OPAALS PROJECT

Contract n° IST-034824

WP3: Autopoietic P2P Networks

Del 3.15 – Finalised infrastructural and interaction model, interfaces and SBVR

Part II – Appendix

Generative Web Information Systems

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

Contract Number: IST-034824

Project Acronym: OPAALS

Deliverable N°: D3.15 (Part II)

Due date: May 2010

Delivery Date: September 2010

Short Description:

This Appendix to D3.15 contains the full PhD thesis of Alexandros Marinos, whose research work was carried out within the context of the OPAALS project. We believe that this provides an important self-contained account of the important advances that have been made in the use of SBVR, and so is reproduced in full.

The work presented in this thesis aims to realize a new type of information system, more dynamic and less opaque to its owners, specified with structured natural language models and queried through hypermedia. To accomplish this, we focus on Semantics of Business Vocabulary and Rules (SBVR) as a modelling language, Representational State Transfer (REST) as an interface paradigm and Relational Databases as the persistence mechanism. All three of these technologies have declarative underpinnings, focusing on the 'what' rather than the 'how', which is why their combination is feasible and effective. By creating appropriate mappings to align these technologies, we create a core platform for Generative Information Systems.

To this end, we present an architecture that binds the three technologies together and adds the concept of a Meta-Process, a way for users to perform process-like workflows without the system having explicit processes defined. The resulting system can gracefully handle unforeseen requests by its users, and exhibit process-like behaviour without requiring processes to be explicitly defined.

To complement this core architecture, we have created RETRO, a RESTful Transaction Model that allows users to perform more than one action with guarantees of atomicity over the Web. We also describe a service composition framework for Generative Web Information Systems which combines the strengths of the Web with the descriptive capabilities of SBVR to create a Web of Models in which GWIS are native.

To validate the conceptual architecture that has been constructed, we have implemented SBVR with Sails, a prototype Generative Web Information System that serves both as a proof of concept and as a basis for future work and exploration of the concept.

This model-driven and declarative approach makes semantics and policy integral to the operation of the information system and therefore the individual information system becomes a self-documenting native citizen of the digital ecosystem and the World Wide Web.

Author: Surrey

Partners contributed: Surrey

Made available to: All

VERSIONING		
VERSION	DATE	NAME, ORGANIZATION
1.0	**10/2010**	**A. MARINOS (SURREY)**

Please refer to Part I of this deliverable for further expansion of the cover sheet.

Contents

List of Figures

List of Tables

1 Introduction

Building an information system with the current methodologies is an uncertain proposition. Recent research indicates that only 35% of software development projects get completed in time and on budget [1] This is a marked increase from 16.2% in 1995 [2], but even this has come at the expense of a longer and more complex development process. It is understandable then, that businesses tend to avoid modifying their production information systems until absolutely necessary, as any attempt at modification introduces further uncertainty.

An objective of modern digital ecosystems (DE) research is to help people, organizations and small and medium enterprises (SMEs) better dynamically integrate, enabling them to utilize capabilities, access infrastructure, and compete in markets currently available only to large enterprises [3]. A large obstacle on the path towards realizing this vision is the inflexibility of information systems currently used by SMEs and other potential DE participants, which constitutes an internal barrier. Viewing the information system from the external perspective, the requirement to explicitly annotate provided services with up-to-date semantics for exposition in a DE effectively limits the population of accurately described services offered in a digital ecosystem, preventing critical mass.

From a more general perspective, technologies can be seen as conforming to one of two different modes of use [4]. Sterile systems are systems whose function is limited by their design and will perform the same tasks for the duration of their lifespan. An example of sterile a system is a typewriter, a television or a telephone network. On the contrary, generative systems are built to enable novel and unplanned usage, far beyond what their designers originally intended or could conceive. Typical examples of generative technologies are personal computers and the internet.

Applying this paradigm to information systems, we can see that most information systems today are sterile. They have been built to carry out a specific task, contain a fixed set of processes that can handle specific data models and this functionality cannot be changed without significant reimplementation effort. Newer developments in the field offer some degree of flexibility but their core is still procedural, ultimately dependent on costly intervention by specialised intermediaries, and therefore resistant to rapid adaptation. This is in contrast with the inherently dynamic nature of business and human society within which these systems are applied and causes significant inefficiencies which hinder the fulfilment of the digital revolution's promise.

1.1 Customizability of modern Information Systems

Information system development models today range between two extremes that have an effect on the resulting product. On the one end are the rigid mass market applications that are created for a general purpose. They include hard-coded models of the data they can handle and specific user interface and processes that limit how a user can interact with the data described by these models. These systems are then disseminated to a wide user base. On the other extreme are custom-built solutions designed around a problem perceived as unique. In that case a team of programmers and business analysts will discuss the problem space with the client and design a unique software solution around those requirements. This solution usually exists in a single instance as it was designed to fit the idiosyncrasies of the entity that commissioned its construction.

Of course, there exist more alternatives than the fixed and custom system extremes. Fixed software may allow configuration options that allow users limited customization of program behaviour. Plug-in architectures also allow a level of customizability. Open source software may qualify as fixed but can be modified if the user can dedicate the programming resources required. More recently, a wave of tools has appeared that allow users and owners to design the processes they want the system to handle on top of which a number of software components can be written to handle transformations and interaction with users. On the more custom side of the spectrum, frameworks factor out common functionality and allow code to be written that mostly deals with the unique business requirements of a software system.

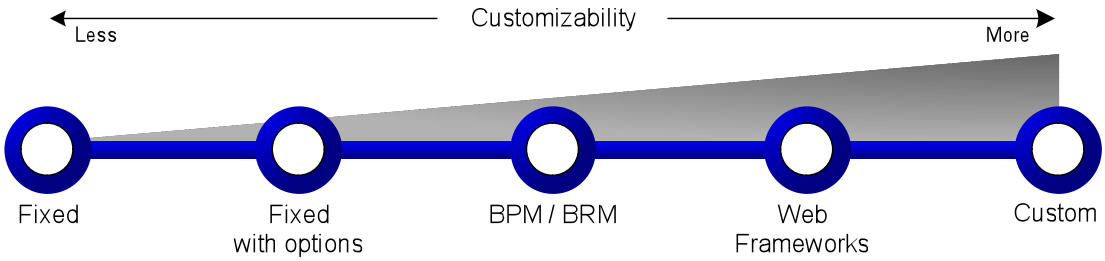

Figure 1. Customization Spectrum of Current Information Systems

Discussion of customization up to this point refers to the design-time of an information system's lifecycle. Once the system is deployed, more complexity presents itself. On the one hand, fixed systems cannot be upgraded without direct involvement of the system's original vendor, especially if the source code is not available.

