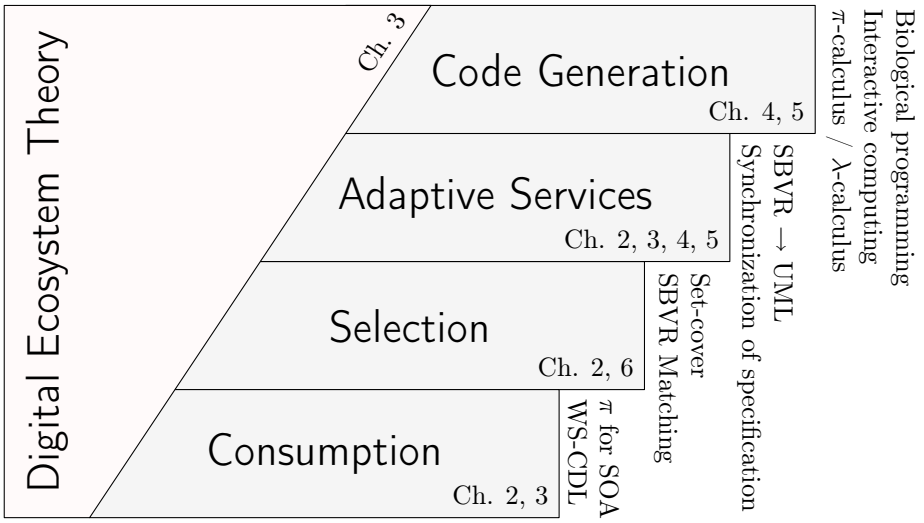

Figure 1.1: Overview of the deliverable structure, with the four steps necessary to reach automatic code generation. Although the deliverable aims more at reaching the third step, we still provide suggestions for a foundational theory of Digital Ecosystems that supports code generation.

1.3 Partners' relationship

As already mentioned, several partners influenced the work presented in this deliverable. As it happens, it is often difficult to trace back who originally introduced an idea or a new approach. For this reason, due credit is not given in this document for the specific ideas but we give here a table with the involved partners and their main areas of contribution.

Partner	**Interfacing topic**
LSE	Science vision, self-organization, autopoiesis, biological models, continuous methods
ISUFI	BML, SBVR, formal systems, software engineering, code generation and adaptation
UBham	Evolutionary algorithms, set cover, networks
HWU	Evolutionary Environment, Distributed Intelligence System
Soluta	SOA infrastructure, MDA, consistence and reality check

1.4 Document structure

This deliverable is structured in five main chapters:

- Chapter 2, Context of Application, presents the context of application of the automatic code generation research, including the relevant part of the DBE computational infrastructure and the optimization machinery of the Evolutionary Environment.
- Chapter 3, Computing with Processes, presents models of computation and suggests interactive models as the computational basis for a unified Digital Ecosystem theory.
- Chapter 4, Biologically Inspired Programming, shows how interactive models of computation are fit enough to describe biological processes and thus to pursue the biological simile of the Digital Ecosystem theory.
- Chapter 5, Software Engineering Implications, explains how natural language modelling affects software engineering, introduces a rule set for generating UML class diagrams from natural language business models and suggests future steps for generating UML activity diagrams and xUML diagrams from business rules.
- Chapter 6, Computer Science Challenges, singles out two of the most difficult problems in the way for automatic code generation. As their difficulty is theoretically founded, the only reasonable approaches are based on heuristics.

Although the chapters are rather independent, the deliverable should be read as a whole. Apart from Chapter 6, that is exposed at the end for its technical nature, the chapters proceed step by step setting the foundations needed for automatic code generation (see Fig. 1.1 for a graphical overview of the document structure).

1.5 Audience and usage of this document

The intended audience of this document is: the scientific partners of the *Digital Business Ecosystem* project, the partners of the *Open Philosophies for Associative Autopoietic Digital Ecosystems* consortium and any researcher involved in the creation of a theoretical foundation for Digital Ecosystems.

Readers interested in getting an overview of the approach used in automatic code generation are advised to skim through the whole document. In particular:

- users with a computer science theoretical background should read Chapters 3 and 4;
- users with a computer science background should read Chapter 6;
- users with a computer engineering background should read Chapter 5.

Chapter 2 should be suitable for anyone and give a reasonable overview of the infrastructure enabling adaptive service generation.

Chapter 2

Context of Application

> *The nice thing about standards is that there are so many of them to choose from.*
> (Andrew Stuart Tanenbaum, 1944–)

The Digital Business Ecosystem project presents many facets, as it tries to address several problems from different domains in an integrated way. In this chapter we will present an overview of the facets of the system useful for the purposes of adaptive service generation: the computational architecture as the underlying layer, the optimization algorithms as symbiotic mechanisms and the users as the energy that drives the system toward its stability points. Most of the contents of this chapter are an adaptation of an unpublished paper [BHK+06] co-authored with Gerard Briscoe (HWU), Philippe de Wilde (HWU) and John Woodward (UBham). The Evolutionary Environment has been originally conceived by Gerard Briscoe (HWU) and Paolo Dini (LSE) and their theoretical work is the foundational basis of its current architecture [BD04].

2.1 The Computational Infrastructure

We present first the computational architecture on top of which the adaptive service generation functionalities are built: the Digital Business Ecosystem is fundamentally an advanced version of a Service Oriented Architecture [Erl05] where several mechanisms help the offering and consumption of services. The Evolutionary Environment is one of such mechanisms that, by using evolutionary optimization techniques, concurs in making services readily available to users that could make use of

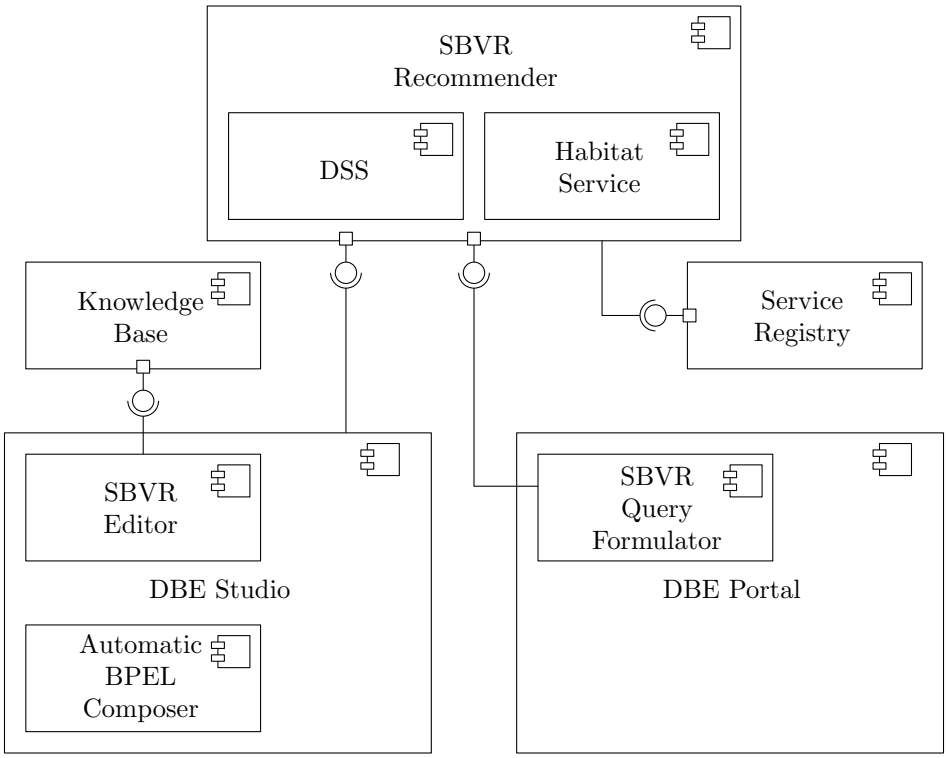

Figure 2.1: Underlying computational infrastructure that enables adaptive service generation.

them. Moreover, it helps the users in the choice of a service for a particular usage, considering the usage patterns and the instantiation context.

The adaptive service generation works in strict collaboration with the Evolutionary Environment, as it uses its mechanisms to reason on semantic service descriptions and requirements analysis (software requests). It is natural then that the interface of the adaptive service generation to the users will be the same as for the Evolutionary Environment. It consists in one of the following:

DBE Studio The DBE Studio[1] is an open-source integrated development environment built on the Eclipse platform[2]: it consists of an evolving collection of editors, tools and wizards for defining, describing, developing and deploying services. The DBE Studio is intented to be used by software developers. One of the tools in the Studio, the *SBVR Query Formulator*, allows the user to express the desired software requirements and to submit the query to the Evolutionary Environment so that a suitable collection of components is found. The orchestration of the components and the gluing code needed to satisfy the request is created by the automatic composer and can be provided to the software developer for changes and further adaptations. In this whole process the adaptive service generation will be working "behind the scenes" together with the Evolutionary Environment and Automatic Composer to provide the required software.

DBE Portal The DBE Portal is a web interface to the Digital Business Ecosystem, intended mainly for normal users of the system (service consumers). The SBVR Query Formulator will be available also through the portal, making accessible for normal users everything that was available for developers. Also in this case the adaptive service generation will work behind the scenes in an absolutely transparent way for the user, keeping all the complex machinery hidden though completely accessible.

Most of the above mentioned components are shown in Fig. 2.1, where the main relations among components are shown. This figure is meant for pointing out the main components involved in adaptive service generation; for the Evolutionary Environment and overall architecture please refer to the appropriate deliverables [Fer05a, Fer05b].

[1] `http://dbestudio.sourceforge.net/`
[2] `http://www.eclipse.org/`

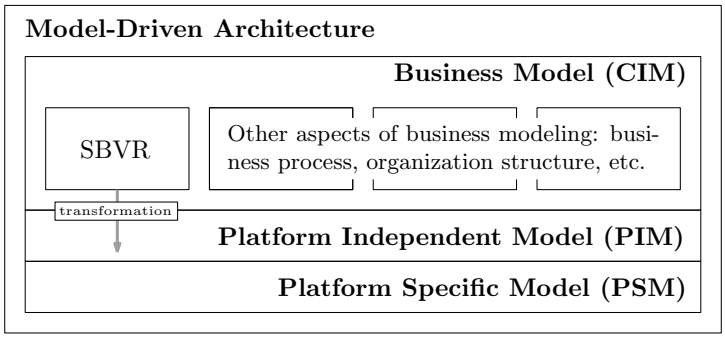

Figure 2.2: Positioning of SBVR in the Model Driven Architecture, according to the vision of the Object Management Group. Picture adapted from the SBVR specification [OMG06].

An important enabler of the adaptive service generation in the DBE is the business modelling language adopted, currently in version 2.0, that allows for extreme richness in semantic description. BML 2.0 is based on *Semantics of Business Vocabulary and Business Rules*, a standard recently approved by the Object Management Group, and is at its first interim specification as of March 2006 [OMG06].

SBVR is a meta-model specification that aims at being positioned as the Computational Independent Model of the Model Driven Architecture (see Fig. 2.2). The models are described in a controlled natural language that avoids ambiguities and consists of three parts: vocabulary, facts and rules. SBVR has a Meta Object Facility (MOF) model, so the models can be stored in MOF repositories, interchanged, and linked with other models based on MOF, including UML models: for this reason it is fully integrated into the OMGs Model Driven Architecture [Fra03, MDA].

SBVR allows for semantic integration of businesses by providing a formal description language that is still non-technical by being very close to natural language. The formal structure is inherited from logic, as the following are directly included in SBVR: typed first-order logic with equality; higher-order logic restricted to Henkin semantics (so that function variables cannot range infinitely); alethic logic for necessities; deontic logic for obligations. Additionally, it includes already coding for set theory, multi-sets and basic mathematics. An instrument for formulating queries (questions) is provided in the form of projections.

Without the rich expressiveness of SBVR, many of the ideas presented in this deliverable would not be realizable. Still, it is important to remember that the business models described through SBVR are not direct descriptions of the under-

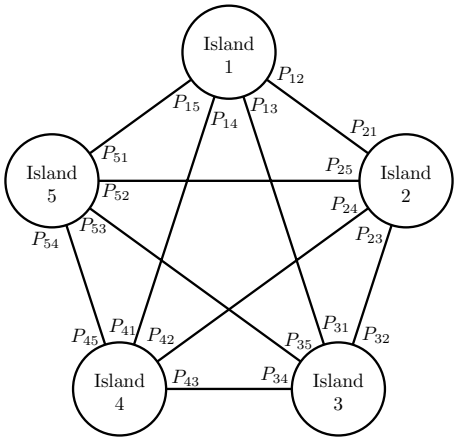

Figure 2.3: Island Model of Distributed Evolutionary Computing

lying computational services but rather descriptions of the business services. Each business service can then have computational services connected to it as operations. This could be clearer with an example: an SBVR description of a hotel would describe the hotel features, the number of rooms, the price range, and so on, but will not say if a certain room is available for a particular date. With this model there will be some operations associated that will for instance allow the user to check for the availability of rooms, to book a room, or to make inquiries to the hotel. The computational services are described using the appropriate DBE language for describing the Platform Independent Model (the Service Description Language).

2.2 The Evolutionary Optimization

The current evolutionary optimization machinery for the DBE is a continuous distributed optimization that feeds and enhances a local optimization in every node aimed at satisfying some constraints that are specified in, and relevant for, an individual node itself. This twofold process speeds up and ameliorates the quality of a local search by giving hints based on the computations already performed in other nodes having similar constraints[3].

Distributed evolutionary computing aims at achieving parallel processing to find solutions more efficiently. In the *island model*, probably the most analogous to the notion of migration in nature [Gol89], a distance is set between the sub-populations

[3]For the details see deliverable D9.2 – "Report on evolutionary and distributed fitness Environment"

on each island, together with the probability of migration between one island and another. In Figure 2.3, different probabilities are defined for going from island ① to island ② and from island ② to island ①. This allows maximum flexibility for the migration process. It also mirrors the naturally inspired quality that, although two populations have the same separation no matter how one looks at them, it may be easier for one population to migrate to another than *vice-versa*. An example of this from nature could be the situation where it is easier for one species of fish to migrate downstream than for a different species to migrate upstream. This model has been used successfully in the determination of investment strategies in the commercial sector in a product known as the Galapagos toolkit [Fle05].

Within the Digital Ecosystem the distributed evolutionary computing component comes from evolving populations being created in response to similar requests. So, whereas in the island model there are multiple evolving populations in response to one request, here there will be multiple evolving populations in response to similar requests. This is shown in Fig. 2.4 where the shading of the evolving populations indicates similarity in the requests being managed.

For example, the four Habitats in the top left of Fig. 2.4 may all be travel agencies able to organise travel arrangements, where the three remaining habitats host evolving populations that may be looking for package holidays within the same continent. A "great deal" solution found and used in one Habitat will then be migrated to the other connected Habitats where it will be integrated into any evolving populations via the local agent-pool, so that it will potentially help to optimize the search of similar package holidays at the Habitats of the other travel agencies. The agent-pool is the set of agents registered at the Habitat (Agent Station), and is used like a gene-pool for the local evolutionary optimization. This will also work in a time-shifted manner because the great deal solution will be stored in the agent-pool of the Habitats to which it is migrated, so that it will potentially be available to optimize a similar request placed later.

The Habitat also provides a place for agent migration to occur, as the Habitats are interconnected with one another. The migration of an agent within the ecosystem is initially triggered by deployment to its Home Habitat for distribution to business users who will potentially use the agent. The successful use of the migrated agent in response to the business user requests for applications, will lead to further migration (distribution) and therefore availability of the agent to other po-

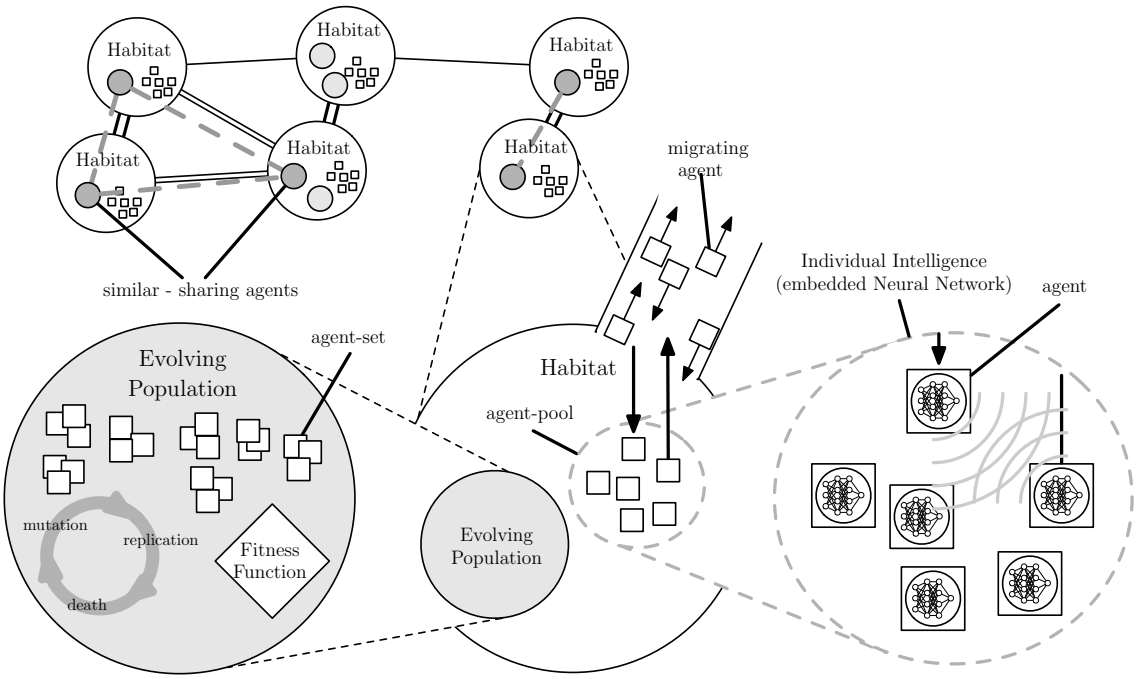

Figure 2.4: Visualization of the Digital Ecosystem (figure by Gerard Briscoe, HWU). The novel distributed evolutionary computing, in which multiple populations respond to similar requests while sharing agents (solutions), indicated by the gray dashed lines between the Evolving Populations. The interaction of the agents within the agent-pool, via the Neural Networks based Individual Intelligence components embedded within the agents, creates the distributed intelligence within the Digital Ecosystem

tential business users. The migration path from the current Habitat is dependent on the migration probabilities associated between the Habitats, similarly to the island model mentioned above. The success of the migration, the migration feedback, leads to the reinforcing and creation of stronger migration links between Habitats, just as the failure of migration leads to the weakening and negating of migration links between Habitats.

There will be at least hundreds of Habitats, as there is one for each business entity, and potentially three or more times the number of populations at any instance in time. There will then be thousands of agents and applications (groups of agents) available to meet the application requests of the business users within the DBE. The union of the Habitats is the Digital Ecosystem. The connectivity between the Habitats will correspond to business sector similarities and interactions that exist between the SMEs.

The feedback from agent migration will result in a *caveman* topology of the Habitats within the Digital Ecosystem, with 'caves' of high connectivity being the equivalent to the strong interactions within business sectors. This will require that the business user base is sufficiently large.

When a user triggers creation of a new service, its representative agent is deployed at the user's Home Habitat. The agent is then copied (migrated) to any Habitat connected to the home Habitat, conditionally on the probabilities associated with the connections, to make it available in other Habitats where it could potentially be useful.

The agent is then available to the local evolutionary optimization to be used in response to a user request. When an agent is used in a Habitat, an attempt is made to copy (migrate) it to every other connected Habitat: the success of every migration operation depends on a probability associated with every connection, similarly to the island model shown in Fig. 2.3.

Agent migration, via copying, is only allowed after the agent is actually used. An agent can also be deleted in case it did not prove useful for several user requests: it will have a small number of free migrations, in which it is not copied, but moved to another Habitat. If the agent fails to find a 'niche' before running out of free migrations, it will be deleted.

2.2.1 Local Optimization

The local optimization problem consists of finding the best combination of the agents available in the local agent station to satisfy a particular user request. As a combination of agents is a variable sized set, there are some issues arising with the usage of genetic algorithms.

Genetic algorithms are concerned with the evolution of fixed length bit strings to solve a given problem, where each bit string is interpreted as a potential solution to a problem. In some cases the nature of the problem does not allow us to determine what the length of a solution is beforehand and therefore a variable length approach must be adopted, as is usually the case with genetic programming and more specifically in our Digital Ecosystem optimization approach.

When variable length representations of solutions are used, a well known phenomenon, called *bloat*, arises: the individuals in the population tend to grow in size during evolution. This causes a reduction in performance time (typically fitness of longer bit strings takes longer to be evaluated), but also ultimately causes the algorithm to terminate due to the finite amount of memory of the machine on which the algorithm is running.

As reported in Langdon, *"bloat is not specific to genetic programming and suggests it is inherent in search techniques with discrete variable length representations"* [Lan97]. Similarly we can take the viewpoint that whether we are looking at variable length genetic algorithms or genetic programming (the former using linear structures and the latter using tree structures), ultimately in both cases the individuals in the population are stored as linear structures in the computer's memory. A lot of work on bloat has been done in connection with genetic programming, and we believe that the genetic algorithm community can benefit directly from this research.

Before attempting to cull the bloating experienced during a search process, it is worth considering the causes of this effect. If we understand the origins, then we may be in a better position to produce preventative measures to deal with bloat. There are a number of different qualitative theories which attempt to explain bloat.

McPhee et al. [MP00] review a number of theories of bloat. Firstly, protection against crossover and bias removal (which we will lump together) and secondly, the nature of program search spaces.

Firstly, toward the end of a run, the population consists of relatively fit individuals and any crossover is likely to be detrimental to the fitness of the offspring. In

fact, in any combination of services there may be services that do not contribute semantically to the overall functionality of the combination if, for example, their functionality was not requested by the user or it is duplicated in the combination; similarly to genetic programming, we can call these reduntant services *introns*. The genotype can then be grown further without affecting the phenotype if services with similar functionalities are added: but, as the genotype grows larger, crossover is more likely to transfer redundant services to the new offsprings (assuming uniform crossover).

Secondly, above a certain threshold size, the distribution of functionality does not vary with the size of the search space. Thus if we randomly sample large and small services sets above the size threshold, we are likely to get combinations having the same functionality with the same probability. As a search process progresses, we are therefore more likely to sample bigger sets of services, as there are more of them (all other things being equal) and this will give rise to the bloating phenomenom.

Each stage of the construction of a genetic algorithm (i.e.: choice of fitness function, selection method and genetic operator) can have an effect on bloat. As McPhee et al. show that even small differences in fitness function can cause a difference: *"a single program glitch in an otherwise flat fitness landscape is sufficient to drive the average program size of an infinite population."* [MP00]. If fitness-proportional selection is used, individuals with zero fitness will be discontinued as they have zero probability of being selected as parents. However, if tournament selection is used, then there is a finite chance that individuals with zero fitness will be selected to be parents. Finally, the choice of genetic operator affects the size of the programs which are sampled; *"standard crossover on a flat landscape heavily over samples the shorter programs"* [PM01]. It should also be noted that some researchers report they experience no bloat due to the representation they use [Mil01]. This list is not exhaustive and other factors may affect bloat, for example how the population is initialised.

While bloat is a phenomenon which was first observed in practice, theoretical analyses have been attempted [PM01, MP01]. One should take care with these approaches as implementations will always deal with finite populations while theoretical approaches often deal with infinite populations and these differences can be important. We support both theoretical and empirical approaches to understanding bloat.

Bloat is a fundamental area of research regarding search based approaches such as genetic algorithm, genetic programming and other non-population based approaches such as simulated annealing [Lan97]. There are likely to be many factors contributing to bloat, and while the phenomenon may appear simple, the reasons are not. Bloat is a fact, whatever the reasons, happening in this kind of optimisation and needs to be controlled if the space is to be searched effectively. One solution is to apply a hard limit to the size of the sets that can be sampled: this enables the search algorithm to keep running without having out-of-memory runtime errors, but poses questions as to how to set this hard limit. An alternative but similar method is to apply parsimony pressure, where a term is added to the fitness function which punishes big sets in preference for smaller sets. Again this is a rather ad hoc method, and raises the question of how to balance the 'real' fitness with parsimony pressure.

Poli introduces a theoretically inspired method of controlling bloat [Pol03]. Individuals in the population which are larger than the average size of an individual in the population are evaluated with reduced probability. This biases the search to smaller sets and it is a dynamic limit which adapts as the average size of individuals in the population changes. This method has the built-in benefit that an individual in the population does not need to be evaluated (which is usually a more costly process than calculating the size of an individual), and therefore can be discarded quickly, improving the computational time of the algorithm. This is an important point as often algorithms are compared using the number of evaluations rather than processor time.

2.2.2 Distributed Optimization

This section is by Gerard Briscoe (HWU) and summarizes part of the work contained in deliverable D6.4 - "Intelligence, Learning and Neural Networks in the Distributed Agent Model of the Evolutionary Environment". The *distributed intelligence* will work to complement the local evolutionary optimization. If each company offers one or more services, represented as agents within the Digital Ecosystem, then the total number of services in the system will be large. The efficiency of the local evolutionary optimization process at the Habitats increase with the size of the Ecosystem (number of peers connected) as in selecting an optimal set of agents it would have more help from neighboring nodes. So, optimal subsets of the available agents are required at the Habitats. This will allow the local evolutionary optimization to continue to work

efficiently as the Digital Ecosystem scales up in size. The migration probabilities between the Habitats work to achieve this in a 'passive' manner. Although the mechanism is effective, the migration of agents will be slow and not strongly directed. It allows the agents, based primarily upon their success at their current location, to spread to the correct general direction in the Digital Ecosystem (Habitat network). The *distributed intelligence* works in a more 'active' manner allowing the agents to direct their migration to specific Habitats, rather than generally directed migration. This helps to optimize the subset of agents found at the Habitats, which are then used in the local optimization process to find applications (combinations of agents) to user requests. There are many ways in which this agent behaviour could be achieved.

The approach chosen maintains the consistency of the mobile agent system and Digital Ecosystem models: the intelligence is within the agents instead of the Habitats. For example, the Habitats could communicate more and share agent (service) usage information more directly, but this is not a behaviour found within a biological ecosystem or often within mobile agent systems. Behaviour in which agents communicate and share information about suitable Habitats is much closer to biological ecosystems and known within mobile agent systems. The intelligence mechanism for the *distributed intelligence* is biologically inspired.

Neural network-based *individual intelligence* components are embedded within the agents, and thereby distributed throughout the Habitat network. The agents can then interact with one another, determining if other agents are functionally similar. If the agents are similar, they can compare their 'migration and usage' histories and determine Habitats where they could be valuable. This interaction occurs outside of the local evolutionary optimization, in the agent-pools of the Habitats.

The pattern recognition capabilities of neural networks are leveraged to allow agents to determine similarity based on their descriptions. The neural networks are multilayer feed-forward, and use the back-propagation training technique to specify the required pattern recognition behaviour. As the ideal scenario of having a large training set upon which to train a neural network is not possible in the Digital Ecosystem, a continuous training regime is being followed. Firstly, as the agent needs to be trained to recognize similarity to itself, the training set will be its own description. At the start of the agents life-cycle the pattern recognition capabilities of its neural network would be limited but still functioning. So the agents could

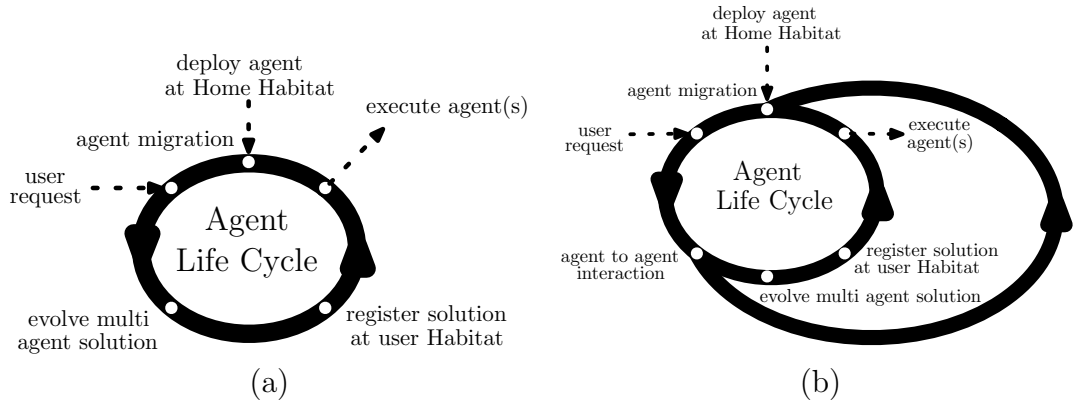

Figure 2.5: (a) Agent life-cycle (b) New agent life-cycle (figure by Gerard Briscoe, HWU)

train their neural network based on experience. The training set can be extended from the result of visiting a Habitat based on an inter-agent interaction, whether it proves to be successful. Both success and failure are equally useful. Additionally, untrained inexperienced agents could copy the neural network weightings of a more experienced agent when similarity is determined during inter-agent interactions.

The main effect of the *distributed intelligence* can be seen in the changes to the agent's life-cycle, which is depicted in Fig. 2.5. Basically, where and when the trigger for agent migration will occur has been changed. There will be more opportunities for agent migration, but more importantly these opportunities are for targeted migration.

The interaction between the *distributed intelligence* and the evolutionary dynamics will be studied further but will not be subject to the Baldwin effect as the individual intelligence components will not be subject to the evolutionary optimization [Tur96]. This approach will make the agents even more like software agents by giving them a form of intelligence and direct control over their behaviour. This combined with the migration will make the agents within the Digital Ecosystem similar to agents in a mobile agent system. A very weak generic definition of agent has been used thus far, albeit a fundamentally correct one. This approach will alleviate this criticism.

2.3 Reaching critical mass

The Digital Business Ecosystem, similarly to biological systems, needs a constant input of high-free energy "food" to remain "alive". In the case of the DBE, the high-free energy food consists in informational input from the users: the whole adaptive service generation, together with the vision of automatic code generation, could not happen without a constant contribution from them. For further information on the DBE as a biological system, refer to D18.1 - "Report on DBE-specific Use Cases".

In the ecosystem metaphor, for an ecosystem to survive there has to be a certain number of species (kinds of services) and individuals (the services themselves), with their corresponding habitats (the companies offering the services): this number is called the *critical mass*. Given the difficulty of modelling the digital business ecosystem as a whole, it will probably be impossible to determine exactly the critical mass, but the use of simpler models can give hints. In particular, the use of simulators making use of these models and of the real-world algorithms (even if on a much smaller scale, that is in a single computer instead that on a network) can give quite precise measurements of the required minimum number of companies, services and providers needed in the system. This is the aim of task C37 - "Simulator Evolutionary Environment" and the results of the research in this merit are presented in deliverable D9.2 - "Report on evolutionary and distributed fitness environment". The critical mass can as well be seen from a mere physical point of view as the phase transition from order to disorder; an interesting experiment with ant colonies showed that a minimum number of about 600 ants was needed to achieve self-organization [DB04].

One of the assumptions for automatic code generation implies that there are plenty of atomic components in the system, that is components that provide a basic and well defined functionality, offering an interface that avoids particular dependencies and that can be adapted easily to any kind of use (technically said as *loosely coupled*). Such components are said to be *finely grained*, as each one of them offers a solution to a small isolated problem; on the contrary big monolithic components are said to be *coarsely grained*. Without such components, or better without the providers able to offer them, the adaptive code generation would not be possible. The same way, adaptive service generation could not be boot-strapped without the presence of integrators in a first phase.

We will present in the next chapters how the interactions among atomic services

can be described and organized to achieve self-organization of components for automatic software and code generation, out of the pressures of the market driven by the user requests.

Chapter 3

Computing with Processes

> *When we had no computers, we had no programming problem either. When we had a few computers, we had a mild programming problem. Confronted with machines a million times as powerful, we are faced with a gigantic programming problem.*
> (Edsger W. Dijkstra, 1930-2002)

This chapter is organized in the following way: first, models of computation are introduced, focusing on two particular calculi, λ-calculus [Bar84] and π-calculus [Mil99, SW01]; then, their equivalence is shown; finally, their applicability as the theoretical base for code generation is presented.

3.1 Models for computation

Computer science started as an autonomous line of research long before modern digital computers existed: Alonzo Church, one of the pioneers, defined in the beginning of the thirties a formalism aimed at becoming a foundation for logic and mathematics. The formalism was called λ-calculus [Chu32a, Chu32b].

Unfortunately, Kleene and Rosser, two students of Church, proved the inconsistency of the calculus some years later by showing that all the terms could be equated [KR35]. Several paths were taken to make the system consistent: the one followed by Church together with his students was to isolate an appropriate subset of the calculus that is exactly what we know nowadays as λ-calculus (proved to be consistent by the same Church and Rosser).

Very soon Church realized that the terms definable in his theory corresponded to *effectively computable* functions (function for which an *algorithm* can be written). That is, at the same time the following people were trying to capture the concept of computability as:

- Turing as the programs writable for the Turing machine;
- Gödel and Hebrand with the theory of recursive functions;
- Church with his λ-calculus.

All these formalisms, developed independently and almost at the same time, although being very different in their nature, proved to be equivalent and thus to represent exactly the same class of functions. Additionally, for each formalism there exist predicates that cannot be decided, that is they cannot be described algorithmically.

Some of the main features of λ-calculus are that:

- it focuses on functions, so they are "first class citizens" in the calculus;
- there is no distinction between programs and data; a program can be given as input to another program (this eliminates the need of coding functions as in classical computability theory);
- it is intensional, that is terms that are different in the theory can actually compute the same thing and thus look externally as if they are exactly the same program; an extension to the theory can anyway make the calculus extensional.

Computing models like λ-calculus were well suited to describe the way a computational system behaved and satisfied every need of having a proper way to represent data and algorithms that was abstracted from the actual implementation. Of course these models were not enough once computer networks gained importance, when it became important to capture the new capabilities of interaction and collaboration among systems.

It is in this context, around the beginning of the nineties, that the π-calculus was born to give an understanding of communicating systems in the same way that λ-calculus gave an understanding of computer programs, being able to capture all possible patterns of communication among systems, allowing to model computer

networking, mobile computing, distributed systems and so forth. The calculus has been so successful that it is used nowadays in many derived forms, built with the spirit over the original π-calculus but aimed at different types of systems, up to biological ones. Another advantage of π-calculus is that is does not lose capability to describe programs but, on the contrary, λ-calculus can be interpreted in it in several of its reduction strategies.

3.2 The λ-calculus

As already mentioned, the focus of λ-calculus is on functions. Let us consider a generic expression E defining a function f of a variable x; a common practice is to write it as $f(x) = E$. The equivalent λ-notation, used also in mathematics (but more often in computer science) is the following:

$$\lambda x.E$$

This notation does not introduce any new symbol to denote the function, as the whole expression is already a function, being λ the functional abstraction operator.

Application is simply expressed by concatenating the functional abstraction with the parameter.

Example 3.2.1 (Application) *Given a function $f(x) = 4x^2 - 2x + 1$, its corresponding lambda notation, using normal mathematical terms, is $\lambda x.4x^2 - 2x + 1$. Instead of $f(2)$ we write then in lambda notation:*

$$((\lambda x.4x^2 - 2x + 1)2)$$

The functional abstraction can be used only for unary function, but this is not a problem, as we can always use currying, a specific version of the s_{mn} theorem [Kle52], to reduce a function with n parameters into n functions with one parameter (see the following example).

Example 3.2.2 (Currying) *Given a function $f(x, y) = E$, we can consider the equivalent lambda notation $\lambda x.\lambda y.E$. The application of this term to its arguments*

consecutively gives the following:

$$((\lambda x.\lambda y.E)a)b) = f(a, b)$$

Where λx and λy are functional abstractions and a and b are the parameters of the function.

As the model of lambda calculus is rather simple, we would like to present it here for completeness, also to show how this very simple machinery can describe the full capabilities of a computational system. We will define the expressions of λ-calculus (λ terms), the reduction rules and some useful properties and applications of the calculus.

Definition 3.2.3 (λ pre-terms) *Given an alphabet of variables:*

$$V = \{v_0, v_1, \dots\}$$

the pre-terms Λ^- are defined by the following grammar:

$$\begin{array}{rcl} \Lambda^- & ::= & V \\ & | & (\Lambda^-\Lambda^-) \\ & | & (\lambda V.\Lambda^-) \end{array}$$

Conventionally, upper case letters are used to represent elements of Λ^-, while lower case letters represent elements of V. As an example, $(\lambda v_0.(v_0 v_1))(\lambda v_2.(v_2 v_3))$ is a pre-term. Often parenthesis are omitted, using a left precedence policy (the leftmost terms associate first), as well as multiple subsequent lambda abstractions are grouped together.

Definition 3.2.4 (Free variables) *Given a pre-term $M \in \Lambda^-$, the set $FV(M)$ of the free variables of M is defined inductively in the following way:*

$$\begin{array}{rcl} FV(x) & = & \{x\} \\ FV(\lambda x.P) & = & FV(P) \setminus \{x\} \\ FV(PQ) & = & FV(P) \cup FV(Q) \end{array}$$

A pre-term is closed when it has no free variables.

All the variables that are not free in a lambda pre-term are said to be *bound*.

Definition 3.2.5 (Substitution) *Given the pre-terms $M, N \in \Lambda^-$ and a variable $x \in V$, the substitution of N for x in M, written as $M[x := N]$, is defined inductively as:*

$$\begin{array}{rcll} x[x := N] & = & N & \\ y[x := N] & = & y & \\ (MP)[x := N] & = & M[x := N]P[x := N] & \\ (\lambda x.M)[x := N] & = & (\lambda x.M) & \\ (\lambda x.M)[x := N] & = & (\lambda y.M)[x := N] & \text{if } y \notin FV(N) \vee x \notin FV(M) \\ (\lambda x.M)[x := N] & = & (\lambda z.M)[y := z][x := N] & \text{if } y \in FV(N) \wedge x \in FV(M) \end{array}$$

Now that we have the notion of substitution, we can see the first important equivalence of λ-calculus: it is clear that we can write a pre-term that describes the same function in many different ways, just changing the bound variables used. The equivalence defined here express exactly this: pre-terms being different only for the name of bound variables are the same.

Definition 3.2.6 (α-equivalence) *α-equivalence, written $=_\alpha$, is the smallest relation such that:*

$$\begin{array}{rcll} M & =_\alpha & M & \text{for each } M \in \Lambda^- \\ \lambda x.M & =_\alpha & \lambda y.M[x ::= y] & \text{if } y \notin FV(M) \end{array}$$

and is closed with respect to the following rules:

$$\begin{array}{lll} M =_\alpha M' & \Rightarrow & (\forall x \in V).(\lambda x.M =_\alpha \lambda x.M') \\ M =_\alpha M' & \Rightarrow & (\forall Q \in \Lambda^-).(MQ =_\alpha M'Q) \\ M =_\alpha M' & \Rightarrow & (\forall Q \in \Lambda^-).(QM =_\alpha QM') \\ M =_\alpha M' & \Rightarrow & M' =_\alpha M \\ M =_\alpha M' \wedge M' =_\alpha M'' & \Rightarrow & M' =_\alpha M'' \end{array}$$

The α-equivalence relation can be used to rename bound variables in a pre-term and is fundamental to finally define the lambda terms.

Definition 3.2.7 (λ terms) *The set of lambda terms Λ is defined as the quotient*

set of the pre-terms with respect to the α-equivalence relation, that is:

$$\Lambda \quad = \quad \Lambda^{-}/_{=_\alpha}$$

The definition of free and bound variable for lambda terms are equivalent to those for pre-terms.

Terms are the programs of λ-calculus: apart for the notion of equivalence of terms due to renaming of bound variables (α-equivalence), their syntax is so easy that anyone can grasp it. Terms are the only elements of the calculus: they express in fact not only the programs but the data themselves. Any program can thus be used as input to another program. This is a shortcut to a notion that has always been used in computability theory using an isomorphism between programs and data. Traditionally programs and data were mapped to natural numbers; this is possible as the computable functions are countable (their set has cardinality $\aleph_0$)).

The computational behaviour is described through β-reduction, a relation that, in its simplicity, has many nice properties and allows λ-calculus to achieve Turing-completeness.

Definition 3.2.8 (Reduction relation) *A binary relation R over Λ is said to be a reduction iff:*

$$\begin{aligned}
(M, M) &\in R && \textit{(reflexivity)}\\
(M, N) \in R &\Rightarrow \forall Z, (MZ, NZ) \in R && \textit{(compatibility)}\\
(M, N) \in R &\Rightarrow \forall Z, (ZM, ZN) \in R && \textit{(compatibility)}\\
(M, N) \in R &\Rightarrow (\lambda x.M, \lambda x.N) \in R && \textit{(compatibility)}\\
(M, N) \in R \wedge (N, Z) \in R &\Rightarrow (M, Z) \in R && \textit{(transitivity)}
\end{aligned}$$

Definition 3.2.9 (β-reduction) *Let $\rightarrow_\beta$ be the smallest relation over Λ such that (we use the infix notation):*

$$(\lambda x.M)N \rightarrow_\beta M[x := N]$$

and such that it is closed with respect to:

$$\begin{aligned}
M \rightarrow_\beta M' &\Rightarrow (\forall x \in V).(\lambda x.M \rightarrow_\beta \lambda x.M')\\
M \rightarrow_\beta M' &\Rightarrow (\forall N \in \Lambda).(MN \rightarrow_\beta M'N)\\
M \rightarrow_\beta M' &\Rightarrow (\forall N \in \Lambda).(NM \rightarrow_\beta NM')
\end{aligned}$$

A term M is said to be in β-normal form when there is no term N such that $M \to_\beta N$.

A computation is then a set of β-reduction steps until a term reaches its normal form. As in computer programs, it could be that a program never reaches a normal form and thus proceeds indefinitely. We present now some examples of reductions.