Things may seem more survivable in the middle of the spectrum where BPM solutions can be found. These systems often allow the processes to be defined with graphical or natural language interfaces and therefore are available for modification by the system's owners. However, the processes themselves are by nature a secondary artefact, inferred from the primary requirements of the owners which they implicitly embed in their operation. Not being explicitly aware of the rationale with which the processes were designed, a maintainer may modify the processes in a way which leaves them in an inconsistent state and therefore exposes the system's data to the danger of corruption or leaves windows for the violation of the intended requirements. Additionally, business process tools usually terminate at a service invocation. The service behind that invocation also has to be somehow programmed. If the owner wishes to modify the behaviour of this service, they are again in need of programmer assistance. This set of issues with BPM tools show us that while they offer some improvement, they are not the way forward to the fully generative information system as processes embed implicit knowledge, and ultimately revert to calling other software with the same issues at a lower level.

Even in the case of custom systems, design-time decisions become a bottleneck to the evolvability of the resulting information system since its stakeholders depend on the developers for any change on the system. More specifically, any change in the business logic of a firm would result in a request to the programmers to modify or re-implement the relevant parts of the information system. The end users may also come up with new requirements as the environment changes. These requests must

be communicated to the owners and then again to the developers if they are to be implemented. Finally, integration with partner systems and their respective owners are also dependent on human actors to effect integration between the systems. Especially when a change in the information system interface is required, again the owner must be contacted to communicate the request to the developers. This state of affairs is visualised in figure 2. These considerations are further compounded when considering that a modern internet-facing information system is perpetually active and any changes have to be introduced without resulting in extended down time for the system.

1.2 Reliance on External Actors

From the structure of the customization spectrum in figure 1, it can be induced that flexibility comes at the cost of increasing the workload on specialized human actors such as business analysts and programmers. This is indeed currently the case and results in a number of drawbacks.

As a first concern, the use of humans to convert requirements into executable code can introduce flaws and inconsistencies in the resulting code. Secondly, the team of developers can become a single point of failure since the internal workings of the resulting code are immediately understandable only to them. Also, the separation between requirements and code that this development approach enforces, distances the information system from its stakeholders making auditing and validation infeasible. From the point of view of the developers, the work required of them consists mostly of reapplying well-known patterns which demands repeated mechanistic work if it is to be done correctly. This assumes that the process is able to complete at all which is not a given for software projects which often fail exactly because of this distance between requirements and code.

The discussed barriers to change in current approaches to information system development also make many individually low-frequency or timeframe specific use cases (the ‘long tail’) difficult to implement and maintain. Users are forced to operate in a sub-optimal combination of more common processes or forego specific use cases altogether. When seen as a whole, these individually minor use cases may represent a significant potential use value for the information system and the data therein. A typical information system, not explicitly encoding the owner's requirements but only a subset of the potential pathways to realizing pre-determined use cases as a function of those requirements, cannot respond to unforeseen but valid requests the way a human can. Therefore the information system itself becomes less useful to the business as a whole, not delivering its full potential for the owners and users. On the contrary, it may slow the business down becoming a necessary evil of the information age, more efficient than, but not as agile as, a purely human organization would have been. This can be thought of as a motivator for businesses to standardize their operations and forego more opportunities than they would like, especially in the case of SMEs. However, agility is the main differentiator of SMEs and asking them to conform to their information system rather than it conforming to business realities is a high price to pay.

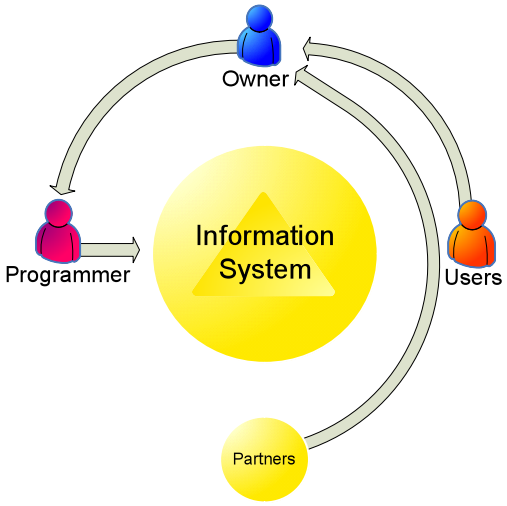

Figure 2. Stakeholder Interactions around a modern Information System

Conceptualising the spectrum of information systems along the lines of customizability makes dependence on human developers seem inevitable. However this is not a trade-off imposed by the very nature of information systems but rather a result of the currently predominant approach to their development. Instead of viewing requirements as a first-class citizen in the software execution process, they are seen as an external, informal and disconnected artefact from which program's desired behaviour must be extracted during an intermediate implementation phase. In this regard, even the design of processes is a form of programming since the owner is not concerned with the sequencing of the activities themselves but with the policies which the sequencing enforces, a point which will be elaborated further in following chapters.

1.3 Rethinking the Information System

In order to mitigate the drawbacks of the current approach, we turn to re-evaluating many of the foundations of the modern Information System. The approach presented here is geared to separation of concerns between stakeholders, which implies separation of requirements from executed code as a first priority. As a result, the other features of an information system are considered in relation to this central design principle.

In order to measure success in following this principle, specific roles for the stakeholders must be outlined. In this context, we view the owners' main concern as specifying policy-level requirements through structured natural language. The users should be able express arbitrary, unanticipated goals instead of using processes set at design time. Partner systems must have access to an owner-determined subset of the business logic which they can integrate with. The programmers can then focus on implementing and improving reusable logic-agnostic components of the system.

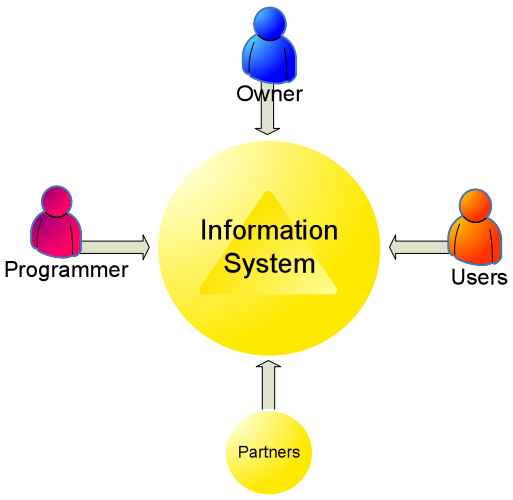

Figure 3. Stakeholder Interactions with the Generative Information System

The role of business analysts in this state of affairs is limited to the requirement of discussing with the business owners and collaborating with them to outline unambiguous requirements, to the extent that their assistance is required. While this role is significant, it is limited compared to their current role that includes the creation of business processes and the communication of the requirements with the programmers, both tasks outside the ability of the average business owner. In an ideal scenario, it should be feasible for a savvy owner to set their own requirements or modify pre-existing ones. In any case it is imperative that an owner is able to at least audit the final produced specification and can verify that this is indeed an accurate and satisfactory depiction of desired constraints on the system's behaviour.

The resulting Information System should be able to adapt to changes in requirements, IT paradigm, hardware architectures, development team, or business personnel. The newfound agility should also make feasible use cases that were impractical to implement explicitly.

1.4 Aims

The generative approach to information systems aims to model requirements unambiguously and positions them as first-class citizens on the information system architecture to be executed directly. This step mirrors the separation of data, data structures and data access algorithms from program code that the advent of the relational database effected. In this context, generative information systems can be seen as the evolutionary extension of the relational database, now aiming to cover the whole of the functionality of an information system. The requirements, formalized as business vocabulary and rules. This vocabulary and rules will serve a number of different functions apart from constraining system behaviour. They can also appear as error messages to the users, as service description for exposure in a DE and as a language for internal, multilingual communication within a business. A number of approaches have been proposed that are positioned outside the customizability spectrum and have a common theme of explicitly representing the requirements of the owner of the system. These approaches however, only address specific aspects of the problem and a comprehensive solution that can provide a complete information system based on unambiguous policy-level, structured natural language requirements has yet to be proposed.

This work aspires to produce a usable prototype release of a generative information system that is supported by a theoretical foundation of model driven engineering and declarative programming. In the process identifying, exploring and aligning these approaches, a number of other seemingly fundamental information system assumptions are challenged. The assumption that information systems require the definition of business processes which are inherently procedural is examined and a declarative alternative is proposed. Additionally, the traditional reliance on code generation for model driven engineering is re-examined against the alternative of direct model interpretation and caching of results.

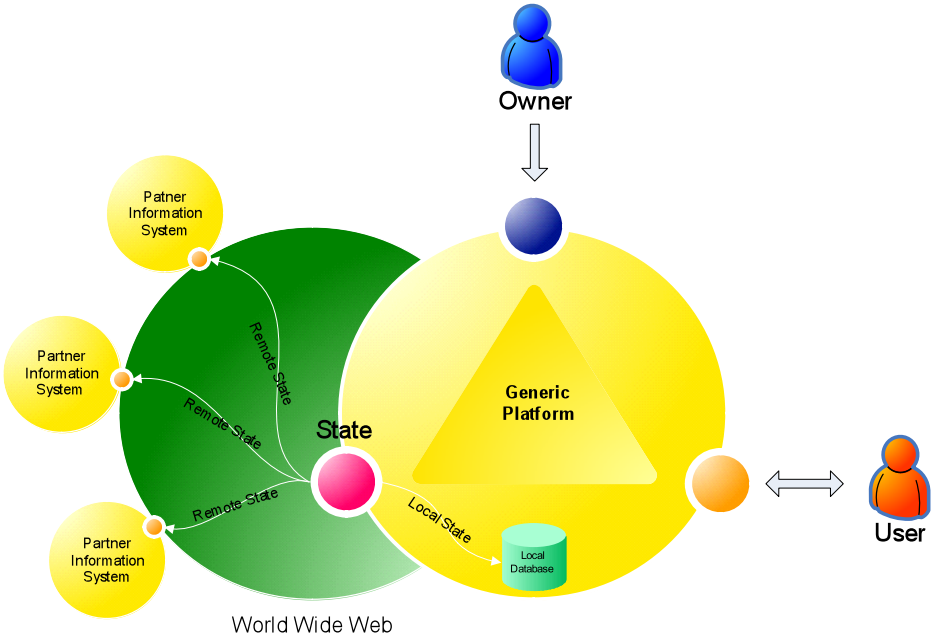

Figure 4. Abstract view of Generative Web Information Systems

By combining pre-existing work with new contributions, the expressive limits and flexibility that can be offered by the declarative paradigm in creating generative information systems will be explored.

This PhD project aims to realize a new type of information system, more dynamic and less opaque to its owners, specified with structured natural language models and queried through hypermedia. To accomplish this, we focus on Semantics of Business Vocabulary and Rules (SBVR) as a modelling language, Representational State Transfer (REST) as an interface paradigm and Relational Databases as the persistence mechanism. All three of these technologies have declarative underpinnings, focusing on the 'what' rather than the 'how', which is why their combination is feasible and effective. By creating appropriate mappings to align these technologies, and completing gaps where necessary with new technologies, we create the core of a purely declarative information system. The resulting system can gracefully handle unforeseen requests by its users, exhibiting process-like behaviour without requiring processes to be explicitly defined.