Example 3.2.10 (Computing with λ-calculus) *The following is a very simple example:*

$$\lambda x.xy \to_\beta (\lambda z.z)y \to_\beta y$$

This is a more complicated example that computes $T \wedge F$, using Church booleans (true is $(\lambda xy.x)$, false is $(\lambda xy.y)$, the function "and" is $\lambda pq.pq(\lambda xy.y)$):

$$\begin{gathered}(\lambda pq.pq(\lambda xy.y))(\lambda xy.x)(\lambda xy.y) \to_\beta \\ (\lambda q.(\lambda xy.x)q(\lambda xy.y))(\lambda xy.y) \to_\beta \\ (\lambda xy.x)(\lambda xy.y)(\lambda xy.y) =_\alpha \\ (\lambda xz.x)(\lambda xy.y)(\lambda xy.y) \to_\beta \\ (\lambda zxy.y)(\lambda xy.y) \to_\beta \\ (\lambda xy.y)\end{gathered}$$

Example 3.2.11 (Computing with λ-calculus) *An example showing a term that has no normal form as it reduces to itself:*

$$(\lambda x.xx)(\lambda x.xx) \to_\beta (\lambda x.xx)(\lambda x.xx) \to_\beta \ldots$$

Example 3.2.12 (Computing with λ-calculus) *Finally, an example showing that the reduction strategy (the choice of* what *to reduce) is important; we achieve this by "embedding" the example from before in a bigger term able to "cancel" it.*

If we consider the term $(\lambda xy.x)(\lambda x.x)((\lambda x.xx)(\lambda x.xx))$, and always reduce the rightmost redex, we cannot get to a normal form:

$$(\lambda xy.x)(\lambda x.x)((\lambda x.xx)(\lambda x.xx)) \to_\beta (\lambda xy.x)(\lambda x.x)((\lambda x.xx)(\lambda x.xx)) \to_\beta \ldots$$

While, if we reduce the leftmost redex first, we can get to a normal form in two steps:

$$(\lambda xy.x)(\lambda x.x)((\lambda x.xx)(\lambda x.xx)) \to_\beta (\lambda y.(\lambda x.x))((\lambda x.xx)(\lambda x.xx)) \to_\beta (\lambda x.x)$$

Beta reduction is a very powerful instrument: as we already said, it lets λ-calculus achieve Turing-completeness. As in many other languages, this is achieved by the use of recursion: we show how a fixed point operator can be defined for defining recursive functions.

Definition 3.2.13 (Curry fixed point combinator) *A fixed point combinator for λ-calculus, denoted $\mathbf{Y}$, has the following property:*

$$\forall F \in \Lambda, \qquad F(\mathbf{Y}F) = \mathbf{Y}F$$

Traditionally, the combinator is defined as:

$$\mathbf{Y} \equiv \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$$

To verify its property:

$$\begin{array}{rcl} \mathbf{Y}F & \equiv & (\lambda f.(\lambda x.f(xx))(\lambda x.f(xx)))F \\ & \rightarrow_\beta & (\lambda x.F(xx))(\lambda x.F(xx)) \\ & \rightarrow_\beta & F(\lambda x.F(xx))(\lambda x.F(xx)) \\ & =_\beta & F((\lambda f.(\lambda x.f(xx))(\lambda x.f(xx)))F) \\ & \rightarrow_\beta & F(\mathbf{Y}F) \end{array}$$

Traditional λ-calculus terms having the same *functional behaviour* can be distinguished depending on their inner computational process. More precisely, two terms defined in different ways (not β-equivalent) but that behave exactly the same way (they have the same return values for any possible argument) are still not considered equal. This is the *intensional* property of the calculus.

For some applications (including our case), it could be desirable to describe computational processes *extensionally*: this can be achieved in λ-calculus by adding a new axiom. The resulting theory is called $\lambda\eta$.

Definition 3.2.14 ($\lambda\eta$ theory) *The $\lambda\eta$ theory is equivalent to the normal λ-calculus with the following addition axiom (ξ rule):*

$$\forall M, N \in \Lambda \qquad Mx = Nx \Rightarrow M = N$$

We conclude this section with two examples of applications of λ-calculus for

computing on natural numbers and to assign a natural number to each λ term.

3.2.1 Church numerals and Gödel numbering

Natural numbers and operations applied to them can be represented in λ-calculus by means of an encoding defined by Church.

Definition 3.2.15 (Church numerals) *Given a natural number $n \in \mathbb{N}$, we encode it in λ-calculus as:*

$$[n]_{Church} = \lambda sz.s^n z$$

Where s^n means n times the application of s. As an example, the following is the encoding of three:

$$[3]_{Church} = \lambda sz.s(s(sz))$$

Definition 3.2.16 (Operations on Church numerals) *The following operations can be defined within Church numerals:*

successor $\mathbf{S} \equiv \lambda nfx.f(nfx)$

addition $\lambda m.m\mathbf{S}$

multiplication $\lambda mnf.n(mf)$

exponentiation $\lambda mn.nm$

An interesting morphism from λ terms to natural numbers is given by Gödel numbering.

Definition 3.2.17 (Gödel numbering of λ terms [Chu36]) *By using traditional techniques, each λ term can be assigned a natural number. Consider an injective function $h()$ from symbols of λ terms to naturals; an example of such a function is: $h(\lambda) = 1$, $h(() = 2$, $h()) = 3$ and for each variable in $V = \{v_1, v_2, \dots\}$, take $h(v_i) = i + 3$. We use $p_1 = 2, p_2 = 3, p_3 = 5, \dots$ for denoting the ordered prime numbers. Now, if the term is composed of the symbols $c_1 c_2 \dots c_n$, its corresponding Gödel number is:*

$$\prod_{i=1}^{n} p_i^{h(c_i)}$$

It is important to notice that these two morphisms are completely independent and it (usually) does not make sense to combine them. Fig.3.1 shows the relations between the two morphisms.

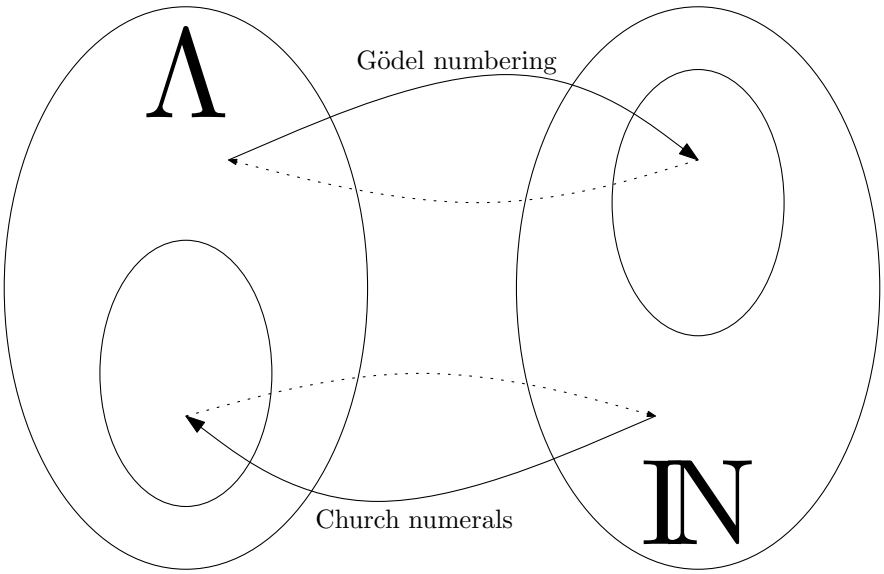

Figure 3.1: Gödel numbering is a monomorphism from lambda terms to natural numbers. Church numerals define a monomorphism from natural numbers to lambda terms.

3.3 The π-calculus

The focus of π-calculus is on processes and their interactions; in particular, it captures input and output behaviour between processes. Processes can progress in their status by performing actions defined by *capabilities*.

Definition 3.3.1 (Capabilities) *Given an alphabet of names, traditionally written as lowercase letters:*

$$N = \{n_1, n_2, n_3, \dots\}$$

the capabilities are defined by the following grammar:

$$\begin{array}{rcl} \pi & ::= & \overline{x}y \\ & | & x(z) \\ & | & \tau \\ & | & [x = y]\pi \end{array}$$

Informally, this is the meaning of the various capabilities:

$\overline{x}y$ send name y through channel x;

$x(z)$ receive a name through channel x and use it in place of every occurrence of z;

τ the capability is not observable (internal, without input/output);

$[x = y]\pi$ if the name x and y are the same then the capability is π.

Definition 3.3.2 (π-calculus) Processes P *and* summations M *are mutually recursively defined as:*

$$\begin{aligned} P &::= M \\ &\mid P|P' \\ &\mid \boldsymbol{\nu} z P \\ &\mid\ !P \end{aligned}$$

$$\begin{aligned} M &::= \mathbf{0} \\ &\mid \pi.P \\ &\mid M + M' \end{aligned}$$

Informally, this is the semantic interpretation of processes:

inaction, $\mathbf{0}$ − is the process that does not do anything.

prefix, $\pi.P$ − must perform the capability π (as described above) before evolving into P. For example $x(z).\overline{(y)}z.\mathbf{0}$ gets a name through channel x, passes it on through channel y and stops.

sum, $P + P'$ − the capabilities of this process is given by the capabilities of both P and P'; once one of the two capabilities is performed, the other part is discarded. For example, $x(u).([u = r]\overline{y}u.\mathbf{0} + \overline{z}u.\mathbf{0})$ receives a name through channel x and will send it through z if it is not equal to r; if it is equal to r it could send it through y (but also through z).

composition, $P|P'$ − the two components P and P' proceed independently and can interact. For instance, $\overline{x}y.\mathbf{0}|x(u).\overline{u}v.\mathbf{0}|\overline{x}z.\mathbf{0}$ can evolve into $\mathbf{0}|\overline{y}v.\mathbf{0}|\overline{x}z$ or, alternatively, into $\overline{x}y.\mathbf{0}|\overline{z}v.\mathbf{0}|\mathbf{0}$.

restriction, $\boldsymbol{\nu} z P$ − creates a new name z whose scope is restricted to P.

replication, $!P$ − is an infinite composition of the kind $P|P|P|\ldots$. For example, $!x(u).\overline{y}u.\mathbf{0}$ is an echo server that receives through channel x and echoes back through channel y.

Definition 3.3.3 (free names) *Given a process P, the set $\mathsf{fn}(P)$ of the free names of P is defined inductively as:*

$$\begin{array}{rclcrcl}
\mathsf{fn}(\tau) & = & \emptyset & & \mathsf{fn}(\pi.P) & = & \mathsf{fn}(\pi) \cup \mathsf{fn}(P) \\
\mathsf{fn}(\overline{x}y) & = & \{x, y\} & & \mathsf{fn}(P + Q) & = & \mathsf{fn}(P) \cup \mathsf{fn}(Q) \\
\mathsf{fn}(x(z)) & = & \{x\} & & \mathsf{fn}(P|Q) & = & \mathsf{fn}(P) \cup \mathsf{fn}(Q) \\
\mathsf{fn}([x = y]\pi) & = & \{x, y\} \cup \mathsf{fn}(\pi) & & \mathsf{fn}(\boldsymbol{\nu} zP) & = & \mathsf{fn}(P) \setminus \{z\} \\
\mathsf{fn}(\mathbf{0}) & = & \emptyset & & \mathsf{fn}(!P) & = & \mathsf{fn}(P)
\end{array}$$

A name that is not free is said to be bound.

We have to define syntactic substitution to let processes use names that they receive.

Definition 3.3.4 (Substitution) *A* substitution *is a function on names $\sigma() : \mathbb{N} \to \mathbb{N}$ that is the identity except on a finite set.*

We write $\{^{y_1,\ldots,y_n}/_{x_1,\ldots,x_n}\}$ to mean the substitution of each occurrence of x_i with y_i. To solve the problem of avoiding capture of bound names, we define α-convertibility.

Definition 3.3.5 (α-convertibility)

(1) If $w \in \mathsf{fn}(P)$, then $P\{^w/_z\}$ is the process obtained by replacing each free occurrence of z in P by w;

(2) A change of bound names *in a process $x(z).P$ is the substitution $x(w).Q\{^w/_z\}$, where w does not occur in Q.*

(3) A change of bound names *in a process $\boldsymbol{\nu} zQ$ is the substitution $\boldsymbol{\nu} wQ\{^w/_z\}$, where w does not occur in Q.*

α-equivalence between P and Q is defined as the possibility of obtaining Q from P with a finite number of changes of bound names.

Definition 3.3.6 (Structural congruence) *Structural congruence is the smallest*

equivalence relation satysfying:

$$
\begin{array}{rcll}
[x = x]\pi.P & \equiv & \pi.P & (\textsc{Sc-mat}) \\
M_1 + (M_2 + M_3) & \equiv & (M_1 + M_2) + M_3 & (\textsc{Sc-sum-assoc}) \\
M_1 + M_2 & \equiv & M_2 + M_1 & (\textsc{Sc-sum-comm}) \\
M + \mathbf{0} & \equiv & M & (\textsc{Sc-sum-inact}) \\
P_1|(P_2|P_3) & \equiv & (P_1|P_2)|P_3 & (\textsc{Sc-comp-assoc}) \\
P_1|P_2 & \equiv & P_2|P_1 & (\textsc{Sc-comp-comm}) \\
P|\mathbf{0} & \equiv & P & (\textsc{Sc-comp-inact}) \\
\boldsymbol{\nu} z \boldsymbol{\nu} w P & \equiv & \boldsymbol{\nu} w \boldsymbol{\nu} z P & (\textsc{Sc-res}) \\
\boldsymbol{\nu} z \mathbf{0} & \equiv & \mathbf{0} & (\textsc{Sc-res-inact}) \\
\boldsymbol{\nu} z(P_1\ P_2) & \equiv & P_1|\boldsymbol{\nu} z P_2, \textit{ if } z \notin \mathsf{fn}(P_1) & (\textsc{Sc-res-comp}) \\
!P & \equiv & P|!P & (\textsc{Sc-rep})
\end{array}
$$

Definition 3.3.7 (Reduction relation) *The fundamental reduction over processed is defined by structural induction as:*

$$\frac{}{(\overline{x}y.P_1 + M_1)|(x(z).P_2 + M_2) \rightarrow P_1|P_2\{{}^{y}/_{z}\}}\ \textsc{R-inter}$$

$$\frac{}{\tau.P + M \rightarrow P}\ \textsc{R-tau} \qquad \frac{P_1 \equiv P_2 \rightarrow P_2' \equiv P_1'}{P_1 \rightarrow P_1'}\ \textsc{R-struct}$$

$$\frac{P_1 \rightarrow P_1'}{P_1|P_2 \rightarrow P_1'|P_2'}\ \textsc{R-par} \qquad \frac{P_1 \rightarrow P_1'}{\boldsymbol{\nu} z P \rightarrow \boldsymbol{\nu} z P'}\ \textsc{R-res}$$

Definition 3.3.8 *Polyadic π-calculus A common extension to the syntax that does not need any extension in the calculus is to add the capabilities to send and receive several names (or also zero). The resulting calculus, although equivalent, is commonly referred to as* polyadic π-calculus. *Sending and receiving multiple names is defined as:*

$$
\begin{array}{rcl}
\overline{x}\langle y_1, \ldots, y_n\rangle.P & = & (\boldsymbol{\nu} p)\overline{x}p.\overline{p}.y_1. \ldots \overline{p}.y_n.P \quad \textit{with } p \notin \mathsf{fn}(P) \\
x(y_1, \ldots, y_n).P & = & x(p).p(y_1). \ldots p(y_n).Q \quad \textit{with } p \notin \mathsf{fn}(P)
\end{array}
$$

3.3.1 Numerals and arithmetic

Numerals and arithmetic operations on them can be codified in π-calculus processes [Mil93].

Definition 3.3.9 (Milner numerals) *Given a natural number $n \in \mathbb{N}$, we encode it in λ-calculus as:*

$$[n]_{Milner} = (\overline{x}\langle\rangle.)^n\overline{z}\langle\rangle$$

Where $(\overline{x}\langle\rangle.)^n$ means n times the capability of sending no data through channel x. As an example, the following is the encoding of three:

$$[3]_{Milner} = \overline{x}\langle\rangle.\overline{x}\langle\rangle.\overline{x}\langle\rangle.\overline{z}\langle\rangle$$

Definition 3.3.10 (Operations on Milner numerals) *To define arithmetic over Milner numerals we need first to parametrize the transformation on the used names and to define a* copy *operation, as once a number representation is used, it is consumed (it is* ephemeral*). The parametrized version of Milner numerals is:*

$$[n]_{Milner}(xz) = (\overline{x}.)^n\overline{z}$$

The copy operation is defined together with the successor operation by mutual recursion as:

$$\begin{aligned} Copy(xz, yw) &::= x.Succ(xz, yw) + z.\overline{w} \\ Succ(xz, yw) &::= \overline{y}.Copy(xz, yw) \end{aligned}$$

Now addition can be defined as:

$$Add(x_1z_1, x_2z_2, yw) ::= x_1.\overline{y}.Add(x_1z_1, x_2z_2, yw) + z_1.Copy(x2_z2, yw)$$

Also operations on booleans can be defined.

Definition 3.3.11 (Milner booleans) *The basic* true *and* false *are encoded as:*

$$\begin{aligned} True(b) &::= !b(t, f).\overline{t}\langle\rangle \\ False(b) &::= !b(t, f).\overline{f}\langle\rangle \end{aligned}$$

Testing a boolean can then be defined as:

$$Test(b, P, Q) ::= (\boldsymbol{\nu}t)(\boldsymbol{\nu}f)\overline{b}\langle t, f\rangle.(t().P|f().Q)$$

With this definition, $Trueb|\,Test(b, P, Q)$ proceeds as process P and $Falseb|\,Test(b, P, Q)$ proceeds as process Q, allowing to use if-then-else constructs.

Another interesting construct present in the literature for π-calculus allow to encode lists, but we will not present it here [Mil93].

3.4 Bridging the two calculi

While the focus of λ-calculus is on functions, the focus of π-calculus is on the communications. The main entity of λ-calculus are functions, described by the input-output behaviour, while in π-calculus they are processes, described by the communication channels where they do input output.

It is interesting to notice how λ-calculus treats data and programs equally (lambda terms), the same way that for π-calculus channels and data passed through channels are exactly the same (the names).

The symmetries between the two main reductions in the calculi are striking:

$$(\lambda x.M)N \longrightarrow M[x ::= N] \qquad p(x).P \mid \overline{p}a.Q \longrightarrow P\{^{a}/_{x}\} \mid Q$$

In λ-calculus the main reduction models the application of parameters to a function; in π-calculus it models the communication between two processes (interaction). It is striking also how in the first case it is a syntactical substitution on variables, while on the other it is on names. This causes two very important clash points:

Sequentiality against parallelism In λ-calculus sequential computations are captured. In particular, there is no possibility to represent the *parallel or* function:

$$\mathsf{Por}(M,N) = \begin{cases} (\lambda xy.x) & \text{if } M \text{ or } N \ \beta\text{-reduces to } (\lambda xy.x) \\ \uparrow & \text{otherwise} \end{cases}$$

As, on the contrary, π-calculus captures parallel computation, it is straightforward to define the function Por with it.

Confluence against divergence If a term in λ-calculus has two different reductions, there always exist a common term that can be obtained by further reducing these two. The implication is that every finite reduction path of λ-terms gives the same normal form. In particular, the leftmost reduction strategy always guarantees to find such normal form (see Def. 3.2.9). In π-calculus, two different reduction paths lead to programs that behave differently, for the

symmetrical property of the reduction (one process sends, one receives). A reduction path in a π-calculus process can influence all of the terms operating on a certain name; in λ-calculus it does not influence any other redex.

In spite of these differences, λ-calculus can be interpreted in π-calculus, in several of its reduction strategies that are actually used by programming language implementations (call-by-name, call-by-value, call-by-need) and even in several of its typed versions (simply typed, with subtyping and with recursive types). The direct implication of the possibility of interpreting λ-calculus is that π-calculus is Turing-equivalent. On the contrary, as we already mentioned, some functions like *parallel or* cannot be expressed in *lambda*-calculus; in particular *lambda*-calculus cannot interpret *pi*-calculus.

The independence of the two calculi has been accepted without particular concerns from the computer science community, as they were seen as orthogonal, modelling two different aspects of computation. But in more recent times, the Church-Turing hypothesis itself (or at least in the interpretation is has been commonly given) has been challenged [Weg98, WG03, GW05]. The claim is that *lambda*-calculus, Turing machines and recursive functions are able to model only algorithms and are just a theory of functions but they do not define everything that is computable. The supporters of this theory claim that it was not Church's nor Turing's original idea to define everything that was computable with, respectively, λ-calculus and Turing machines but this was a later interpretation of their seminal works.

It is clear that modern distributed computing needs a new notion of computability taking into account interactions and the environment. Several researchers acknowledged this and came up independently with different models of computation that are all equivalent (see Fig. 3.2). The aim of these models is to go beyond the algorithmic definition of computable by taking into account the possibility of performing interaction in the middle of an algorithm execution, with the possibility of influencing the execution itself.

These new models of computation are seen as adding a new dimension to distributed and parallel models (see Fig. 3.3): in distributed models in fact, computation is just separated in space, but keeps its algorithmic aspect; the same way, in parallel models computation is just simultaneous in time. Interactive models on the contrary take into account external inputs that can change the computation path.