2 Definitions & Related Work

The terminology used in the first chapter has been purposefully been kept simple and underspecified aiming at a brief general introduction. The purpose of this chapter is to offer definitions for the most crucial terms that will be used throughout this thesis. Therefore, whereas until now a general description of the generative information system has been provided, this section aims to provide its theoretical derivation and position it within the web and digital ecosystem paradigms. We then examine efforts that are relevant to the development of a Generative Web Information System. Some of these efforts have served as direct influences with elements appearing directly in the proposed approach, while some others are related or parallel and should be covered both as a source of future work and as validation of the feasibility of various parts of the generative information systems vision.

2.1 Defining Generative Information Systems

2.1.1 Generative vs. Sterile technologies

In his book “The Future of the Internet-And How to Stop It” [4], Jonathan Zittrain defines the spectrum of technologies that can be defined as Generative on one end, and sterile on the other. Generative systems foster innovation and disruption by being open to use cases that the original designers could not predict or imagine. On the other hand, sterile technologies come pre-programmed with their functionality locked in. Their manufacturers may perform centralized feature rollouts, in which case they can be described as 'tethered'. The stability of a sterile system may be an advantage over generative systems where unexpected developments can happen, for better or worse. On the other hand, the security of the sterile system may be higher against attacks that focus on the lower quality of code that may exist in a generative system, but is affected by the centralized point of failure and control that monoculture and tethering to a vendor causes. It is of course possible to argue what the definition of ‘predetermined functionality’ is, and therefore support or challenge any assertion about a specific technology being generative. For instance a hammer can be used as a paperweight. Does this make it generative? This can be overcome by thinking of generativity as not necessarily a binary property that either exists or is absent, but rather as a comparative statement. So something is ‘more generative’ or ‘less generative’ than something else. As an approximation things at both extremes of the scale can broadly be called ‘generative’ or ‘sterile’ but it is important to keep in mind that this is indeed a relative characterisation.

2.1.2 Information System

Since this work is aimed at the field of information systems, it is only fitting that the term itself is thoroughly defined. Aside from the related but separate definition of information systems as the department of a business that is responsible for maintaining the information system itself, the definition of information system has two distinct aspects in literature. The first is the technological:

"Any written, electronic, or graphical method of communicating information. The basis of an information system is the sharing and processing of information and ideas. Computers and telecommunication technologies have become essential information system components." [5]

This family of definitions is centred on the technical components that comprise an information system but leaves out a very important aspect, the social element that arises when the system includes the human factor. In fact, while paper-based information systems have existed before the invention of the computer, imagining an information system that exists isolated from human activity is much more challenging. Therefore more modern definitions are careful to incorporate the human factor.

"Organised collections of hardware, software, supplies, policies, procedures and people, which store, process and provide access to information." *[6]*

A system which supports decision-making concerning some piece of reality, the object system, by giving the decision makers access to information concerning relevant aspects of the object system and its environment." [7]

These definitions, while much more accurate, could be considered generic from the point of view of the research presented here, since they do not explicitly separate the logic from the data in a system. This is a crucial distinction if we are to see an information system from a declarative point of view. Based on the previous definition, for the purposes of this document, an information system is defined as follows:

The structure and policies that constrain the information, the hardware and software components that store, manage and provide access to the information, and ultimately the humans that interact with the information and shape the technology, structure and policies.

It is also important to note that the internet itself can be seen as an information system in the sense of providing access to information. This information system arises from the interconnection of multiple smaller information systems. Specific application that help navigate such an information system, such as search engine, may often be considered as information systems in and of themselves, but these applications exist in symbiosis with the rest of the network and are worthless in isolation.

2.1.3 Declarative and Imperative Programming

The discussion of generativity in the context of information systems inevitably leads to the question of what a generative information system would resemble. From the first chapter discussion it is clear that our target information system allows all stakeholders maximum flexibility within the constraints set by the owner. At the root of the differences between the current approaches to information systems, and the approach proposed in this thesis, lies the difference between the declarative and the imperative paradigms of programming. In order to understand their differences, it is important to separate the ‘what’ and the ‘how’ of a solution to a computing problem, given specific input [8]. The ‘what’ refers to a description of the properties that an acceptable solution to the problem must possess. The ‘how’ refers to the steps followed to achieve the solution.

Declarative programming focuses on specifying the ‘what’ and depending on a software platform to decide which is the most appropriate way of materialising the goal. An example of a declarative language is the well-known Structured Query Language (SQL) which specifies properties of data but not the way to retrieve it, which is left to the database implementation. [9] The example of SQL as a

declarative language and the database as a platform is in many ways related to the approach presented here and will be explored further.

Imperative programming focuses on the 'how', bypassing the need to define the properties of the required solution since the programmer can guarantee the desired properties of the result by directly controlling the algorithm. Languages such as C, C++, and Java are generally considered imperative at their core.

Talking about automated inference of 'how' some task is to be performed is bound to bring up references to the Artificial Intelligence field. It should be noted that since the field of application is information systems, with well defined inputs and outputs, no true breakthroughs in artificial intelligence are believed to be required in this project. The means by which the inference layer is to be implemented are purely conventional and similar in style to technology used in databases for many decades.

By keeping the system specification at the declarative level, the objective of generativity is much easier to meet.

2.1.4 Declarative approach to Generative technologies

The major drawback of generative technologies, namely the possibility for misbehaving code to degrade the functionality of a product can be brought under control to a certain extent by applying the declarative approach to programming. This is accomplished by the fact that a declarative program is self-documenting and therefore cannot make false claims about its functionality or conceal other layers of functionality inside complicated imperative algorithms. Additionally, since declarative and model-driven approaches abstract technical concepts and focus on the operating logic, the generative properties of a system can in fact be amplified by allowing many more users to innovate on any given generative system, for personal use or for further distribution.

2.1.5 Defining Generative Information Systems

Having established the concept of a declarative business model, the description of a generative information system is now within reach. A generative information system requires direct execution of a declarative business model through a generic platform. The declarative business model allows the owners to redefine the behaviour of the system to fit with the current needs. The result is an operating information system that satisfies the requirements of the declarative business model without requiring additional programming work specific to the information system instance.

Whereas the method to achieve this result is declarative and model-driven design, the result is an information system with generative properties. By applying the declarative mantra 'what not how', it follows that the result should be named 'Generative Information Systems'.

2.1.6 Service-Oriented Architecture

The modern information system does not exist as an isolated island. It is connected to other information systems through various networks and most notably, the internet. Over time, various concepts of how to achieve interconnecting information systems have arisen. The currently dominant super-paradigm on interconnecting information systems across technological and business boundaries is called service-oriented architecture. Defining service-oriented architecture is a thorny issue but a sufficiently general definition is the following:

Service Oriented Architecture (SOA) – a collection of loosely-coupled, distributed services which communicate and interoperate via agreed standards. [10]

From this definition, the fundamental elements to note are: distribution, loose-coupling and the concept of services.

2.1.7 Service

Although the object of our work is specific to e-services, there is insight to be gained by finding a definition of the term 'service' and what that is connected to in general. Among the many definitions in literature, the following three are interesting due to their focus on value:

A service is defined as a useful labour that produces business value, usually not embedded in a tangible commodity, and that is mainly delivered by software [11],[12]

A service is a provider/client interaction that creates and captures value. [13]

Services [...] have an inherent value that is transferred from the provider to the recipient. [14]

There is a deep connection between the definition of a service and the production of value, a connection that will be useful in further analysis.

2.1.8 Service Description and Service Implementation

For services arranged in a service-oriented architecture, the separation between description and implementation is crucial to maintaining loose coupling. While the service description represents the contract that the service adheres to, the service implementation is the software that actually behaves according to the contract. The loose coupling property is maintained by the fact that the implementation can change while the description remains fixed. As long as the behaviour described remains intact, no further constraints of platform, programming language, location, methodology or other are placed on the implementation. In a similar vein to the declarative/imperative dichotomy, the description focuses on the 'what' while the implementation is concerned with the 'how'.

2.1.9 Service Composition

Service composition is the process of combining atomic services in order to achieve a composite goal. It is separated into manual composition and automated composition. In manual composition, the services are positioned in a workflow by the user whereas in automated composition the user defines the goal of the service composition and depends on a software tool to define the most suitable way to achieve this goal. According to [15], automated composition can also be subdivided into three categories.

The first is ‘Fulfilling Preconditions’ in which pre-existing services are combined in UNIX pipeline fashion to fulfil the requirements of a service that does not exist in atomic form. For example a .doc to .PDF converter and a .PDF printer can be combined to provide a .doc printer service.

The second is ‘Generating Multiple Effects’ in which a number of services should be executed to produce separate but interrelated outcomes. A typical example of this type of composition is the travel scenario where flight and hotel can be booked independently but must be coordinated for their combined results to be compatible and therefore usable.

Finally, a type of composition called 'Dealing with Missing Knowledge' is defined, where for instance a list of metro stations may be combined with a list of hotel addresses to identify hotels near metro stations. In this type of composition additional information sources are queried and the results are combined with the results from a service provider to satisfy more complicated queries.

While these types of service composition can be independent, there are problem domains which require a combination of approaches. In this regard, the three types can be considered tools that should be at the disposal of a complete service composition system, to be combined according to the requirements of a given query.

2.1.10 Declarative Programming and Service Composition

In the domain of service composition it is clear that there is a strong correlation between automated composition and declarative programming. Similarly, manual composition can be thought of as an application of imperative programming. It is important to be aware of the connection between programming paradigm and service composition strategy when selecting the appropriate method of interfacing with the user.

2.2 Digital Ecosystems and the Web

Using a declarative approach can be a limiting factor if the appropriate context is not chosen. Designing an information system as an isolated island of data and logic, capable of only interfacing with local users, is a naïve assumption that can derail an entire paradigm. Modern information systems live in a networked world and are able to consume and expose themselves as services. In fact, the motivation of this work is the need for flexible information systems for interconnected SMEs, in the context of a digital ecosystem. Generative Web Information Systems are therefore architected to be natively interconnectable.

2.2.1 Digital Ecosystems

A digital ecosystem is defined [16] as a "self-organising digital infrastructure aimed at creating a digital environment for networked organisations that supports the cooperation, the knowledge sharing, the development of open and adaptive technologies and evolutionary business models". In this context, the European Union has initiated a number of projects including DBE [17] and OPAALS [18] to further research in this area. This PhD project has been pursued in the context of the OPAALS project. The hope is that digital ecosystems can enable smaller businesses and individuals to form ad-hoc value networks that can provide offerings competitive to larger enterprises that none of the individual collaborators would be capable to provide as a stand-alone entity. In this way, it is hoped that information technology will unleash the creative energy of individuals and SMEs and permit them to continue evolving as agile independent units.

2.2.2 Combining Generative Information Systems and Digital Ecosystems

One major shortcoming of Digital Ecosystem prototypes up to now is the large barrier to entry that the lack of semantics for most businesses raises. A participant needs to adequately and unambiguously describe the services they offer in order to participate in the ecosystem. Additionally, the BML modelling language [19], [20] used for the DBE project, required the modelling of processes, especially for service composition. These obstacles create an even larger problem in the Digital Ecosystems field, as a digital ecosystem faces a tremendous collective barrier to bootstrapping. In order for the network effects to emerge, making the effort of modelling services worthwhile for businesses, a large number of businesses must model their services and make them

available to the ecosystem. This constitutes a typical catch 22 situation. On the other hand, generative information systems themselves face increased complexity of specification as a system grows in scope. Eventually the requirement set becomes too large to manage as a single unit.

Considering these problems, the two paradigms can form a symbiotic relationship. From the digital ecosystem point of view, since generative information systems require an unambiguous business model to start with, a subset of it can be published to a digital ecosystem as a service description therefore reusing the effort that was made to instantiate the declarative information system in the first place. The digital ecosystem can therefore accomplish bootstrapping without any large effort undertaken on the part of the participants for this purpose. From the generative information systems viewpoint, a large information system can be modelled as an ecosystem of smaller, interconnected generative information systems. This allows the information system to take advantage of the concepts of digital ecosystems such as service composition, transactions and service discovery while maintaining small business models for each of the composing systems.

Finally, declarative service composition with a low barrier to entry would satisfy the constraints of both approaches. It therefore becomes the point of unification between digital ecosystems and generative information systems as it is integral to achieving the objectives of both approaches, more so in their combined form.