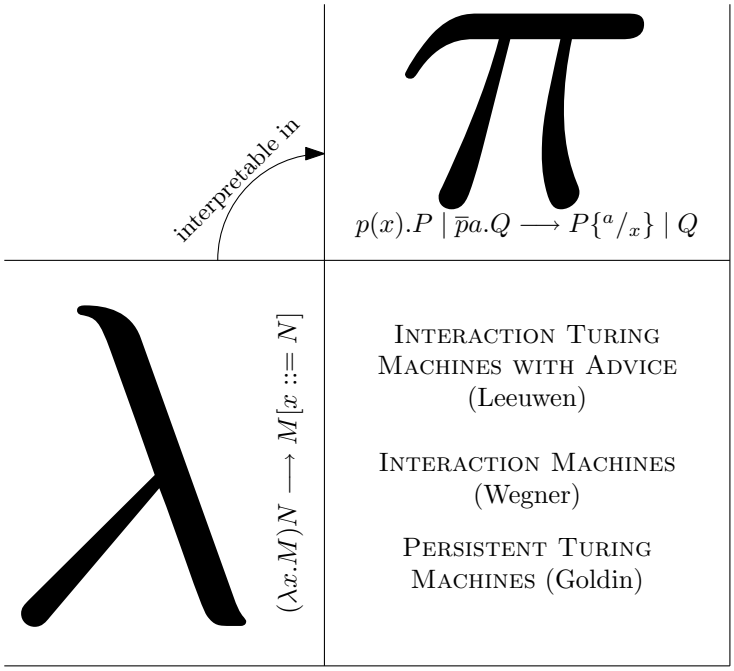

Figure 3.2: Models of computation: λ-calculus and π-calculus are orthogonal, one modelling the behaviour of functions and the other interaction among processes. Several new models have been defined in the last years to bridge the two calculi (see text for references).

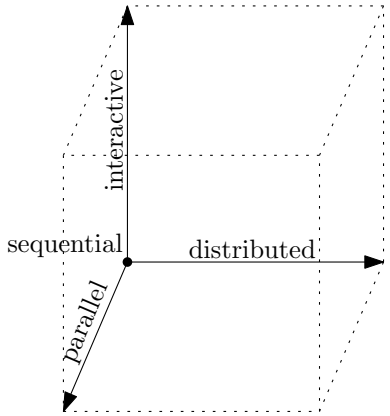

Figure 3.3: View on models of computation, adapted from Wegner [Weg98]. Distributed and parallel models do not add expressiveness, while interactive computation adds a new dimension.

3.5 Applications

The same way that λ-calculus set the ground for a new family of programming languages (functional, like LISP, Haskell, Caml) that are being widely adopted by the industry for their support of robust programming, π-calculus is starting to be adopted as the foundational theory for distributed systems. The direct outcome is the creation of a Service Oriented Architecture infrastructure based on it, namely "Pi Calculus for SOA"[1,2]. The specification underlying this implementation is the Web Services Choreography Description Language (WS-CDL) and is backed by the World Wide Web consortium [KBR+05]. The innovative approach over other choreography languages is the addition of communication channels, that can then be shared and passed among processes.

WS-CDL is going to include other World Wide Web consortium standards (for example Web Service Choreography Interface) and is expected to become the standard for modelling choreography of web services [AAF+02].

[1] `http://sourceforge.net/projects/pi4soa/`
[2] `http://www.pi4tech.com/`

Chapter 4

Biologically Inspired Programming

In Nature's infinite book of secrecy / A little I can read.
(William Shakespeare, 1564–1616)

The availability of large amounts of biological data, in particular since the discovery of Sanger sequencing in 1975, asked for the intervention of automatic treatment. Unfortunately, the problems to be solved were not simple at all, besides the fact that there were huge instances of them to solve. Computer science suddenly was flooded with new problems (some reducible to known ones, some totally new) that required new complexity studies, algorithms and practical implementations. At the same time, nature-inspired optimization and architectures were invading the rigorous computer engineering world.

Very soon, computer scientist started modelling and studying biological problems while biologist were implementing heuristics for crunching their large amounts of data. The two disciplines became so intertwined that we have now the new fields of bioinformatics, computational biology, computational chemistry, computational system biology and bio-computing. The amount of modeled data in different formats that are now available is so high that there has been the need to meta-model the biological information for transforming among different model formats [RRS05a].

We present in this chapter how computational modelling tools served the understanding of biological systems and propose to re-apply these same adapted models again to processes description for a better understanding of complex systems.

4.1 Molecules as processes

Computer scientists trying to contribute to the modelling and understanding of (bio)molecular systems soon discovered to have an extremely powerful advantage over biologists: years of research in computer processes description.

The models needed for system biology can be summarized as [Car05a]:

Protein machine Models the interaction among proteins and is thus close to the chemical structure (atoms and chemical compounds). The related networks are called biochemical networks.

Membrane machine Models the interactions among cells. The related networks are called transport networks.

Gene machine Models the genes expression, in particular in the interactions that create specific phenotypes. The related networks are called gene regulation networks.

The protein abstract machine is the "easiest" to model, as it can benefit a lot from the experience of computational process calculi. Some of the calculi that were developed to model it include:

Stochastic π-calculus It adds different probability distribution to the different composition paths of a π-calculus process to obtain stochastic race instead of non-deterministic choice [PRSS01]. The probabilities are given to the channels and specifz *communication rates*.

BioSPi Models chemical interactions (exchange of electrons and small molecules) as communication between the compounds, again using a stochastic approach. A simulator is provided [RSS01].

Join calculus and Bio-calculus Join calculus adds reflexion to the chemical abstract machine of Berry and Boudol [BB90] to obtain a formal model of concurrency that is consistent with mobility and distribution. It has functional and object-oriented features. It is expressively equivalent to π-calculus [FG96, FGL$^+$96, FLMR97]. Bio-calculus is similar to Join and has complex formation as a primitive [MSSH99].

Pathway logic It is a rewrite system for proteins, where reactions are defined as rewriting rules. Being purely sequential, it misses parallel execution [EKL+02].

κ-calculus Calculus specifically designer for representing and studying biological networks, where complex formation is primitive. It compiles to π-calculus [DL03].

Most of the implementations of these models make use of the well-known Gillespie algorithm for stochastically simulating the chemical kinetics, by computing concentrations and reaction times for biochemical networks [Gil77].

Gene machines and membrane machines are more complicated to model, as they work at a level very far from the single molecules and need different abstractions. The gene machine can be modeled the same way as proteins but they have the following important differences [Car05a]:

- an asynchronous stochastic control
- biologically poorly understood and thus harder to model
- networks present motifs, sort of regular patterns, that are being analyzed

A novel approach more powerful than protein models consists in the use of hybrid petri nets [DFM+04].

Membrane machines on the contrary need a completely different abstraction, due to the different mechanics. As they define transport mechanisms for moving objects among different boundaries, the closest computational model is *ambient calculus*. The following are some of the models defined for abstracting the membrane machine:

P-systems Strongly connected to formal languages and grammars theory [Pău02].

Bioambients An extension of BioSPi (see above) for membranes that models dynamic compartments similarly to the ambient calculus [RPS+04].

Brane calculi A family of calculi for bitonal membrane systems performing computation *on* the membrane [Car05b].

These models for abstracting the machines of system biology could prove extremely useful when re-applied to describe DBE services. If we consider the whole digital ecosystem, the various languages help describe it at different levels, from the change of environments of services (membrane computing), to the interaction patterns of single services, depending on their concentration in the system (molecular computing).

Chapter 5

Software Engineering Implications

> *Einstein argued that there must be simplified explanations of nature, because God is not capricious or arbitrary. No such faith comforts the software engineer.*
> (Frederick P. Brooks Jr., 1931-)

The previous chapters started to outline some of the elements and theoretical frameworks for adaptive service generation by means of automatic code generation. In this chapter we analyze the implications regarding software engineering processes: the changes are relevant from the requirements analysis up to the deployment and certification of software, including the implementation itself.

5.1 On the software specification

It is current software engineering practice to formulate in writing the requirements and high-level specifications for software solutions, where the aim is the achievement of a common understanding of the terminology involved and a clear picture of the functional requirements that at this stage are still far from the technical details.

The resulting documents are then gradually transformed into the technical specification expressed in modelling notations such as UML for software or, to give an example for hardware, in VHDL. At that point, the link between the natural language description and the technical specification is broken and the customer loses the possibility to influence and verify the ongoing work (see Fig. 5.1). A more homogeneous approach that does not break so sharply between the requirements analysis and software design has long been preached [Boy99, SHE89].

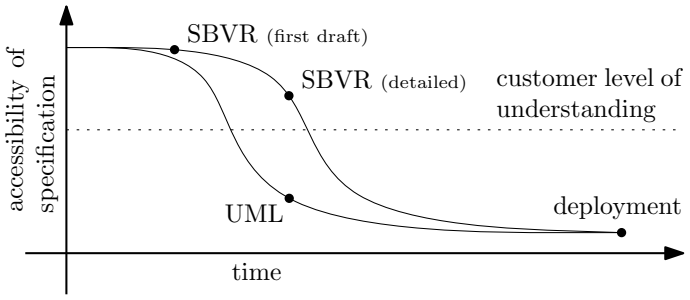

Figure 5.1: Software specification using standard current practices (UML) and natural language based techniques (SBVR). From an initial natural language requirements analysis, current practices drop very soon below the level of understanding by the customer, whereas with natural language software engineering the customer can follow most of the process, thus reducing the "digital divide" in software engineering.

Modelling notations for software are unfortunately required to avoid the ambiguity of natural languages: SBVR, that we already introduced in section 2.1, bridges this gap [OMG06]. A business or process model written in SBVR satisfies the future implementers and the customer, as it is both formal and understandable.

Another possibility given by SBVR is the capability to define vocabularies and rules in different languages (English, Italian, ...), so that the business models can be automatically translated, although at the time of writing only the structured English vocabulary is defined in the specification.

When using SBVR, the customers can easily influence the requirements analysis process until a later stage compared to processes utilizing UML: the development of tools for editing the vocabulary and rules give them independence from the analysts. Currently, at least a commercial[1] and an open-source[2] implementation are being developed for this aim.

5.2 Exploiting service orientation

The whole process of software engineering can assume a completely different facet when performed in the context of the Digital Business Ecosystem, where it would be more properly called *software procurement*, once the customers' requisites are

[1]Unisys Rules Modeler by Unisys Corporation, `http://www.unisys.com/businessrules/`

[2]SBeaVeR - Business Modeller by the eBusiness Management School department of the Institute for Advanced Interdisciplinary Studies at the University of Lecce, `http://sbeaver.sourceforge.net/`

clear enough. In fact, the abundance of services, together with the facilities for finding, orchestrating and processing them in a distributed environment, will make the implementation phase lighter and lighter as it takes advantage of the latest outcomes of research.

In spite of being used in a wide contextual variety, the term "service-orientation" means that logic required to solve a large problem can be better processed if it is decomposed into a collection of smaller related pieces, called services [Erl05].

In service oriented architectures the services are usually composed manually. By adding semantic service descriptions, the user is able to formulate requests or even specifications in a way that (see Fig. 5.2, where the question marks in the figure denote the status "open" for software or service requests):

- a complex request for a service can be divided automatically into single service specifications;
- these services can be searched in a local or distributed service pool;
- the best available service-combination can be assembled; and
- the service requests or specifications which are not available yet can be published for future implementation by other enterprises in the network.

The current implementation of the Digital Business Ecosystem provides only the means for querying BML 1.0 descriptions, local search and manual composition. The following components are thus missing: BML 2.0 querying ("SBVR Recommender", will not be implemented by the DBE project), distributed search ("Distributed Intelligence", being implemented) and automatic composition ("Automatic BPEL Composer", some functionalities being implemented in the DBE). See Fig. 2.1 on page 9 for the relationships among components and the underlying computational infrastructure enabling adaptive service generation.

The decomposition of software specification is a commonly followed approach due to the issues of construction, processing and management of software projects that are too large. The complexity and variety of disciplines involved in today's big projects result in a large effort in risk management and communication among different stakeholders to deal with rising intricacy. Even if small and medium enterprises could increase their manpower to the necessary number, they are often unable

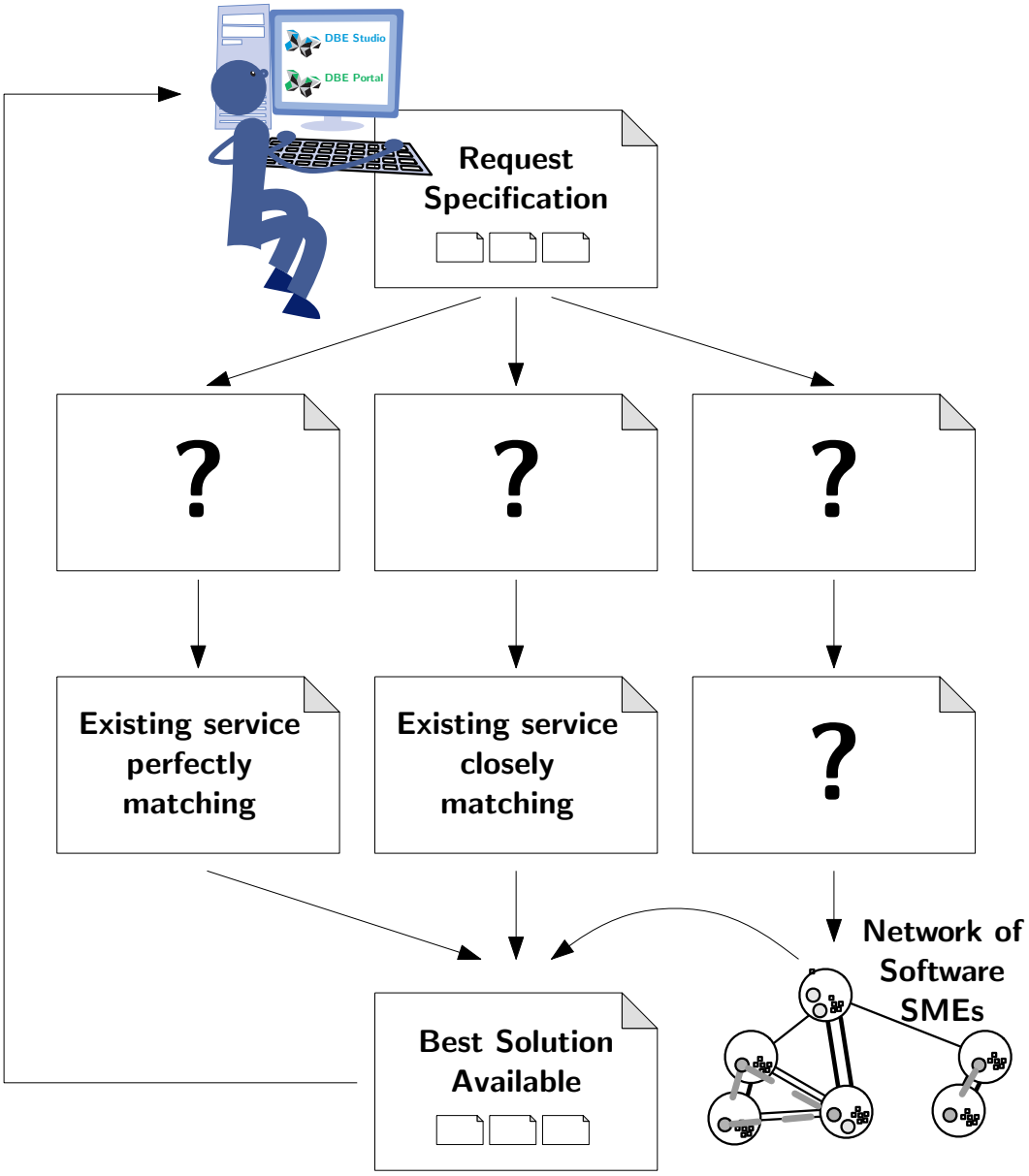

Figure 5.2: Service oriented approach in the Digital Business Ecosystem: the user can express his needs via the available interfaces (DBE Studio for developers and DBE Portal for other users). These needs are a request and a software specification at the same time that can be decomposed in units of functionality. Services matching these functionalities are found in the system, while missing ones are provided by software implementors: the final software solution is created by composing all them.

to deal with the new challenges of larger projects and get overwhelmed by communication and quality assurance issues [HK04]. In the same way that bigger companies have reacted in overcoming the increasing complexities of software development by creating small agile units to deal with different parts of the system and super-units to synchronize them, small and medium enterprises can distribute the share of risk when collaborating in larger projects by using a collaborative environment.

The automotive industry is a well-suited example when considering the complexity of current systems. Concerning risk management, the Failure Mode and Effects Analysis is one of the state-of-the-art and most extensively used methodologies in manufacturing, especially in the automotive sector. The rising importance of software in the automotive sector also resulted in initiatives to adopt methods for the new needs of such interdisciplinary projects [KHH04]. The first and constitutive step in complex projects like car production is a decomposition of the complex product in *system elements* and a deep knowledge about the processes involved. As bigger companies decompose their products for evaluating and distributing the share of risk, the cooperation of fine-grained services provided by smaller enterprises are in the best place to do that risk segmentation automatically. Nevertheless, the processes of interaction and composition of services in this sector are still an open issue.

In service oriented architectures there are several strategies to overcome the issue of service composition: besides the definition of interfaces, strategies have been developed to model also processes of services [Erl05]. Risk management activities as well as testing can be done for each single component or service. Nevertheless, most of the integration work is still done manually. The next logical step is the automatic composition of services to larger components without knowledge of the inner implementation of the functionalities but just a guarantee at the specification level of the behaviour of single services. The following section explains how this could be achieved.

5.2.1 Extensional service description

To be able to orchestrate services, a pure technical description is not sufficient. Even in the case of manual composition, the human user has to know the meaning of all the input and output parameters. These are commonly described in natural language in a document such as the user manual or the technical documentation.

Additionally, data in output from a component can undergo a transformation before being sent as input to another component. It is clear that for achieving automatic composition we must have access to all this information through a semantically rich description.

The approach of languages like OWL/OWL-S is only to annotate the technical interfaces with semantic information, that is, to tag input and output parameters and to give preconditions with meaning [BvHH+04, MBD+06]. This unfortunately could not be enough as it does not allow modelling of stateful services but only stateless ones.

Languages like *Executable UML* are not suitable for our purposes as they provide a full intensional specification of the services, up to the level of defining the automaton performing the operations [OMG05]. This is too close to the actual code and is thus not a feasible way, besides the fact that it loses the level of abstraction needed to keep automatic composition within a dimension that could be dealt with.

Here again SBVR can help: business rules define services *extensionally*, as a rule could actually be implemented in several different ways (it corresponds to all different implementations of logical formulas equivalent to the given one). The description of rules in SBVR is then very similar to the description of a process in π-calculus: it details how the services react to external messages, specifying its communication patterns, but does not expose the internal machinery. Note that the definition of an operative business rule does not also specify the way in which the business rule is enforced.

Still, this description is rich enough to allow the automatic generation of code at the two fundamental levels:

- the glue code needed to orchestrate the services in a particular context (adaptive service generation) and
- the code to enforce the rule accordingly to the environment where it has to be deployed (automatic code generation).

The approach being pushed by OMG through MDA and the role of SBVR are a very promising starting point for solving the issues of automatic requirements decomposition and automatic service composition. Nevertheless, many issues are still left open and technologies are not sufficiently mature for being fully deployed in real-world large-scale projects (see Chapter 6 for the challenges in doing this).

5.3 Resynchronizing the specification

There are several reasons for maintenance of software: for implementing requirements that were missing in the original analysis; for the correction of incorrect behaviours; or to adapt to changes in the organization and/or processes. These adaptations of the process rarely pass through the whole software engineering process, as most of the changes that are not cross-cutting are implemented directly at a lower level.

For this reasons the software actually deployed often does not correspond to the original specification anymore; this is true in particular for changes in the orchestration of components. The result is that business models, including business rules descriptions, are not connected anymore to their actual implementation, implying that the *practice* is not anymore in line with the *theory*, with all the related risks that such a separation implies.

Exploiting the capabilities of the digital ecosystem and SBVR, it is possible to automatically synchronize a software specification from modifications done to the annotated software: this is the reverse direction of code generation that we call *model inference*.

Model inference makes sense in particular when modifications are done on the generated code that contains annotations with references to the original model: these modifications can then be used to automatically adapt the model/specification (see Fig. 5.3).