Having documented the bidirectional relationship, generative information systems are studied within the context of digital ecosystems, as a proposed preferred method of implementing a node in the digital ecosystem, which may be in the form of a local digital ecosystem itself.

2.2.3 The Web as a Digital Ecosystem

While this work intends to drive forward the concept of a Digital Ecosystem, the reality is that digital ecosystems remain a conceptual paradigm, decoupled from large-scale real-world deployments. This means that an implementation of a generative information system for a digital ecosystem would be a theoretical construct, untestable due to the lack of a suitable environment. At the same time, the architectural style of the Web as expressed by Representational State Transfer (REST) offers some insights that are very much in line with the Digital Ecosystem ethos, and may even offer solutions to open problems in the field. On the other hand, the web falls short as a service oriented architecture due to lack of traditional SOA-related capabilities such as service description, service composition and transactions. Also, as a digital ecosystem, the presence of single points of control such as the DNS system or the emerging cloud computing vendors and social networking walled gardens are cause for unease. As a result, significant amount of work has gone into defining additional capabilities for the Web, built on top of REST, to take advantage of its natural strengths while mitigating its shortcomings.

2.2.4 Web Information Systems

Given that the Generative Information System that is to be developed will be tightly integrated with the web, it is appropriate to look into the definition of a Web Information System (WIS). Isaakowitz et al. define it as follows:

[...]We believe this type of system will become more pervasive than client/server systems did a decade ago, with an exponentially higher impact on our lives, simply because the Web has the potential of reaching a much wider audience than client/server systems based on proprietary

networks. There is a clear difference between a set of Web pages and a WIS. The latter supports work, and is usually tightly integrated with other non-WISs such as databases and transaction processing systems. WISs are also different from traditional information systems. They require new approaches to design and development, have the potential of reaching a much wider audience, and are usually a result of grass-roots efforts. These differences introduce managerial and technical challenges. [21]

We see here emphasised the difference of WIS from standard web pages, but also traditional information systems. This definition was written before the rise of REST as a stand-alone web services paradigm, but gains even more meaning because of it. Given the definitions above, what will be developed can be described as both a Generative Information System and a Web Information System, which leads to its description as a Generative Web Information System.

2.3 Conceptualizing Information System Development

Having analysed the environment within which a modern information system exists, we can now examine the methodologies and conceptualizations of information systems and their development. This will allow an initial exposure to the conceptual elements of an information system, making their interconnections visible.

Generic	Buildings	Airplanes	Information Systems
Ballpark	Bubble charts	Concepts	Scope/objectives
Owner's representation	Architect's drawings	Work breakdown structure	Model of the business (or business description)
Designer's representation	Architect's plans	Engineering design/bill-of-materials	Model of the information system (or information system description)
Builder's representation	Contractor's plans	Manufacturing engineering design/bill-of-materials	Technology model (or technology-constrained description)
Out-of-context representation	Shop plans	Assembly/fabrication drawings	Detailed description
Machine language representation	—	Numerical code programs	Machine language description (or object code)
Product	Building	Airplane	Information system

Figure 5. Zachman's comparison of engineering processes across disciplines [22].

2.3.1 The Zachman Framework

When IT systems started entering Businesses, they were too underpowered and simplistic in the capabilities they offered to be able to contain the whole enterprise within their operating reach. This would gradually change and by the late 1980's a seminal paper on Enterprise Architecture was published by John A. Zachman of IBM. This publication, entitled "A framework for information systems architecture" [22].

His vision might have been ahead of its time, however during the 1990's progress was starting to be made on its realization by various groups, especially the Business Rules Group (BRG) with two very important publications focusing on Business rules and the Business Motivation Model in 1995 and 1997 respectively [23],[24]. What Zachman attempted was to draw parallels between IT architecture and other more established engineering disciplines such as building and aircraft construction. By comparing the design processes used to arrive to the final product, he was able to generalize the design stages for engineering any complicated product and apply them to IT Architecture. The initial comparison can be seen in the following table:

During the following years, Zachman's model was refined and expanded to include more aspects of creating an IT system for an enterprise. An up to date representation follows:

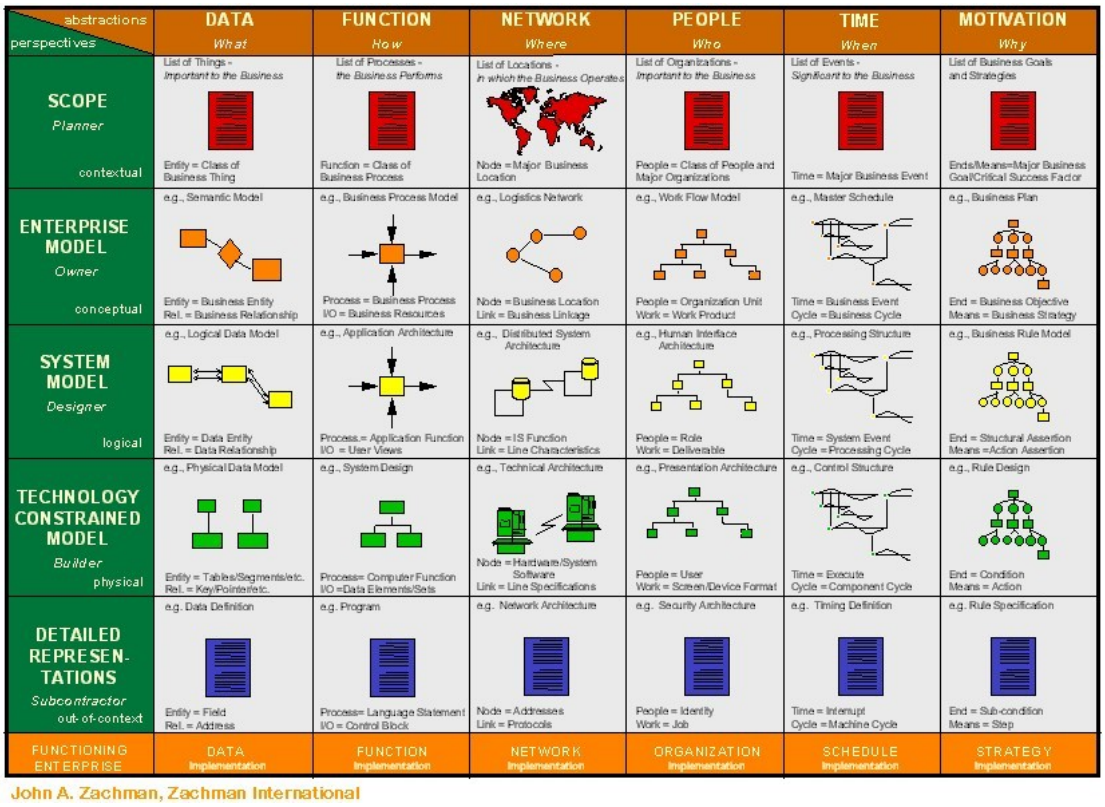

Figure 6. The Zachman Framework [25]