5.4 Generating UML class diagrams from SBVR vocabularies

Current software engineering techniques rely on the interaction between a business analyst and a software modeller. The novel approach introduced by SBVR is fully in line with OMG's Model Driven Architecture and avoids this interaction by giving to the business analyst the power to describe the software requirements by means of natural language. The first step toward automatic generation of software from this description is the transformation of this natural language description into a UML class diagram. We discuss this transformation by screening and selecting the appropriate tools and presenting a rule set for the transformation. We leave the

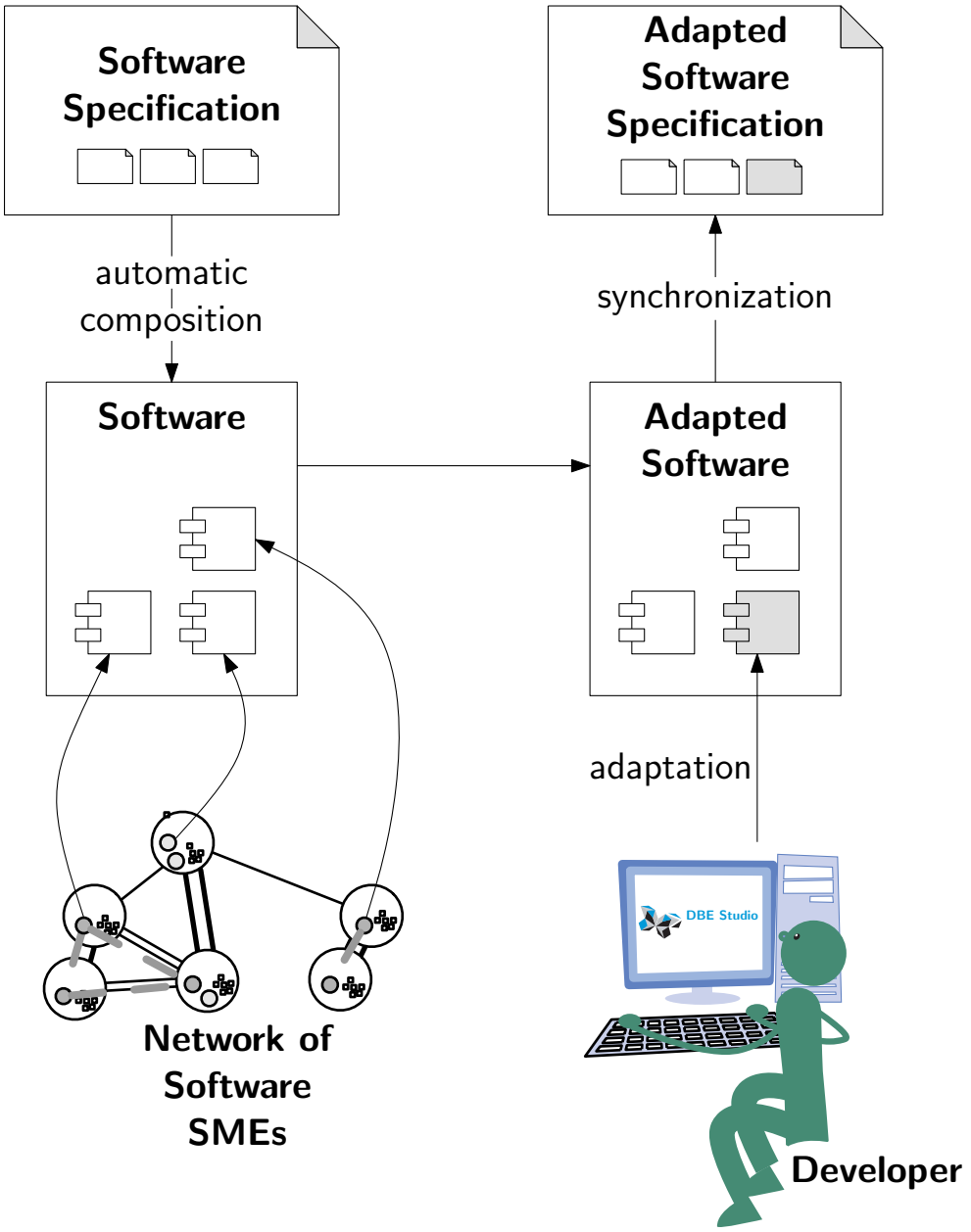

Figure 5.3: Re-adaptation of software specification after change requests are directly implemented.

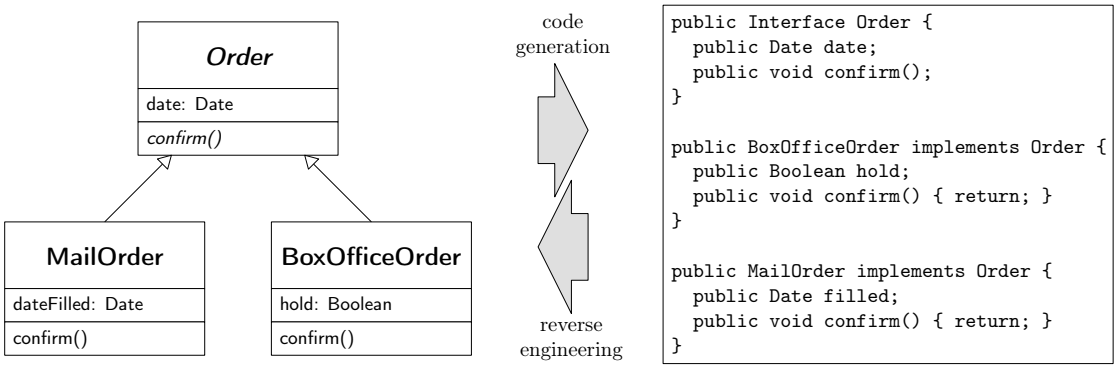

Figure 5.4: Example of a UML class diagram and the corresponding code with both sides of the transformation: code generation (from the model to the code) and reverse engineering (from the code to the model).

implementation of the actual algorithm for the transformation as a future work.

Many UML modelling tools exist and most of them offer functionalities like reverse engineering a code generation: these are the two directions that go from a UML model, usually a class diagram, to the code in some target language, usually object oriented ones and commonly Java/C#/C++ (see Fig. 5.4 for an example of such a transformation). More advanced tools enable other kinds of transformations (for example from sequence diagrams to code including the various method calls) and are able to keep the UML models and the code synchronized using vendor specific tags in the models and particular comments in the code. Some of the most widely adopted tools for this aim are: the IBM Rational[3] suite, No Magic MagicDraw[4] and ArgoUML[5] (open source but still does not support UML 2.0).

As already mentioned, SBVR has a MOF model, so it can be stored into any MOF repository (see Section 2.1). Together with the model, a mapping from an SBVR vocabulary to MOF/XMI is defined as a rule set [OMG06, Chapter 13]. It is then possible to interchange and link to any other model based on MOF, like UML, although this does not mean that we can easily transform from one model to the other, only that they are able to "speak the same language". An approach in transforming SBVR models to UML class diagrams is exactly the subject of this section.

The implementation of the rule set proposed below should be based on existing

[3] http://www.ibm.com/rational/
[4] http://www.magicdraw.com/
[5] http://argouml.tigris.org/

tools for UML modelling (both programmatically and graphically). We suggest the use of Eclipse UML2[6], an implementation of the UML 2.x metamodel for the Eclipse platform based on the Eclipse Modelling Framework, for the following reasons:

- the whole DBE tool chain (DBE Studio) is based on Eclipse and so it makes sense to chose a compatible instrument;
- the support for Ecore[7] exporting enables code generation from the Eclipse Modelling Framework;
- the use of the Graphical Editing Framework (GEF) allows to graphically represent and edit models;
- the possibility of full integration with tools like Omondo EclipseUML[8], of which a free edition is available;
- full UML 2.x support, with XMI 2.x export;
- a big community and a good connection to standards bodies.

We would discourage usage of the Novosoft metadata framework and UML library, although it is widely adopted and used from well-known tools like ArgoUML. The main disadvantage of this library is that it does not support UML 2.x (UML 1.3/XMI 1.0 until ArgoUML 0.18.1, UML 1.4/XMI 1.2 from ArgoUML 0.20); the same ArgoUML project is expected to drop the usage of the library in favor of UML2 very soon.

5.5 Transformation rule set

An SBVR model is composed of two main parts(see Fig.5.5):

- a collection of terms defining meaning;
- a collection of business rules.

[6] `http://www.eclipse.org/uml2/`
[7] EMF Ecore meta model`http://www.eclipse.org/emf/`
[8] `http://www.omondo.com/`

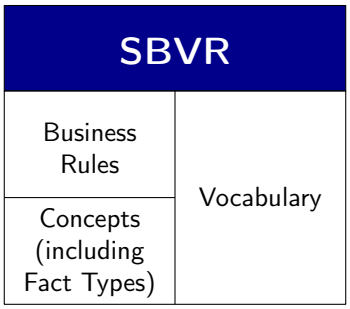

Figure 5.5: An SBVR model consists of two main parts: concepts and business rules. Figure adapted from the SBVR specification [OMG06].

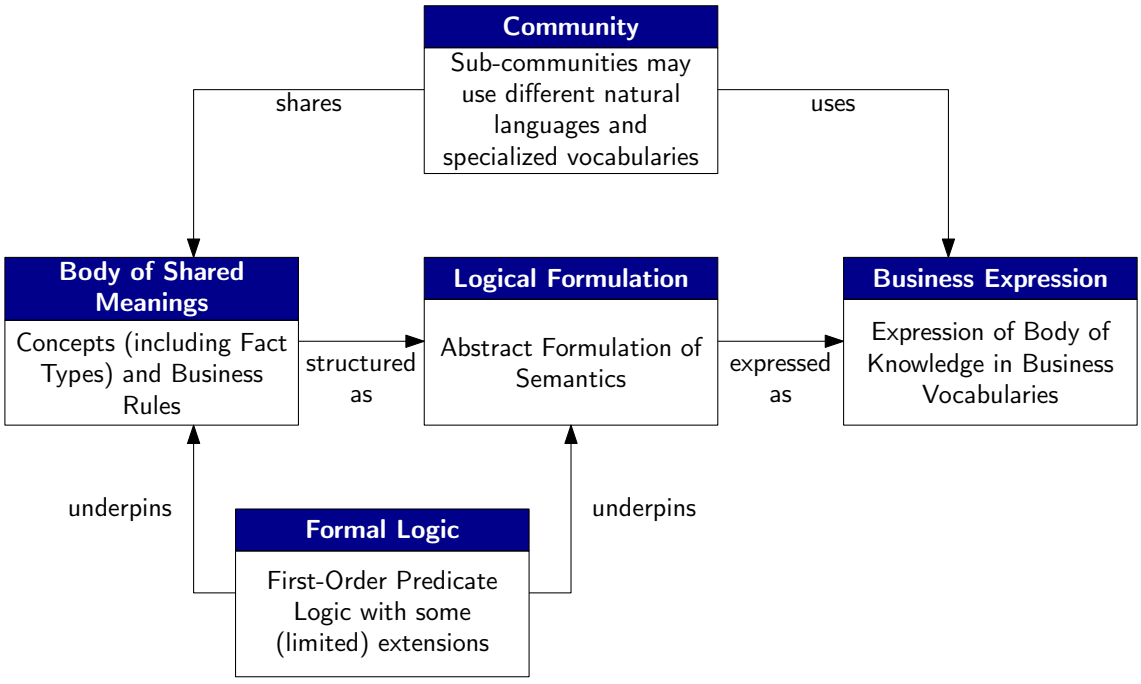

Figure 5.6: Informal overview of SBVR, adapted from the SBVR specification [OMG06].

In an SBVR model there is no distinction between the conceptual schema (the facts structure) and the population (the ground facts); there is then no proper distinction between an instance and a model. The concept of instantiation for model-driven approaches is substituted by the concept of extension in SBVR (see Fig.5.6).

The rule set for the transformation works on the terms (concepts) defined in the vocabulary, taken one by one, and proceeds considering their concept type. We show the rule set here by example; the SBVR model snippets are taken from the EU-Rent Vocabulary example contained in the SBVR specification [OMG06, Appendix D].

Concept Type :: Individual concept (double underlining) These terms are actually objects, and should then be represented in an implementation diagram; in this case they are an instance of the class referred by the General Concept. Alternatively, they can be represented in the class diagram with the use of the singleton pattern, specializing the concept defined as the general concept [GHJV95].

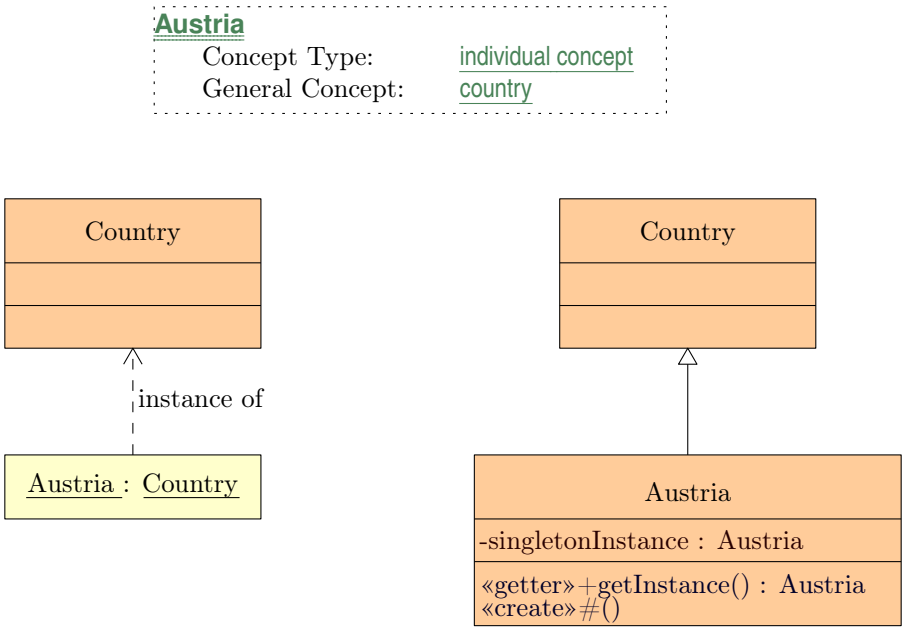

Figure 5.7: Two ways of interpreting an individual concept in a UML class diagram.

Concept Type :: Characteristic A characteristic is also a fact type with only one occurrence of a role (the entity having that characteristic). Characteristics can be expressed as attributes or operations to check the characteristic. Good design requires that the attributes of a class are private, so in any case for

each characteristic we should define a method with the name obtained by concatenating "is" and the characteristic name. This method can then use a private attribute or verify the given rules to compute the proper return value (see Fig. 5.8).

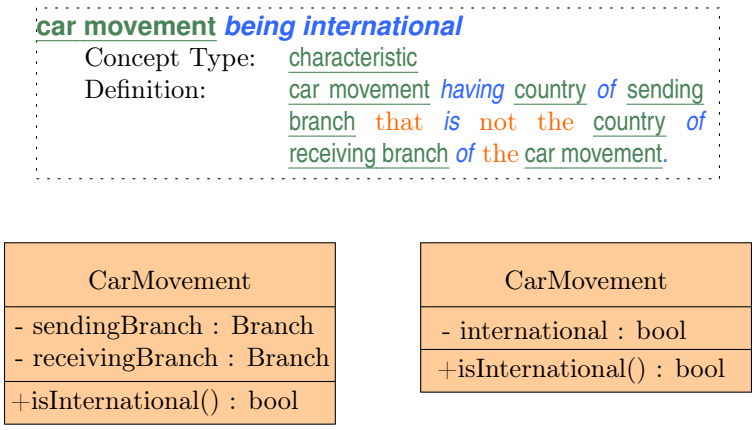

Figure 5.8: Two ways of interpreting a characteristic in a UML class diagram.

Concept Type :: Role Roles of concepts define possible instantiations of classes in UML terms. When a concept has as a concept type "role", we should find all the instances of that concept in the definitions of other concepts and put an attribute in the class defined by that concept that refers to the role with the appropriate name (see Fig. 5.9).

Concept Type :: Binary Fact Type Binary fact types can be of many kinds but they all define a relationship between the two concepts being related. Many relationships are synonymic forms and thus can be modeled in the same way in UML. Two common relations are *has* and *specifies*, that define a strong relation between two classes: depending on the multiplicity, these relations could be modeled with attributes or as associations (see Fig. 5.10 and compare with Fig. 5.9).

Inferring generalization A strong concept in object-oriented modelling is generalization/instantiation. In SBVR, individual concepts must have a reference to a general concept, representing in that case a UML generalization relationship. Unfortunately, it is not always the case that concepts have their general concept defined, also when there is a relationship to another class that natu-

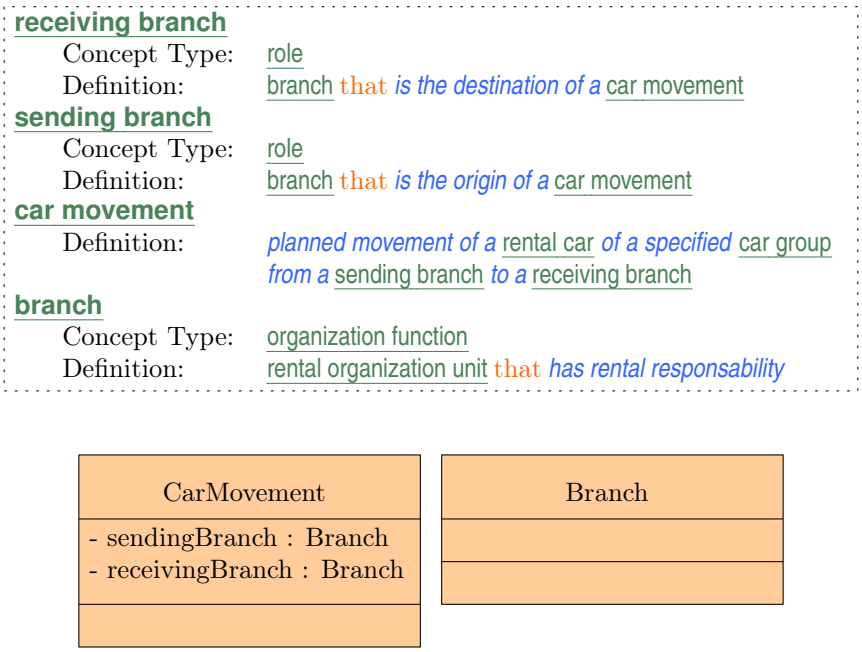

Figure 5.9: Interpreting a role in a UML class diagram. Here both "receiving branch" and "sending branch" are interpreted.

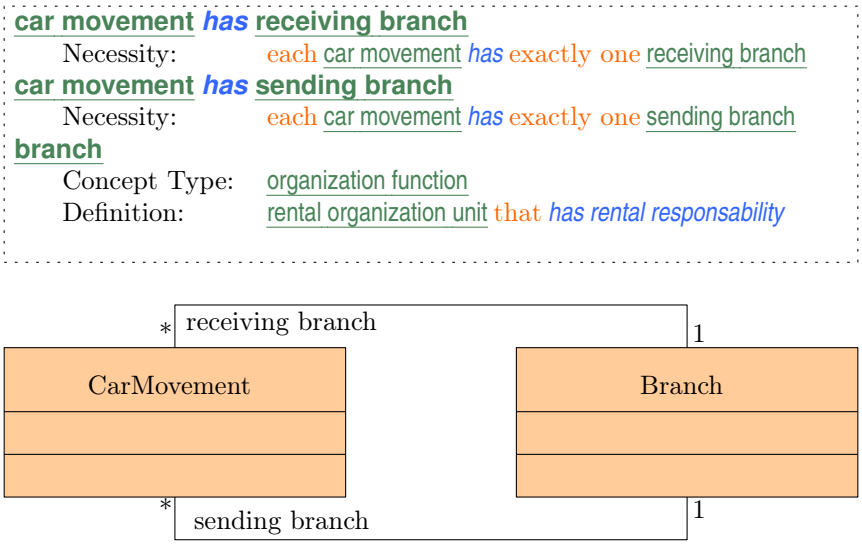

Figure 5.10: Interpreting binary fact types of the kind "has" in a UML class diagram. Here both "car movement has receiving branch" and "car movement has sending branch" are interpreted.

rally maps to a generalization in a UML class diagram. In most of the cases, we can infer the relationship from the definition of the concept (see Fig. 5.11).

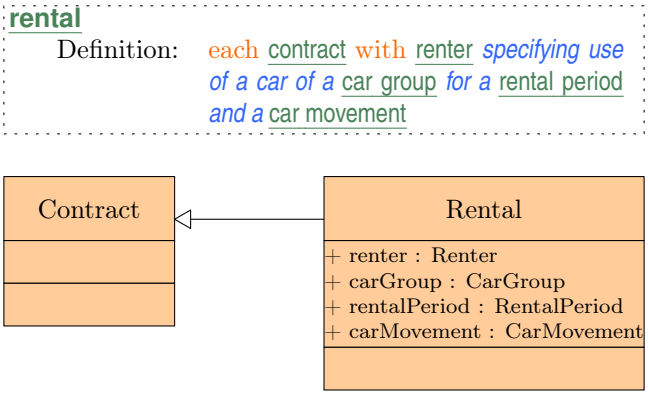

Figure 5.11: Interpreting definitions of concepts as generalization relationships in UML class diagram.

5.6 Future work

We have seen a rule-set for transforming SBVR models into UML class diagrams and suggested the appropriate tools for an implementation.

We have seen several implications of natural language modelling on software engineering. The research in these implications has to go further, as well as the research itself should flow into the appropriate tools.

We propose the following points as future work:

Implementation of UML class diagrams generator from SBVR models In this Chapter we have already presented an implementable rule-set together with suggestions of the appropriate libraries and tools to use for the implementation.

Transformation of business rules The semantic richness of SBVR business rules should allow to transform them into xUML programs or UML activity diagram. Some steps toward this direction have been done but further research is needed [Sch06].

Systematic approach for transformations Transformations from SBVR models to other models more common to software engineers can be done consistently through a systematic approach. The models involved in the proposed

transformations have all a MOF model. Transformation among MOF models can be defined formally by means of a suitable language like ATLAS[9] [JK05]. ATLAS has been developed in response to an OMG request for proposals on query/views/transformations in the model driver architecture and allows to formally define transformations among MOF models.

The next chapter presents some computational problems that have to be efficiently solved to enable what was addressed here.

[9] http://www.sciences.univ-nantes.fr/lina/atl/atlProject/

Chapter 6

Computer Science Challenges

Stat rosa pristina nomine, nomina nuda tenemus.
(U. Eco, 1932–)

In the context of the optimization of services combination, two very difficult computational problems arise: the Set Covering Problem and Automated Theorem Proving. In this chapter we explain where and why the problems arise, their complexity and our approaches in dealing with them. The Set Covering Problem results are an outcome of the collaboration with John Woodward (UBham) and Jonathan Rowe (UBham). The approach for Automated Theorem Proving has been developed together with Maurizio de Tommasi (ISUFI).