This complete framework is the basis for the latter developments in the related areas of business rules, business modelling, and Model-Driven Architecture. Specifically, the main focus is on the transition from row 2 to row 3 where the perceived barrier between the business and technological domains is located, and a shift of responsibility occurs between business people and IT specialists and systems.

2.3.2 Model-Driven Architecture

Model-Driven Architecture (MDA) is the name of a Model-Driven Engineering (MDE) initiative undertaken by the Object Modelling Group (OMG), a standards organisation known for UML, in order to address this problem. In fact this initiative is central to the OMG vision for interoperability and streamlined software development. MDA is an ambitious effort in the fact that it aims to change the way that software is developed and maintained. In many ways it can be seen as an extra level of abstraction in the direction that 4GL tools such as Power Builder and also CASE tools have attempted to go without much success. It can therefore be said that MDA is not a new concept but rather a new attempt at realizing the ultimate abstraction: linking the problem space with the solution space therefore limiting the need for input from the human factor to the absolute minimum.

Essentially MDA aims to create a clear separation of concerns and aid in interoperability and maintainability of software by standardizing many aspects of the software development life-cycle. Schmidt [26] mentions that MDA tools can detect and prevent many design errors early in the life cycle of a project which is a very important motive in choosing MDA over the traditional trial and

error method. It can be said that approaches such as MDA help bring software development closer to a true engineering discipline and further away from case-by-case trial and error methodologies that seem to be the de facto standard.

MDA prescribes use of modelling in order to allow the developer to work at the level that is more suitable for a given problem. The initial model is a Computation Independent Model (CIM), one that is used in gathering the requirements for the application to be constructed. The CIM does not show details of structure and is usually implemented with a domain-specific language. Once the CIM is complete, it is transformed to a Platform Independent Model (PIM) which specifies the operation of the system but without going into the details of operation within a specific platform. A Platform Specific Model (PSM) is then constructed in order to match the PIM with the platform that the target system needs to run on. In fact, a PIM can have multiple PSMs derived from it if it is to be implemented on many different platforms. Finally the PSM is converted to an initial code skeleton which then needs to be completed in order to have a fully working implementation.

In order to create the CIM and PIM languages, many times on demand, there was a need for a new framework to describe information, one that would allow domain specific languages to be implemented almost on-demand. This framework is the Model-Object Framework (MOF) which is defined as four levels of abstraction of data. Initially MOF was created as a meta-model for UML however today it is used for many other languages.

Stefan Tilkov [27] gives an approximate but effective description of the MOF:

"Level 0: A customer, ACME Corp., has placed an order for 32 boxes of candy on the 23rd of April.

Level 1: A customer has a name and can place zero to n orders, each order having a date and referencing a specific item that is being sold.

Level 2: Classes have attributes which have a type, and classes can be related to each other.

Level 3: There are things that describe something, and there are connections between those things.

"With numbers increasing, we move away from pure information (level 0) to a model layer (level 1), to a meta-model layer roughly resembling UML (level 2), to a meta-meta-model layer (level 3) representing a way to describe meta-models. "

At the base level (M0), information resides. This information is described at level (M1). The language used at M1 is described at M2 by means of the MOF meta-meta-model which resides at M3. This short description summarizes the essence of the MOF Framework.

While the concepts presented by MDA are useful in conceptualising the different levels of abstraction, the approach to information systems taken cannot be said to adhere to the prescriptions of MDA. The main weakness of MDA is its seeming lack of flexibility and overall complexity for the modellers. These are lessons learned that we have tried to heed in the work presented here.

2.4 Business Modelling Concepts

2.4.1 On the use of the word 'Business'

Many terms presented here are composite terms that contain the word business. Examples include 'business model', 'business process' and 'business rule'. While this terminology has originated from the business domain towards which information systems development was initially geared, information systems are no longer exclusive to businesses. There is no reason that a music application or a mobile phone cannot be considered an information system, and as such can be said to have a business model even though it may not operate within a business. To remedy this inconsistency, an effort has been made to reduce the usage of the word business throughout this document, as it may evoke an incomplete set of use cases in the mind of the reader. However, where the term is used, it is done to retain compatibility with common use. These cases should be treated as compound terms whose definition has drifted from the original and therefore are not indicative of anything relevant only to businesses but to businesses as well as any other field of human endeavour that may benefit from the application of an information system.

2.4.2 Business Process

The second column of the Zachman Framework is concerned with the "how" of a business. Lindsay and Lunn [28] note the difficulty of arriving at an accurate definition of business process and quote a number of definitions and descriptions the most interesting of which are:

"It contains purposeful activity, it is *carried out collaboratively by a group, it often crosses functional boundaries, it is invariably driven by outside agents or customers."*

'Set of partially ordered activities intended to reach a goal'.

A more specific and value-oriented definition is offered by Davenport [29] which defines a business process as:

"a structured, measured set of activities designed to produce a specific output for a particular customer or market. It implies a strong emphasis on how work is done within an organization, in contrast to a product focus's emphasis on what. A process is thus a specific ordering of work activities across time and space, with a beginning and an end, and clearly defined inputs and outputs: a structure for action. ... Taking a process approach implies adopting the customer's point of view. Processes are the structure by which an organization does what is necessary to produce value for its customers."