6.1 Efficiently Solving Set Covering

The classical *set covering problem* (SCP) is an NP-hard optimization problem typically used to model resource selection problems. An instance of the SCP consists of a set M containing m elements, and a set N of size n consisting of subsets of the elements in set M. Each element in M must be included in at least one subset found in N. A *cover* is a subset $C \subseteq N$ where every element in M is included in at least one or more of the subsets found in C. Thus $M = \bigcup_{S \in C} S$ which is equivalent in saying that all elements in M belong to at least one subset found in C or that all elements found in the subsets in C are in M. The objective of most *set covering problems* are to either:

- find a minimum cover where the cardinality of C is the smallest of all possible covers, or

- in the case that each subset in the set N has an associated cost, find a minimum cover where the total cost of all subsets used is the smallest.

For the DBE project, each subset will be assumed to have a fixed unitary cost and thus the objective would be to find a minimum cover with the smallest $|C|$.

6.1.1 An abstraction of services composition

In the case of the DBE project, the SCP is applied to the problem of finding the minimum number of services that includes all features requested by a *user request*. Since not all features are to be found, this will be referred to as the *partial set covering problem*. The partial SCP is a relaxed variant of the SCP where the cover C is only required to include at least a small subset, the *request* $R \subseteq M$, of all the elements in M. Thus a valid cover must satisfy:

$$R \subseteq \bigcup_{S \in C} S \subseteq M$$

For implementation purposes, the SCP instance will be represented in matrix and vector format. An instance of the SCP consists of a zero-one $m \times n$ matrix A with m rows representing elements or features, n columns representing subsets or services, and a zero-one vector C of size n representing which columns are included in the cover. An entry a_{ij} in the matrix A will be 1 if the feature i is offered by the service j and 0 otherwise. An element c_j in the vector C will similarly be set to 1 if the service i is included in the cover and 0 otherwise. The request is represented by the zero-one vector R of size m. An element r_i in the vector R will be 1 if there is a request for the feature i and 0 otherwise. The cover must then be subject to

$$\sum_{j=1}^{n} a_{ij} c_j \geq r_j \qquad i = 1, \ldots, m$$

where in the classical SCP,

$$\sum_{j=1}^{n} a_{ij} c_j \geq 1 \qquad i = 1, \ldots, m$$

A simple and naïve reduction can be performed to transform an instance of the partial SCP to the classical SCP where for every 0 entry in the request vector R,

the rows of the corresponding feature in both R and A be removed. This can cause some service columns of A to be all zeroes and thus the entire column can also be removed, shrinking A to a new $n' \times m'$ matrix A'. It will be assumed that this reduction step will be performed in the beginning and thus the sizes of n, m, and A will refer to the reduced form and the classical SCP is to be solved.

We will present two approaches to solving or approximating the SCP: through the *Univariate Marginal Distribution Algorithm* (UMDA) and through the *Genetic Algorithm* (GA). Both will use similar data structures (matrices and vectors) and formulas (fitness functions). Thus a testbench for the two methods will be constructed to analyze their performances to see which is more suitable for the particular instances considered.

6.1.2 Literature

The *Set Covering Problem* (SCP) is a well-known model for many real world problems, among others on location, distribution and scheduling. It represents the problem of covering the rows of a zero-one matrix by a subset of columns with minimal cost.

The problem is the dual of Hitting set, and has Vertex Cover as a special instance. When an element can be covered more than once, the problem is known as Set Packing.

Although being conceptually rather simple, SCP is known to be NP-hard in the strong sense [GJ79], its decision version being NP-complete by reduction from Vertex Cover [Kar72]. More recent results show also hardness of approximability and a still unpublished work closes the gap between the theory and the best approximation reached by known algorithms [LY94, Fei98, AMS04].

The most common operations research approach is to use linear programming relaxation and additionally Langrangian relaxation combined with subgradient optimization [CNS98] or alternatively surrogate relaxation [LL94].

Although most of the algorithms for SCP are heuristic, some exact algorithms exist, mainly with a branch-and-bound approach [FK90]. In particular, also an exact algorithms using subgradient optimization has been developed [BC96a].

The classical approximation approach is the greedy algorithm [Chv79, GHY93], that is a $\ln n$-approximation algorithm. The most recent theoretical result says that the problem is not polynomially approximable within a ratio of $c \ln n$, where c is a

constant [AMS04] and thus this approximation algorithm is optimal.

Other approaches to SCP include simulated annealing [BJT99, JB95, Sen93], mean field annealing [AH95], genetic algorithms [BC96b], indirect genetic algorithms [Aic02], parallel genetic algorithms [CP98], neural networks [GW97], ant colony optimization [AK00], meta-heuristics [LK04], 3-flip local search [YKI03] and hybrid approaches mixing several techniques [EKZ00, BC96a, BJ92].

It is possible to find several surveys on the SCP problem, comparing the theoretical, experimental and computational results [CFT00, GMPV05b, GMPV05a].

Variations of the SCP that can be considered in the context of the DBE are the Online SCP, where the columns of the matrix are given one-by-one [AMS04] and the Probabilistic SCP, where not all the rows have to be covered [BR01].

6.1.3 A genetic algorithm for the SCP

A *Genetic Algorithm* (GA) is an algorithm which mimics the evolutionary process of organisms in nature to intelligently guess solutions of combinatorial problems [BC96b]. Within a population of a typical organism, each individual within the population will have its own unique DNA sequence. Depending on what combination of genes each individual has in its DNA sequence, individuals will have varying levels of fitness in relation to one another within the population. Those that are more fit will be more likely to mate and pass on their DNA to the next generation while those that are less fit will not. Offspring will have a combination of DNA from their parents with the crossover process and thus can be more or less fit then their parents. They will again be subject to the fitness test of the generation where the cycle repeats. Random mutations also exist to introduce new genes or remove genes from the population and those undergone mutation will be subject to the test of the fitness function. As the generations progress, it will become likely that the population as a whole will shift wo becoming more fit in relation to their surroundings.

Genetic Algorithms use this concept by using a population of solutions or individuals where each solution has its own combination of services or DNA sequence. For the purpose of the DBE project, a solution (each a potential cover) is a represented by a vector indicating which services are used, and a population P which is a set of solutions of size p. Each solution vector C_k, where $k = 1, \ldots, p$, is a zero-one vector of size n mapping the services included in its solution by having for every jth

entry

$$C_k[j] = \begin{cases} 0 & \text{Service } j \text{ from } N \text{ is used by the solution } C_k \\ 1 & \text{Service } j \text{ from } N \text{ is not used by the solution } C_k \end{cases}$$

The solution will have a combination of services which will offer a set of features and thus will have a fitness value based on the number of features covered. Higher fit solutions will be more likely to combine their solution vector with others, cross over and introduce new solutions to the population by replacing less fit solutions. Random mutations will also exist to introduce or remove services from the population resulting in more variation within the population. As each generation is iterated, it is likely that the population converges to produce a suitable solution, a set of services that fulfills the requested set of features.

In the context of the SCP, every solution within the population is required to be a feasible solution and thus is a valid cover for all the rows in A. When a child solution is created through cross-over and mutation, it is unlikely that the resulting solution will be a valid cover for all the rows. Also, the new solution may also have redundant columns, columns that when removed will still have a valid cover. The method it uses to fix this is called the *heuristic feasibility operator* and is applied to each child solution before it replaces a solution in the population space.

The calculations and algorithms used for the benchmark will follow closely from Beasley's implementation of the genetic algorithm [BC96b]. Given an $n \times m$ matrix A mapping each service to the features offered, the main heuristics are as follows.

Fitness Value Every k-th solution vector C_k for $k = 1, \ldots, p$ within a population P will have a *fitness value* based on their solution vector by the formula

$$f_k = \sum_{j=1}^{n} C_k[j]$$

where $C_k[j]$ is the jth entry bit in the soluction vector C_k. Note that a better fitness is one which is lower than the other solutions and has fewer services used. As the population as a whole converges, the fitness of all solutions will become more or less equal. Thus, a *scaled fitness value* f^s will be used differentiate a solution C_k within the population by the formula

$$f_k^s = f_k - \min(f_k, k = 1, \ldots, n)$$

Parent Selection Probability The two top fittest solutions will not automatically be chosen to be parents. Although random, higher fit solutions will have a higher probability of being selected. For the *proportionate selection method*, each solution C_k in a population has the probability that they will be selected as parents by the formula

$$\Pr(\text{solution } C_k \text{ is chosen as parent}) = \frac{1/f_k}{\sum_{j=1}^{n} 1/f_j}$$

However, Beasley [BC96b] makes use of the *binary tournament selection* in which a parent is selected by picking two solutions from a population and choosing the one with the lowest (better) fitness value. His study shows that the performance of binary tournament selection is better while the quality of the solution is comparable to that of proportionate selection.

Crossover Probability To avoid crossover bias on the ends of solution vector, Beasley [BC96b] makes use of a *fusion operator* which takes into account both parents', C_x and C_y ($x, y \in \{1, \ldots, p\}$), fitnesses f_{C_x} and f_{C_y}. The offspring solution vector Q will be constructed as follows:

For every bit j in both parent solution vectors

1. If $C_x[j] = C_y[j]$, assign $Q[j] := C_x[j] = C_y[j]$
2. If $C_x[j] \neq C_y[j]$,
   - Assign $Q[j] := C_x[j]$ with the probability $f_{C_x}/(f_{C_x} + f_{C_y})$
   - Assign $Q[j] := C_y[j]$ with the probability $f_{C_y}/(f_{C_x} + f_{C_y})$

Mutation To introduce new services or to remove some services from a population, *mutation* is applied to the offspring after crossover by inverting each bit with a calculated probability. Other GA researchers [B$\ddot{9}$3, Jon75, as cited by Beasley] suggest a uniform probability of $1/n$ where n is the number of services or the length of the solution vector. Beasley [BC96b] makes use of a increasing mutation rate as crossover becomes less effective as the population converges. The number of bits mutated is calculated as

$$\text{number of bits} = \left\lceil \frac{m_f}{1 + \exp(-4m_g(t - m_c)/m_f)} \right\rceil$$

where t is the number of child solutions generated (i.e. the number of iterations), m_f, m_g and m_c are constants set for each individual case based on its GA convergence curve. Very roughly, typical values chosen are:

1. m_f, the *final stable mutation rate*, is set to 5 or 10,
2. m_c, the number of child solutions required to achieve the mutation rate of $m_f/2$, is set somewhere around 200 to 400,
3. m_g, the gradient at $t = m_c$, is set somewhere around 0.5 to 2.0.

Although it may be more effective, this may give the GA an unfair disadvantage when comparing the algorithm in the testbench as the variables are fine tuned with each problem set. Thus, the value of $1/n$ will be used.

Heuristic feasibility operator When a child is created through cross-over and mutation, it is likely that all rows in A are not covered and that there will be redundant columns such that its removal will still result in a valid cover. The addition of columns can be done by finding the column that maximizes the number of uncovered rows which it covers.

Adding on the extra columns can also lead to the solution having redundant services and will be removed to minimize the cost. The resulting algorithm for a cover C will be as follows:

1. For each uncovered row i in C
   - From the set of all non chosen columns, find an unpicked column j that maximizes $|$ Set of uncovered rows $\cap$ Set of rows covered by$j$ $|$.

Note that Beasley's [BC96b] algorithm iterates through the columns by the weight of the column. The DBE will work with unicost columns thus will probably randomize the iteration chosen break ties arbitrarily.

The algorithm requires an $m \times n$ matrix A representing the m features offered by the n services, and a request vector R representing which of the m features are requested.

The algorithm also needs to keep track of the population by the $p \times n$ matrix P representing the p solution vectors, each keeping track of which of the n services are used by the solution.

The basic algorithm would consist of:

1. Reduce the partial SCP to the classical SCP:
   - Remove rows of A and R for every zero entry in R
   - Remove empty columns from A
   - Reset the variables m and n to reflect the changed sizes of A and R
2. Randomly populate p solutions into P
3. For each iteration
   (a) Pick two parents C_x and C_y from P by two rounds of binary selection
   (b) Create a child solution Q by using the fusion operator
   (c) Mutate the columns of Q
   (d) Apply the heuristic feasibility operator
   (e) If Q is identical to any other solutions in P, go back to step (b)
   (f) Replace a solution in P with the highest (worst) fitness with Q
   (g) If the fitness of solution Q is the lowest so far, record the iteration and contents
4. Output the best solution and the iteration from where it was generated

6.1.4 An univariate marginal distribution algorithm for the SCP

Proposed by Mühlenbein and Paaß [MP96], the *Univariate Marginal Distribution Algorithm* (UMDA) is part of a family of algorithms called *Estimation of Distribution Algorithms* (EDA). The EDA has emerged from the GA for the purpose of addressing the problems of having poor performance in some problems and difficulty of managing such a large number of algorithm variants [GLLn02].

Since the goal of finding a suitable set of services for a request will be the same as the GA. UMDA will also work with the same $m \times n$ A matrix representing the features offered by each service. As with the GA, UMDA will also work with a population P of solutions, each an n sized zero-one vector. Each entry in the solution vector indicates which services from N are used. The biggest difference between GAs and UMDA is that at every GA iteration, every solution in the population is a valid

cover and the goal is to find a solution that minimizes the number of services used. UMDA works by having a population of incomplete covers and the goal is to find a solution that solves the SCP instance.

GAs, for each iteration, select two parent solutions from a population (two occurrances of a binary tournament), create a child solution with cross-over and mutation, and replace a single individual in the population. Each UMDA iteration however, makes a binary tournament for all randomly paired individuals in the population, creates the Estimation Distribution derived from the winning population, uses the distribution to completely replace the succeeding population, and mutates the results. This distribution is called the *probability vector* X of size n and has the form $X = (x_1, x_2, \ldots, x_n)$ where $x_i \in [0, 1]$ is the average of the ith bit in the entire winning half of the population. x_i is then used to generate the next population where for each individual solution, the probability of having a 1 entry is x_i and a probability of having a 0 entry is $1 - x_i$.

Fitness Value Every k-th solution vector C_k for $k = 1, \ldots, p$ within a population P will have a *fitness value* based on their solution vector. It will simply be the number of features covered by all the services used

$$f_k = \sum_{i=1}^{m} \begin{cases} 1 & \text{if } i\text{th row is covered by a column in solution} \\ 0 & \text{if } i\text{th row is not covered by a column in solution} \end{cases}$$

Unlike the GA, note that a better fitness is one which is higher than the other solutions.

Binary Selection Just as Beasley [BC96b] makes use of the *binary tournament selection* to select two parents in the population, UMDA performs this on the entire population by randomly pairing each individual solution in the population. Each pair will compare their fitnesses and which ever has the lower (better) fitness will contribute to the probability vector.

Mutation To introduce new services or to remove some services from a population, *mutation* is applied to the offspring after crossover by inverting each bit with a calculated probability. Conventional GA researchers [B̈93, Jon75, as cited by Beasley] suggest a uniform probability of $1/n$ where n is the number of services or the length of the solution vector.

The algorithm requires an $m \times n$ matrix A representing the m features offered by the n services, and a request vector R representing which of the m features are requested.

The algorithm also needs to keep track of the population by the $p \times n$ matrix P representing the p solution vectors, each keeping track of which of the n services are used by the solution.

The basic algorithm would consist of:

1. Reduce the partial SCP to the classical SCP:
   - Remove rows of A and R for every zero entry in R
   - Remove empty columns from A
   - Reset the variables m and n to reflect the changed sizes of A and R
2. Randomly populate p solutions into P
3. Until a valid cover is made
   (a) Randomly pair up the population and perform a binary tournament with each pair
   (b) From the winning half of the population, calculate the probability distribution X
   (c) Create a new population based on X
   (d) Mutate the population
4. Output the valid cover

6.1.5 Testing approach

The algorithms are to be tested using the 65 SCP problems, categorized into 11 problem sets labeled 4 to 6, and A to E used by Beasley's OR-Library [BC96b]. These problem sets come in a variety of sizes and densities and should be suitable for this testbench. Problem sets 4 to 6 and A to D are those in which the optimal solutions of the minimum set cover is already known while problem sets E to H are large problems in which the optimal solution is unknown. Note that there may be costs associated with the columns in the original problem sets but for the purposes

of the DBE project testbench, they are assumed to be equal. This will undoubtedly change the known optimal solutions for these problem sets.

In the testbench, each of the 65 SCP instances are to undergo 10 trials, each with it own random seed and end after 100,000 iterations of producing non-duplicate solutions. For each test, execution time, the best individuals, and the number of iterations required to get the fittest solution are recorded. Since each problem is repeated 10 times, the average percentage deviation can calculated by the formula

$$\sum_{i=1}^{10}(S_{T_i} - S_o)/10 \times 100$$

where S_{T_i} is the fitness of the best solution for the ith trial and S_o is the fitness fo the best known solution for that problem.

The algorithms of the two approaches are sufficiently similar so that the total time taken and the number of iterations required to get to the best solution can be comparable.

The aim would be to test which of the two methods is more practical for use in the DBE project and to see how arguments for or against each algorithm holds. Specifically, Gonzalez et. al [GLLn02] state that certain GA problems are deceptive and produce poor performance, and there is difficulty in modelling so many algorithm variants. Problems with UMDA include the problem of finding solution diverging traps as in the ONEMAX problem, and the convergence to local optima [GLLn02]. Mühlenbein and Paaß [MP96] even mention that UMDA is not a global optimizing for difficult fitness functions.

6.2 Automatic theorem proving

As we introduced in chapter 2, SBVR has been chosen as version 2.0 of the business modelling language for the DBE. SBVR models have a direct mapping to first order logic and higher order logic with Henkin semantics. There are many well known results on these logics that imply the hardness of matching two SBVR models, where "matching" means to show that an SBVR model satisfies what is in a request formulated as an SBVR model. We shortly report here these results and propose how to overcome them for efficiently solving SBVR matching.

6.2.1 A short excursus on logic

Logic was born as a common language for formally expressing theorems and proofs. Although it is a basis for mathematics, the study of logic itself was not so relevant until late in the 19$^{\text{th}}$ century. This work brought a standardization and understanding of logic that is still current and found the basis for the theory of computability. The International Organization for Standardization is working on a specification of an interchange format for logic that will be used as the base to represent logical statements [ISO06].

We introduce here the propositional calculus, quantification theory, first order logic and higher order logic [Men97].

Definition 6.2.1 (Statement form) *Given some* statement letters*, the following are statement forms:*

- *all statement letters and such letters with numerical subscripts;*
- *if $\mathscr{B}$ and $\mathscr{C}$ are statement forms, then so are $(\neg\mathscr{B}), (\mathscr{B} \wedge \mathscr{C}), (\mathscr{B} \vee \mathscr{C}), (\mathscr{B} \Rightarrow \mathscr{C}), (\mathscr{B} \Leftrightarrow \mathscr{C})$.*

To avoid the burden of too many parenthesis, often precedence rules are used. Commonly, this is the operators precedence, from highest to lowest: $\neg, \wedge, \vee, \overset{\Rightarrow}{\Leftrightarrow}$.

Definition 6.2.2 (An axiom system for propositional calculus) *We call L the formal axiomatic theory defined by the following scheme of axioms (for any given statement form $\mathscr{B}, \mathscr{C}, \mathscr{D}$):*

(A1) $(\mathscr{B} \Rightarrow (\mathscr{C} \Rightarrow \mathscr{B}))$

(A2) $((\mathscr{B} \Rightarrow (\mathscr{C} \Rightarrow \mathscr{D})) \Rightarrow ((\mathscr{B} \Rightarrow \mathscr{C}) \Rightarrow (\mathscr{B} \Rightarrow \mathscr{D})))$

(A3) $(((\neg\mathscr{C}) \Rightarrow (\neg\mathscr{B})) \Rightarrow (((\neg\mathscr{C}) \Rightarrow \neg\mathscr{B}) \Rightarrow \mathscr{C}))$

and the following inference rule (MP):

$$\frac{\mathscr{B} \quad (\mathscr{B} \Rightarrow \mathscr{C})}{\mathscr{C}} \text{ (Modus Ponens)}$$

The other logical connectives are defined in the following way:

(D1) $(\mathscr{B} \wedge \mathscr{C})$ *as* $\neg(\mathscr{B} \Rightarrow \neg\mathscr{C})$

(D2) $(\mathscr{B} \vee \mathscr{C})$ *as* $(\neg\mathscr{B}) \Rightarrow \mathscr{C}$

(D3) $(\mathscr{B} \Leftrightarrow \mathscr{C})$ *as* $(\mathscr{B} \Rightarrow \mathscr{C}) \wedge (\mathscr{B} \Rightarrow \mathscr{C})$

Definition 6.2.3 (Terms of quantification theory) *Given the following:*

- *individual variables:* $x_1, x_2, \ldots, x_n, \ldots$;
- *individual constants:* $a_1, a_2, \ldots, a_n, \ldots$;
- *predicate letters:* A_k^n *(*n *and* k *are positive integers)*
- *function letters:* f_k^n *(*n *and* k *are positive integers)*

where the superscript n *indicates the number of arguments of the predicate/function. We define the terms of quantification theory as:*

- *variables and individual constants;*
- *if* $t_1, \ldots, t_n$ *are terms and* f_k^n *is a function letter, then* $f_k^n(t_1, \ldots, t_n)$ *is a term.*

Definition 6.2.4 (Atomic formulas of quantification theory) *If* $t_1, \ldots, t_n$ *are terms and* A_k^n *is a predicate letter, then* $A_k^n(t_1, \ldots, t_n)$ *is an atomic formula.*

Definition 6.2.5 (Well-formed formulas of quantification theory) *Formulas of quantification theory, called wf, are recursively defined as:*

1. *every atomic formula is a wf;*
2. *if* $\mathscr{B}$ *and* $\mathscr{C}$ *are wgs and* y *is a variable, then* $(\neg\mathscr{B})$, $(\mathscr{B} \Rightarrow \mathscr{C})$ *and* $((\forall y)\mathscr{B})$ *are wfs;*

The symbol $\exists$ *is traditionally defined as:*

$$((\exists x)\mathscr{B}) \text{ stands for } (\neg((\forall x)(\neg\mathscr{B})))$$

Any language $\mathscr{L}$ satisfying definition 6.2.4 is called a first-order language.

Definition 6.2.6 (Interpretation of a first-order language) *An interpretation of a first-order language* $\mathscr{L}$ *consists of:*

(a) a non-empty set D (the domain);

(b) for each predicate letter A_j^n in $\mathscr{L}$, an assignment of an n-place relation in D (a subset of D^n);

(c) for each function letter f_j^n in $\mathscr{L}$, an assignment of a function from D^n to D;

(d) for each individual constant a_i in $\mathscr{L}$, an assignment of some element of D.

Definition 6.2.7 (An axiom system for first-order languages) *We call K the formal axiomatic theory defined by the following scheme of axioms (for any given statement form $\mathscr{B}, \mathscr{C}, \mathscr{D}$):*

(A1) $\mathscr{B} \Rightarrow (\mathscr{C} \Rightarrow \mathscr{B})$

(A2) $(\mathscr{B} \Rightarrow (\mathscr{C} \Rightarrow \mathscr{D})) \Rightarrow ((\mathscr{B} \Rightarrow \mathscr{C}) \Rightarrow (\mathscr{B} \Rightarrow \mathscr{D}))$

(A3) $((\neg\mathscr{C}) \Rightarrow (\neg\mathscr{B})) \Rightarrow (((\neg\mathscr{C}) \Rightarrow \neg\mathscr{B}) \Rightarrow \mathscr{C})$

(A4) $(\forall x_i)\mathscr{B}(x_i) \Rightarrow \mathscr{B}(t)$ *if* $\mathscr{B}(x)$ *is a wf of* $\mathscr{L}$ *and* t *is a term of* $\mathscr{L}$ *that is free for* x_i *in* $\mathscr{B}(x_i)$.

(A5) $(\forall x_i)(\mathscr{B} \Rightarrow \mathscr{C}) \Rightarrow (\mathscr{B} \Rightarrow (\forall x_i)\mathscr{C})$ *if* $\mathscr{B}$ *contains no free occurrences of* x_i

and the following inference rules:

$$\frac{\mathscr{B} \quad (\mathscr{B} \Rightarrow \mathscr{C})}{\mathscr{C}} \text{ (Modus Ponens)} \qquad \frac{\mathscr{B}}{(\forall x_i)\mathscr{B}} \text{ (Generalization)}$$

The logic derived from the system of definition 6.2.7 is usually called first-order logic (FOL).

Definition 6.2.8 (First-order theory with equality) *Let K be a theory with a predicate letter A_1^2, and write $t = s$ for $A_1^2(t, s)$. Then K is a first-order theory with equality if the following are theorems of K:*

(A6) $(\forall x_1)x_1 = x_1$

(A7) $x = y \Rightarrow (\mathscr{B}(x, x) \Rightarrow \mathscr{B}(x, y)))$

Where $\mathscr{B}(x, y)$ is obtained from replacing some occurrences of x with ys in $\mathscr{B}(x, x)$.

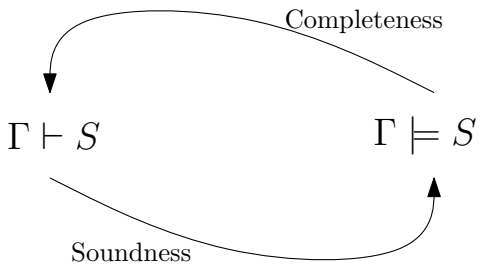

Figure 6.1: Soundness and completeness: $\Gamma \vdash S$ means that S can be deduced by the set of propositions defined in Γ; $\Gamma \models S$ means that all models where Γ is true make also S true. Soundness is when deduction implies consequence in the model. Completeness is when model-theoretic consequence implies deducibility.

The ISO-CL standardizes a first-order logic with equality giving some higher order constructions (like quantification over relations, see later), still preserving a first-order model theory [ISO06].

Informally, a *second-order language* is obtained by extending a first-order language with function variables and predicate variables and allowing to quantify over them. The standard semantics is obtained by adding mapping from the domain to these function and predicate variables. Second-order languages are much more expressive than first-order ones: as an example, they can state sentences that are true in an interpretation if and only if the domain of the interpretation is enumerable.

A *higher-order logic* is obtainable by introducing new function and predicate variables that can have as arguments any other variable defined in a lower-order logic and allowing quantification over them. This can produce any n-order logic.

An important results on second-order logic (and consequently higher order ones) is that standardly valid formulas are not enumerable; moreover, completeness and compactness theorems do not hold [Men97]. A *Henkin* interpretation restores the validity of these theorems, by limiting the range of function and predicate variables; still, also second-order logic with Henkin semantics is undecidable. SBVR uses Henkin semantics for the logical interpretation of its structures.

6.2.2 SBVR models to logical formulas

The logical foundations of SBVR are mainly established in Chapter 10 of the specification [OMG06], in particular in section 10.2 - "Formal Logic Interpretation Placed on SBVR Terms" that defines the mapping of SBVR concepts to the logic vocab-

ulary, and in Annex A - "Logical Foundations for SBVR" that defines the used semantics.

As the mapping is rather natural, we prefer to show some examples, rather than exhaustively cover it.

The following is a simple example that shows how variables are limited in range:

Each rental car *has* exactly one vehicle identification number

$$(\forall r : \text{rental\_cars})\ (\exists^1 v : \text{VIN})\ \text{has}(r, v)$$

The following is a more complicated example, involving also the use of mathematics (that is defined in the SBVR specification):

Each individual customer *is* a given person who *is* not a corporate renter and who *is responsible for* at least one rental that *is* a Reserved Rental or an or an Open Rental that *has* an end date that *is less than* 5 years *earlier than* the current day date.

$$\begin{aligned}&\text{is\_individual\_customer}(p : \text{person}) ::= \neg\text{is\_corporate\_renter}(p) \wedge \\ &(\exists r : \text{rental}(\text{is\_reserved\_rental(r)} \vee \text{is\_open\_rental(r)}) \wedge \\ &(r.\text{end\_date} - \text{today} < 5\ \text{years}) \wedge \text{responsible\_for}(p, r))\end{aligned}$$

If a user request is expressed in SBVR, and services have SBVR models, it should be by now clear that finding a set of services matching a request is equivalent to theorem proving. In particular, if we are given a set of SBVR models and we take the set Γ of all the propositions in them, they satisfy a request r if and only if from Γ we can obtain r, that is $\Gamma \models r$.

Consider the following example of a query formulated as an SBVR projection and mapped to logic:

$$\begin{aligned}&\{(h : \text{hotels}, f : \text{flights}, c : \text{cities}) \mid \\ &\text{hotel\_is\_in}(h, c) \wedge \text{flight\_from}(f, \text{Salzburg}) \wedge \\ &\text{flight\_to}(f, c) \wedge \text{is\_located\_in}(c, \text{Spain})\}\end{aligned}$$

Finding if a set of services corresponds to an element of the projection can be achieved by proving a theorem. In the following example, the logic mappings of SBVR models for a hotel, a flight and the body of shared meanings are proved to imply (satisfy) the request:

$$\begin{array}{r} (\exists h : hotels)\text{hotel\_is\_in}(h, \text{Barcelona}) \wedge \\ \text{hotel\_has\_name}(h, \text{HotelRamadaBarcelona}) \\ (\exists f : flight)\text{flight\_from}(f, \text{Salzburg}) \wedge \\ \text{flight\_to}(f, \text{Barcelona}) \wedge \\ \text{flight\_has\_code}(f, \text{LH2134}) \\ \text{is\_located\_in}(\text{Barcelona}, \text{Spain}) \end{array} \quad \models \quad \begin{array}{l} (\exists h : \text{hotels})(\exists f : \text{flights})(\exists c : \text{cities}) \\ \text{hotel\_is\_in}(h, c) \ \wedge \ \text{flight\_from}(f, \text{Salzburg}) \wedge \\ \text{flight\_to}(f, c) \ \wedge \ \text{is\_located\_in}(c, \text{Spain})\} \end{array}$$

Automated theorem proving is a rather recent field of computer science aimed at designing formal systems that can be used to formalize knowledge and automatically prove theorems on that knowledge. One of the most successful fields of application is hardware verification [KG99, BS89, CRS+94]. In particular, after the discovery of the Pentium FDIV bug in 1994, verification of FPU operations became of extreme importance to the industry [Har95]. Unexpectedly, even hardware verification needs the use of higher-order logic [KSK93, Koh98].

Unfortunately, automated theorem proving is far from being perfect, although it reached a good level of trust in specific fields. In particular, there are no efficient general purposes automated theorem provers [SS01].

In the case of SBVR matching, and in the specific case of the needs for the DBE project, there are different possible approaches:

Theorem Proving The problem can be tackled directly by using the logical representation of SBVR. As mentioned, general purpose automated theorem provers are not performing well enough, so it would probably be necessary to implement a custom prover. A good approach would be by making extensive use of unification, as SBVR variables are bound to specific sets (it corresponds to a sort of type theory) and unification would work very fast on these types. Additionally, unification theory is well known and studied for the implementation of compilers.

It is important to mention that a direct approach through theorem proving gives only yes/no matchings and does not provide a fitness value. If the automated theorem prover is implemented from scratch as proposed, some metric over logical statements can be introduced to simulate the notion of fitness. Theoretically, fitness could be defined as the inverse of the minimum amount of statements that must be added to be able to prove the consequence of the request from the set of services. In the implementation, some heuristic has to be used to approximate this number.

Prolog Another way of achieving the matching is by using the Prolog representation of SBVR models. Each SBVR model has a prolog representation: although not present in the specification, it is similar to the logical mapping. The open source SBVR Editor developed by ISUFI[1] is currently able to export Prolog databases of facts and rules. A request becomes then a Prolog query or a goal and any Prolog engine can be used to find a set of services satisfying the request.

An advantage of the Prolog approach is that there are engines for Fuzzy Prolog that provide a natural notion of fitness and the availability of interfaces for Java[2] or engines directly implemented in Java[3], allowing integration with the rest of the DBE toolkit suite.

It is interesting to notice that Prolog can be interpreted in π-calculus [Li93].

Flattening A total different approach that would rely on the fact that for matching only the actual features described are relevant is flattening. In flattening, the "tree" logical representation is flattened in a bit string, containing an encoding of the relevant features of a service. This approach has been studied by Gerard Briscoe (HWU) and Philippe de Wilde (HWU) in D6.6 - "High-Level Design Specification of the Distributed Intelligence System" as an approach for processing SBVR with neural networks [BW05, Appendix C].

As flattening is the most realistic approach in the short term, it has also been used as the basis for the optimization. In particular, using flattening the problem of SBVR matching reduces to the set cover.

Graph Matching If we consider the graph underlying the logical representation of the SBVR models, we can think of matching similar models by considering the distance between their graphs. We could for instance build a fully bipartite graph by connecting every node of the first model to every node of the second model, assigning a weight on these edges calculated by reasoning on the ontology and finally getting the maximum bipartite matching that would correspond to a fitness measure. Theorem provers based on graphs have been explored already in the past [M$\ddot{\text{O}}$97].

[1] SBeaVeR - Business Modeller, `http://sbeaver.sf.net/`

[2] for example the Ciao Prolog System, `http://www.ciaohome.org/`

[3] for example GNU Prolog for Java, `http://gnuprologjava.sourceforge.net/`

Developing a prototype of the presented approaches should involve the following tools and technologies:

OCaml[4] The Objective Caml system is the main implementation of the Caml functional language. It is actually a dialect with object-oriented features. It can compile very fast native code for numerous architectures and platforms and provides a byte-code interpreter for portability.

OCaml is particularly suitable for developing parsers, for its functional approach and the high availability of tools. It is actually one of the favorite choices of the research community for implementing compilers. Its mixed functional and imperative features allow to rapidly prototype code.

The Link Grammar Parser[5] The Link Grammar Parser is a syntactic parser of English, based on link grammar, an original theory of English syntax. Given a sentence, the system assigns to it a syntactic structure, which consists of a set of labeled links connecting pairs of words. It can be used to parse natural language and transform it into valid SBVR models. Needless to say, there exist a binding of the Link Grammar Parser for OCaml[6].

WordNet[7] WordNet is an online lexical reference system whose design is inspired by current psycholinguistic theories of human lexical memory. English nouns, verbs, adjectives and adverbs are organized into synonym sets, each representing one underlying lexical concept. Different relations link the synonym sets. As "always", there is a WordNet library for OCaml[8]. A project from 2003[9] makes use of WordNet to expand the lexicon of the Link Grammar Parser. WordNet can be used to infer relations among SBVR concepts of different vocabularies, to merge vocabularies and to help the users in creating models.

[6] `http://ramamurthy.ramu.googlepages.com/ocamllinkgrammar`
[8] `http://ramamurthy.ramu.googlepages.com/ocamlwordnet`
[9] `http://www.eturner.net/linkgrammar-wn`

Chapter 7

Conclusions and Future Work

Talk is cheap. Show me the code.
(Linus Torvalds, 1969-)

The ideas presented in this deliverable cry for additional research, testing and implementation. Before that, it is important to establish a solid theoretical base. We propose a double approach:

- from one side, continue the theoretical research that should found the basis for the digital ecosystems theory;
- from the other side, start experimenting with an implementation of heuristics for solving the computational problems.

These two approaches should be in constant contact as theory influences the implementation and the implementation gives hints on the right way for theory. We showed some of the strong links between theory and practice:

- π-calculus and the WS-CDL specification and reference implementation
- higher-order logic theorem proving and tool for heuristically matching SBVR models
- set cover and local/distributed optimization

They all are two ends of a bridge that should be crossed at the same time from both sides to meet in the middle. This requires a highly multi-disciplinary approach that, as was showed by the DBE consortium, is achievable although with difficulties.

The research lines followed in this document will continue in the *Open Philosophies for Associative Autopoietic Digital Ecosystems* Network of Excellence and in other projects part of the Digital Ecosystems cluster.

Bibliography

[AAF+02] Assaf Arkin, Sid Askary, Scott Fordin, Wolfgang Jekeli, Kohsuke Kawaguchi, David Orchard, Stefano Pogliani, Karsten Riemer, Susan Struble, Pal Takacsi-Nagy, Ivana Trickovic, and Sinisa Zimek. *Web Service Choreography Interface (WSCI) 1.0*, August 2002. W3C Note.

[AB05] G. Meredith A. Brown, C. Laneve. PiDuce: a process calculus with native XML datatypes. *Proceedings of 2nd International Workshop on Web Services and Formal Methods*, Lecture Notes in Computer Science 3670:18–34, 2005.

[AH95] Cevdet Aykanat and Ismail Haritaoglu. An efficient mean field annealing formulation for mapping unstructured domains to hypercubes. In *Workshop on Parallel Algorithms for Irregularly Structured Problems*, pages 115–120, 1995.

[Aic02] Uwe Aickelin. An indirect genetic algorithm for set covering problems. *Journal of the Operational Research Society*, 53(10):1118–1126, 2002.

[AJL+02] Bruce Alberts, Alexander Johnson, Julian Lewis, Martin Raff, Keith Roberts, and Peter Walter. *Molecular Biology of the Cell*. Garland Publishing, Inc., New York, NY, USA, fourth edition, 2002.

[AK00] D.A. Alexandrov and Y.A. Kochetov. The behavior of the ant colony algorithm for the set covering problem. In *Proceedings of Symposium on Operation Research*, pages 255–260. Springer-Verlag, 2000.

[AMS04] Noga Alon, Dana Moshkovitz, and Shmuel Safra. Algorithmic construction of sets for k-restrictions. To appear in the ACM Transactions on Algorithms, 2004.

[B̈93] T. Bäck. Optimal mutation rates in genetic search. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 2–9, San Mateo, CA, 1993. Morgan Kaufmann.

[Bar84] Hendrik Pieter Barendregt. *The Lambda calculus: Its syntax and semantics*. North-Holland, Amsterdam, 1984.

[BB90] Gerard Berry and Gerard Boudol. The chemical abstract machine. In *POPL '90: Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 81–94, New York, NY, USA, 1990. ACM Press.

[BC96a] Egon Balas and Maria C. Carrera. A dynamic subgradient-based branch-and-bound procedure for set covering. *Operations Research*, 44(6):875–890, Nov. 1996.

[BC96b] J. Beasley and P. Chu. A genetic algorithm for the set covering problem. *European Journal of Operational Research*, 94:392–404, 1996.

[BD04] Gerard Briscoe and Paolo Dini. Evolutionary environment discussion paper, June 2004.

[BFR05] Jean-Pierre Banâtre, Pascal Fradet, and Yann Radenac. Higher-order chemical programming style. In Jean-Pierre Bantre, Jean-Louis Giavitto, Pascal Fradet, and Olivier Michel, editors, *Unconventional Programming Paradigms (UPP'04)*, volume 35663566 of Lecture Notes in Computer Science, pages 84–95. Springer, 2005.

[BHK+06] Gerard Briscoe, Thomas Heistracher, Thomas Kurz, Giulio Marcon, Claudius Masuch, Jonathan Rowe, Philippe de Wilde, and John Woodward. Optimisation through a digital ecosystem. In *3rd European Workshop on Evolutionary Computation in Communication, Networks and Connected Systems*, 2006. (not accepted).

[BJ92] J.E. Beasley and K. Jörnsten. Enhancing an algorithm for set covering problems. *European Journal of Operational Research*, 58:293–300, 1992.

[BJT99] M.J. Brusco, L.W. Jacobs, and G.M. Thompson. A morphing procedure to supplement a simulated annealing heuristic for cost- and coverage-correlated set-covering problems. *Annals of Operations Research*, 86(0):611–627, January 1999.

[Boy99] Nik Boyd. Using natural language in software development. *Journal of Object-Oriented Programming*, 11(9):45–55, 1999.

[BR01] Patrizia Beraldi and Andrzej Ruszczynski. The probabilistic set covering problem. Technical report, Rutgers Center for Operations Research, 2001.

[BRF04] Jean-Pierre Banâtre, Yann Radenac, and Pascal Fradet. Chemical specification of autonomic systems. In Dosch and Debnath [DD04], pages 72–79.

[BS89] Graham Birtwistle and P. A. Subrahmanyam, editors. *Current trends in hardware verification and automated theorem proving.* Springer-Verlag New York, Inc., New York, NY, USA, 1989.

[BvHH+04] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. *OWL Web Ontology Language*, February 2004. W3C Recommendation.

[BW05] Gerard Briscoe and Philippe De Wilde. D6.6 - high-level design specification of the distributed intelligence system, December 2005.

[Car05a] Luca Cardelli. Abstract machines of systems biology. *Transactions on Computational Systems Biology. III, LNBI*, 3737:145–168, 2005.

[Car05b] Luca Cardelli. Brane calculi: Interactions of biological membranes. *Computational Methods in Systems Biology. International Conference CMSB 2004, Paris, France, May 2004, Revised Selected Papers. Lecture Notes in Computer Science*, 3082:257–280, 2005.

[CFT00] A. Caprara, M. Fischetti, and P. Toth. Algorithms for the set covering problem. *Annals of Operations Research*, 98:353–371, 2000.

[Chu32a] A. Church. A set of postulates for the foundation of logic. *Annals of Mathematics*, 33:346–366, 1932.

[Chu32b] A. Church. A set of postulates for the foundation of logic (second part). *Annals of Mathematics*, 34:839–864, 1932.

[Chu36] Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:345–363, 1936.

[Chv79] V. Chvátal. A greedy heuristic for the set covering problem. *Mathematics of Operations Research*, 4:233–235, 1979.

[CLT05] S. Barry Cooper, Benedikt Löwe, and Leen Torenvliet, editors. *New Computational Paradigms, First Conference on Computability in Europe, CiE 2005, Amsterdam, The Netherlands, June 8-12, 2005, Proceedings*, volume 3526 of *Lecture Notes in Computer Science*. Springer, 2005.

[CNS98] S. Ceria, P. Nobili, and A. Sassano. A Lagrangian-based heuristic for large-scale set covering problems. *Mathematical Programming*, 81:215–228, 1998.

[CP98] E. Cantú-Paz. A survey of parallel genetic algorithms. *Calculateurs Paralleles, Reseaux et Systems Repartis*, 10(2):141–171, 1998. Paris: Hermes.

[CP01] E. Cantú-Paz. Migration policies, selection pressure, and parallel evolutionary algorithms. *Journal of Heuristics*, 7(4):311–334, 2001.

[CPG99] E. Cantú-Paz and D.E. Goldberg. On the scalability of parallel genetic algorithms. *Evolutionary Computation*, 7(4):429–449, 1999.

[CRS+94] D. Cyrluk, S. Rajan, N. Shankar, , and M.K. Srivas. Effective theorem proving for hardware verification. In *Theorem Provers in Circuit Design (TPCD '94)*, volume 901 of *Lecture Notes in Computer Science*, pages 203–222, Bad Herrenalb, Germany, sep 1994. Springer-Verlag.

[DB04] Paolo Dini and Evangelia Berdou. D18.1 - report on DBE-specific use cases, December 2004.

[DD04] W. Dosch and N. Debnath, editors. *Proceedings of the ISCA 13th International Conference on Intelligent and Adaptive Systems and Software Engineering, Nice, France, July 1-3, 2004*. ISCA, 2004.

[DFM+04] Atsushi Doi, Sachie Fujita, Hiroshi Matsuno, Masao Nagasaki, and Satoru Miyano. Constructing biological pathway models with hybrid functional petri nets. *In Silico Biology*, 4(3):271–91, 2004.

[DL03] V. Danos and C. Laneve. Core formal molecular biology. In *Proceedings of the 12th European Symposium on Programming (ESOP'03)*, volume 2618, pages 302–318, Warsaw, Poland, April 2003. Springer-Verlag.

[DL04] Vincent Danos and Cosimo Laneve. Formal molecular biology. *Theor. Comput. Sci.*, 325(1):69–110, 2004.

[DPB+04] Kalyanmoy Deb, Riccardo Poli, Wolfgang Banzhaf, Hans-Georg Beyer, Edmund K. Burke, Paul J. Darwen, Dipankar Dasgupta, Dario Floreano, James A. Foster, Mark Harman, Owen Holland, Pier Luca Lanzi, Lee Spector, Andrea Tettamanzi, Dirk Thierens, and Andrew M. Tyrrell, editors. *Genetic and Evolutionary Computation - GECCO 2004, Genetic and Evolutionary Computation Conference, Seattle, WA, USA, June 26-30, 2004, Proceedings, Part I*, volume 3102 of *Lecture Notes in Computer Science*. Springer, 2004.

[EKL+02] Steven Eker, Merrill Knapp, Keith Laderoute, Patrick Lincoln, , and Carolyn Talcott. Pathway logic: Executable models of biological networks. In *Fourth International Workshop on Rewriting Logic and Its Applications (WRLA'2002*, volume 71 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2002.

[EKZ00] Anton V. Eremeev, Alexander A. Kolokolov, and Lidia A. Zaozerskaya. A hybrid algorithm for set covering problem. In *Proc. of International Workshop "Discrete Optimization Methods in Scheduling and Computer-Aided Design"*, pages 123–129, Minsk, Minsk, 2000.

[Ere99] Anton V. Eremeev. A genetic algorithm with a non-binary representation for the set covering problem. In *Proceedings of OR'98*, pages 175–181. Springer-Verlag, 1999.

[Erl05] Thomas Erl. *Service-Oriented Architecture, Concepts, Technology and Design*. Prentice Hall, Crawfordsville, IN, 2005.

[Fei98] Uriel Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[Fer05a] Pierfranco Ferronato. D21.2 - DBE Architecture Scoping Document, June 2005. release C.

[Fer05b] Pierfranco Ferronato. D21.3 - DBE Architecture Requirements, March 2005.

[FG96] Cédric Fournet and Georges Gonthier. The reflexive chemical abstract machine and the join-calculus. In *Conference record of the 23th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Language (POPL'96)*, pages 372–385. ACM, January 1996.

[FGL+96] Cédric Fournet, Georges Gonthier, Jean-Jacques Lévy, Luc Maranget, and Didier Rémy. A calculus of mobile agents. In Ugo Montanari and Vladimiro Sassone, editors, *Proceedings of the 7th International Conference on Concurrency Theory (CONCUR '96)*, volume 1119 of *Lecture Notes in Computer Science*, pages 406–421, Pisa, Italy, August 1996.

[FK90] Marshall L. Fisher and Pradeep Kedia. Optimal solution of set covering/partitioning problems using dual heuristics. *Management Science*, 36(6):674–688, Jun. 1990.

[Fle05] S Fleming. Codefarm, Oct 2005.

[FLMR97] Cédric Fournet, Cosimo Laneve, Luc Maranget, and Didier Rémy. Implicit typing à la ML for the join-calculus. In Antoni Mazurkiewicz and Jòzef Winkowski, editors, *Proceedings of the 8th International Conference on Concurrency Theory (CONCUR'97)*, volume 1243 of *Lecture Notes in Computer Science*, pages 196–212, Warsaw, Poland, July 1997.

[Fra03] David S. Frankel. *Model Driven Architecture*. Wiley Publishing, Indianapolis, 2003.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Boston, MA, USA, 1995.

[GHY93] Olivier Goldschmidt, Dorit S. Hochbaum, and Gang Yu. A modified greedy heuristic for the Set Covering problem with improved worst case bound. *Information Processing Letters*, 48(6):305–310, December 1993.

[Gil77] D.T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[GLLn02] C. González, J. A. Lozano, and P. Larrañaga. Mathematical modelling of umdac algorithm with tournament selection. behaviour on linear and quadratic functions. *Int. Journal of Approximate Reasoning*, 31(3):313–340, 2002.

[GMPV05a] Fernando C. Gomes, Claudio N. Meneses, Panos M. Pardalos, and Gerardo Valdisio R. Viana. Computational results of approximation algorithms for the vertex cover and set covering problems. To appear in Computers and Operations Research, February 2005.

[GMPV05b] Fernando C. Gomes, Claudio N. Meneses, Panos M. Pardalos, and Gerardo Valdisio R. Viana. Experimental analysis of approximation algorithms for the vertex cover and set covering problems. To appear in Computers and Operations Research, February 2005.

[Gol89] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, 1989.

[GW97] T. Grossman and A. Wool. Computational experience with approximation algorithms for the set covering problem. *Euro. J. Operational Research*, 101(1):81–92, August 1997.

[GW05] Dina Q. Goldin and Peter Wegner. The church-turing thesis: Breaking the myth. In Cooper et al. [CLT05], pages 152–168.

[Har95] John Harrison. Floating point verification in hol. In *Proceedings of the 8th International Workshop on Higher Order Logic Theorem Proving and Its Applications*, pages 186–199, London, UK, 1995. Springer-Verlag.

[HK04] Susanne Hartkopf and Thomas Kurz. Risk management as a means to manage interdisciplinary projects. In *Proc. Interdisciplinary Project Management*, 2004.

[ISO06] ISO. *Information technology – Common Logic (CL) – A framework for a family of logic-based languages*. International Organization for Standardization, Jun 2006.

[JB95] L.W. Jacobs and M.J. Brusco. A local-search heuristic for large set-covering problems. *Naval Research Logistics*, 42:1129–1140, 1995.

[JK05] F. Jouault and I. Kurtev. Transforming models with atl. In Jean-Michel Bruel, editor, *Satellite Events at the MoDELS 2005 Conference: MoDELS 2005 International Workshops OCLWS, MoDeVA, MARTES, AOM, MTiP, WiSME, MODAUI, NfC, MDD, WUsCAMLNCS 3844*, pages 128–138, Montego Bay, Jamaica, October 2005. Springer Berlin / Heidelberg.

[Jon75] K.A. De Jong. *An analzsis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, 1975.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[KBR+05] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon, and C.Barreto. *Web Services Choreography Description Language Version 1.0*, November 2005. W3C Candidate Recommendation (work in progress).

[KG99] Christoph Kern and Mark R. Greenstreet. Formal verification in hardware design: a survey. *ACM Trans. Des. Autom. Electron. Syst.*, 4(2):123–193, 1999.

[KHH04] Thomas Kurz, Susanne Hartkopf, and Thomas Heistracher. Risk management for embedded systems. In *Proc. International Conference on Software Process Improvement*, Washington, DC, June 2004.

[Kle52] Stephen Cole Kleene. *Introduction to Metamathematics.* Van Nostrand, New York, 1952.

[Koh98] M. Kohlhase. Higher-order automated theorem proving. In Wolfgang Bibel and Peter H. Schmidt, editors, *Automated Deduction: A Basis for Applications. Volume I, Foundations: Calculi and Methods*. Kluwer Academic Publishers, Dordrecht, 1998.

[KR35] S. C. Kleene and J. B. Rosser. The inconsistency of certain formal logics. *Annals of Mathematics*, 36(3):630–636, July 1935.

[KSK93] Ramayya Kumar, Klaus Schneider, and Thomas Kropf. Structuring and automating hardware proofs in a higher-order theorem-proving environment. *Form. Methods Syst. Des.*, 2(2):165–223, 1993.

[Lan97] W. B. Langdon. Fitness causes bloat: Simulated annealing, hill climbing and populations. Technical Report CSRP-97-22, University of Birmingham, School of Computer Science, Sep 1997.

[Li93] Zhenzhong Li. A π-calculus specification of prolog. In *ILPS '93: Proceedings of the 1993 international symposium on Logic programming*, page 680, Cambridge, MA, USA, 1993. MIT Press.

[LK04] Jingpeng Li and R.S.K. Kwan. A meta-heuristic with orthogonal experiment for the set covering problem. *J. Math. Model. Algorithms*, 3(3):263–283, 2004.

[LL94] L.A.N. Lorena and F.B Lopes. A surrogate heuristic for set covering problems. *European Journal of Operational Research*, 79:138–150, 1994.

[LY94] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981, 1994.

[M̈97] Tobias Müller. Conceptual graphs as terms : Prospects for resolution theorem proving. Master's thesis, Vrije Universiteit Amsterdam, 1997.

[Mas05] Claudius Masuch. Service manifest conceptional model (internal dbe report). Version 0.9, 2005.

[MBD+06] David Martin, Mark Burstein, Grit Denker, Daniel Elenius, Drew McDermott, Deborah McGuinness, Sheila McIlraith, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. *OWL-based Web Service Ontology*, March 2006. Pre-Release of version 1.2.

[Men97] Elliott Mendelson. *Introduction to mathematical logic*. Wadsworth Publ. Co., Belmont, CA, USA, 4th edition, 1997.

[Mil93] R. Milner. The polyadic pi-calculus: a tutorial. In F. L. Bauer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*, pages 203–246. Springer-Verlag, 1993.

[Mil99] Robin Milner. *Communicating and mobile systems : the π-calculus*. Cambridge University Press, 1999.

[Mil01] Julian Miller. What Bloat? Cartesian Genetic Programming on Boolean Problems. In Erik D. Goodman, editor, *2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*, pages 295–302, San Francisco, California, USA, Jul 2001.

[MKHM05] C. Masuch, T. Kurz, T. Heistracher, and G. Marcon. D16.1, part 2 - Service Manifest Conceptual and Software Models, October 2005.

[MP96] H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions i. binary parameters. In H.-M Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature*, volume IV of *Parallel Problem Solving from Nature*, pages 188–197, 1996.

[MP00] Nicholas Freitag McPhee and Riccardo Poli. A schema theory analysis of the evolution of size in genetic programming with linear representations. Technical Report CSRP-00-22, University of Birmingham, School of Computer Science, November 2000.

[MP01] Nicholas Freitag McPhee and Riccardo Poli. A schema theory analysis of the evolution of size in genetic programming with linear representations. In Julian F. Miller, Marco Tomassini, Pier Luca Lanzi, Conor Ryan, Andrea G. B. Tettamanzi, and William B. Langdon, editors, *Genetic Programming, Proceedings of EuroGP'2001*, volume 2038 of *Lecture Notes in Computer Science*, pages 108–125, Lake Como, Italy, Apr 2001. Springer-Verlag.

[MSSH99] Nagasaki M, Onami S, Miyano S, and Kitano H. Bio-calculus: Its Concept and Molecular Interaction. *Genome Inform Ser Workshop Genome Inform*, 10:133–143, 1999.

[Nac02] Francesco Nachira. Toward a network of digital business ecosystems fostering the local development. Discussion paper, September 2002.

[OMG05] OMG. *Semantics of a Foundational Subset for Executable UML Models*, April 2005. Request For Proposal.

[OMG06] OMG. *Semantics of Business Vocabulary and Business Rules Specification*, March 2006. First interim specification, `http://www.omg.org/docs/dtc/06-03-02.pdf`.

[Pău02] Gheorghe Păun. *Membrane Computing. An Introduction.* Springer-Verlag, Berlin, 2002.

[Pie97] Benjamin Pierce. Foundational calculi for programming languages. In Allen B. Tucker, editor, *The Computer Science and Engineering Handbook*. CRC Press, Boca Raton, FL, 1997.

[PM01] Riccardo Poli and Nicholas Freitag McPhee. Exact schema theorems for GP with one-point and standard crossover operating on linear structures and their application to the study of the evolution of size. In Julian F. Miller, Marco Tomassini, Pier Luca Lanzi, Conor Ryan, Andrea G. B. Tettamanzi, and William B. Langdon, editors, *Genetic Programming, Proceedings of EuroGP'2001*, volume 2038, pages 126–142, Lake Como, Italy, 18-20 2001. Springer-Verlag.

[Pol03] Riccardo Poli. A simple but theoretically-motivated method to control bloat in genetic programming. In Conor Ryan, Terence Soule, Maarten

Keijzer, Edward Tsang, Riccardo Poli, and Ernesto Costa, editors, *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *Lecture Notes in Computer Science*, pages 204–217, Essex, Apr 2003. Springer-Verlag.

[Pri05a] Corrado Priami, editor. *Transactions on Computational Systems Biology I*, volume 3380 of *Lecture Notes in Computer Science*. Springer, 2005.

[Pri05b] Corrado Priami, editor. *Transactions on Computational Systems Biology I*, volume 3380 of *Lecture Notes in Computer Science*. Springer, 2005.

[PRSS01] Corrado Priami, Aviv Regev, Ehud Shapiro, and William Silverman. Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Inf. Process. Lett.*, 80(1):25–31, 2001.

[RPS+04] Aviv Regev, Ekaterina M. Panina, William Silverman, Luca Cardelli, and Ehud Shapiro. Bioambients: an abstraction for biological compartments. *Theor. Comput. Sci.*, 325(1):141–167, 2004.

[RRS05a] Magali Roux-Rouquié and Michel Soto. Virtualization in systems biology: Metamodels and modeling languages for semantic data integration. In *T. Comp. Sys. Biology* [Pri05a], pages 28–43.

[RRS05b] Magali Roux-Rouquié and Michel Soto. Virtualization in systems biology: Metamodels and modeling languages for semantic data integration. *Lecture Notes in Computer Science*, 3380:28–43, January 2005.

[RSS01] A. Regev, W. Silverman, and E. Shapiro. Representation and simulation of biochemical processes using the pi- calculus process algebra. In R. B. Altman, A. K. Dunker, L. Hunter, and T. E. Klein, editors, *Pacific Symposium on Biocomputing*, volume 6, pages 459–470, Singapore, 2001. World Scientific Press.

[Sch06] Markus Schacher. Moving from Zachman row 2 to Zachman row 3 business rules from an SBVR and an xUML perspective. *Business Rules Journal*, 7(6), June 2006.

[Sen93] Sandip Sen. Minimal cost set covering using probabilistic methods. In *SAC '93: Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing*, pages 157–164, New York, NY, USA, 1993. ACM Press.

[SGM02] Clemens Szyperski, Dominik Gruntz, and Stephan Murer. *Component Software - Beyond Object-Oriented Programming*. Addison-Wesley / ACM Press, second edition, 2002.

[SHE89] Motoshi Saeki, Hisayuki Horai, and Hajime Enomoto. Software development process from natural language specification. In *ICSE '89: Proceedings of the 11th international conference on Software engineering*, pages 64–73, New York, NY, USA, 1989. ACM Press.

[SPU02] Mauricio Solar, Vctor Parada, and Rodrigo Urrutia. A parallel genetic algorithm to solve the set-covering problem. *Comput. Oper. Res.*, 29(9):1221–1235, 2002.

[SS01] Geoff Sutcliffe and Christian Suttner. Evaluating general purpose automated theorem proving systems. *Artificial Intelligence*, 131(1-2):39–54, September 2001.

[SW01] D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[Tur36] Alan M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936.

[Tur48] Alan M. Turing. Intelligent machinery. In *Machine Intelligence*, volume 5, chapter 1, pages 3–23. Edinburgh University Press, Edinburgh, UK, 1948.

[Tur50] Alan M. Turing. Computing machinery and intelligence. *Mind*, 59:433–460, 1950.

[Tur96] P Turney. Myths and legends of the baldwin effect. In *Proceedings of the 13th International Conference on Machine Learning*, pages 135–142, 1996.

[vLW01] J. van Leeuwen and J. Wiedermann. The turing machine paradigm in contemporary computing. In Bjorn Enquist and Wilfried Schmid, editors, *Mathematics Unlimited – 2001 and Beyond*, pages 1139–1155. Springer-Verlag, Berlin, 2001.

[Weg98] Peter Wegner. Interactive foundations of computing. *Theor. Comput. Sci.*, 192(2):315–351, 1998.

[WG03] Peter Wegner and Dina Q. Goldin. Computation beyond turing machines. *Commun. ACM*, 46(4):100–102, 2003.

[YKI03] Mutsunori Yagiura, Masahiro Kishida, and Toshihide Ibaraki. A 3-flip neighborhood local search for the set covering problem. To appear in European Journal of Operational Research, 2003.

Appendix A

A LaTeX template for SBVR

The following template was used for producing figures in this deliverable. As no SBVR template in LaTeXis available at the time of writing (only a Microsoft Word-based template is), we considered it useful to make this material public. Of course further work would be needed for a full template but the current one is already enough for the examples and gives all the basic ideas for the full one.

The template consists justs in the following LaTeX preamble:

```
\usepackage[T1]{fontenc}
\usepackage{color}
\usepackage{ulem}
\RequirePackage{calc}

\renewcommand{\sfdefault}{phv}
\renewcommand{\seriesdefault}{mc} % Narrow

\definecolor{sbvr-keyword-color}{rgb}{1,0.4,0}
\definecolor{sbvr-concept-color}{rgb}{0.28627, 0.51373, 0.34510}
\definecolor{sbvr-text-color}{rgb}{0.2, 0.4, 1.0}
\def\sbvrhead#1{{\sffamily \textbf{\textit{\textcolor{sbvr-text-color}{#1}}}}}
\def\sbvr#1{{\sffamily \textit{\textcolor{sbvr-text-color}{#1}}}}
\def\concept#1
  {\textup{\textcolor{sbvr-concept-color}{\uline{#1}}}}
\def\indconcept#1
  {\textup{\textcolor{sbvr-concept-color}{\uuline{#1}}}}
```

```
\def\sbvrkeyword#1
  {\textrm{\textup{\textcolor{sbvr-keyword-color}{#1}}}}
\def\sbvrthat{\sbvrkeyword{that} }
\def\sbvrnot{\sbvrkeyword{not} }
\def\sbvrthe{\sbvrkeyword{the} }
\def\sbvreach{\sbvrkeyword{each} }
```

The following is a usage example (it produces the text that can be seen in Fig. 5.9):

```
\def\deflen{65pt}
\def\textlen{\textwidth-\deflen-6\tabcolsep}

\sbvr{\concept{receiving branch}}\\
\begin{tabular}{p{0pt}p{\deflen}p{\textlen}}
& Concept Type:
& \sbvr{\concept{role}}\\
& Definition:
& \sbvr{\concept{branch} \sbvrthat is the destination of a
\concept{car movement}}
\end{tabular}

\sbvr{\concept{sending branch}}\\
\begin{tabular}{p{0pt}p{\deflen}p{\textlen}}
& Concept Type:
& \sbvr{\concept{role}}\\
& Definition:
& \sbvr{\concept{branch} \sbvrthat is the origin of a
\concept{car movement}}
\end{tabular}

\sbvr{\concept{car movement}}\\
\begin{tabular}{p{0pt}p{\deflen}p{\textlen}}
& Definition:
& \sbvr{planned movement of a \concept{rental car} of a specified
```

```
\concept{car group} from a \concept{sending branch} to a
\concept{receiving branch}}\\
\end{tabular}

\sbvr{\concept{branch}}\\
\begin{tabular}{p{0pt}p{\deflen}p{\textlen}}
& Concept Type:
& \sbvr{\concept{organization function}}\\
& Definition:
& \sbvr{\concept{rental organization unit} \sbvrthat has rental
responsability}
\end{tabular}
```

Appendix B

Church numerals in OCaml

```
let rec church = function
  | 0 ->
    ( function f -> function x -> x )
  | n ->
    ( function f -> function x -> f ( church ( n - 1 ) f x ) )

let unchurch n =
  n ( function x -> x + 1 ) 0

let succ n f x =
  f ( n f x )

let add m =
  m succ

let mult m n f =
  n ( m f )

let exp m n =
  n m
```

Example of usage:

```
val church : int -> ('a -> 'a) -> 'a -> 'a = <fun>
```

```
val unchurch : ((int -> int) -> int -> 'a) -> 'a = <fun>
val succ : (('a -> 'b) -> 'c -> 'a) -> ('a -> 'b) -> 'c -> 'b = <fun>
val add : ((((('a -> 'b) -> 'c -> 'a) ->
  ('a -> 'b) -> 'c -> 'b) -> 'd) -> 'd = <fun>
val mult : ('a -> 'b) -> ('b -> 'c) -> 'a -> 'c = <fun>
val exp : 'a -> ('a -> 'b) -> 'b = <fun>
# unchurch ( add ( church 3 ) ( church 7 ) );;
- : int = 10
# unchurch ( exp ( church 2 ) ( church 4 ) );;
- : int = 16
# unchurch ( mult ( church 3 ) ( church 7 ) );;
- : int = 21
```