Hammer & Champy's [30] definition is similar to but more focused than Davenport's. They define a process as:

"a collection of activities that takes one or more kinds of input and creates an output that is of value to the customer."

As with services, the same link with value is observed. Both services and business processes are defined in terms of value production. Based on this correlation, we can argue that services and business processes are two sides of the same coin, with services referring to the view from outside an organization while processes refer to the internal view that the information system has of itself. While the relation may not be one to one, the correlation holds when considered in the context of

real implementations of service oriented architectures. This connection is instrumental in the architecture of Generative Information Systems.

2.4.3 Business Rules

Significant confusion exists in the business rules space since different approaches to the issue of business rules exist, with different standards and different definitions of the term that they adhere to. In many occasions, the term is used by approaches that have little in common to each other. For the purposes of this document, two types of business rules will be covered: Production rules, and Logic-based rules. In fact these names are largely used to differentiate the approaches since the relevant literature mostly refers to its preferred flavour simply as 'business rules', furthering the confusion.

Production Rules

Production rules systems are considered the evolution of expert systems, applied to information system needs. Production rules systems are popular enough that the term Business rules is considered synonymous to Production rules in some settings. Possible definitions of production rules are:

"An if/then condition incorporated into the inference engine of an expert system." [31]

"a statement of programming logic that specifies the execution of one or more actions in the case that its conditions are satisfied." [32]

It follows that the rules engines that contain there rules are called Business Rules Management Systems (BRMS) even though they cannot accommodate every kind of rule but simply production rules. A well known open source implementation of a BRMS is JBoss Rules [33] which is supported by Red Hat. Since an event may cause a requirement for multiple production rules to fire, many times with different results depending on the sequence the rules are executed, different algorithms have been implemented to determine which rules fire and when. The currently most popular approach is the RETE algorithm [34], improved versions of which are implemented in most popular BRMS systems. This rule chaining and co-dependence hints at an underlying imperative foundation for production rules. According to C. J. Date [8], event-based rules cannot be considered truly declarative and therefore are not considered a useful foundation for our approach

Logic-Based Rules

As the Zachman Framework was gradually maturing and becoming more accepted, the need for the description of a business started to become more pronounced. The GUIDE Business Rules Project was organized in November 1993 [23]. This project was created “to formalize an approach for identifying and articulating the rules which define the structure and control of the operation of an enterprise.”

This effort brought into existence the publication “Defining Business Rules – What are they really?” which put in place the foundations for the business rule approach. In it, a business rule was defined as

“A statement that defines or constrains some aspect of a business. It is intended to assert business structure or to control or influence the behaviour of the business” [23].

An alternative and more general definition is “A rule that is under business jurisdiction” [24], which defines a business rule as a rule under the exclusive control of the business as opposed to natural laws or mathematical laws. In the first definition, the two types of business rules are hinted. The one type called a Structural Rule is concerned with things that are true about the business and can never be false. An alternative and perhaps more modern name for a structural rule is Alethic rule, stemming from the Greek word «Alitheia» (Αλήθεια) which means truth. An example of a structural rule is:

It is necessary that an employee was born in exactly one country.

The second type of rules called Operative Rules are the rules that are concerned with the obligations of a business, which means they exist to control or influence the behavior of a business. The alternative name is “deontic rule” stemming from the Greek “Deon” (Δέον) which translates as “what should be” or “what is proper”. A deontic rule can possibly be broken, however enforcement of the business rules is not within the scope of business modelling. An example of an operative business rule is:

It is obligatory that an employee is married to exactly one person

While it is generally accepted in western societies that a person should only be married to only one person, this rule can and has been broken due to the fact that it is not naturally enforced like the structural rules are.

Alethic		Deontic	
Reading	Symbol	Reading	Symbol
It is necessary that	□	It is obligatory that	O
It is possible that	◊	It is permitted that	P
It is impossible that	~◊	It is forbidden that	F

Modal negation rules ~□p ◊~p ~Op P~p
~◊p □~p ~Pp O~p Fp

□ = true in all possible worlds ◊ = true in some possible worlds

Figure 7. Modal operators and rule verbalization [35]

Figure 7 compares deontic with alethic rules while displaying the relevant modal operators along with their reading. Additionally the model negation rules are listed. In practice, UML-based business modelling techniques are limited to Alethic rules which describe the structure that exists between specific components of an organization.

2.4.4 Declarative Programming and Business Rules

As with other areas explored previously, a familiar pattern is emerging with business rules also. Whereas production rules allow creating chains of events and therefore lend themselves to an imperative programming style, logic-based rules are independent statements. As the Business Rules Manifesto [47] states: “A set of statements is declarative only if the set has no implicit sequencing”.

For this reason, Logic-based rules are the option that is consistent with the declarative approach to information systems. Consequently, Logic-based rules are the type of rules that are used throughout our system and are what is meant when the term 'business rules' is used for the remainder of this document.

2.4.5 Business Model

Defining the term 'business model' is also quite challenging. Part of the problem was that the term was never officially coined and therefore never got an initial definition as a starting point. Instead, various communities (such business analysts, venture capitalists, journalists and IT specialists) have used the term in a way that makes sense within a single field but is highly inconsistent across fields. A very comprehensive study has been made by Osterwalder et al [36] which attempts to create an ontology for describing business models, and even makes a very interesting observation on the correlation of the dot com bubble of the late 90's with the rise to prominence of the term 'business model'. A quote from the paper reveals the general outlook of the authors on the relationship between business processes and business models:

"The business model concept is generally understood as a view of the firm's logic for creating and commercializing value, while the business process model is more about how a business case is implemented in processes."

However this reading of the literature does not cover all aspects. A different aspect surveyed in the same paper is the following:

"A business model is a conceptual tool containing a set of objects, concepts and their relationships with the objective to express the business logic of a specific firm."

This aspect is more conductive to one very important aspect that is absent from the first definition. It is the view of the Zachman Framework [25] which was also used throughout the EU-FP6 DBE project for the definition of a business model [19] [20] [37] and is also compatible with OMG's Model Driven Architecture [38]. Interestingly, the OMG is now referring to its CIM layer as 'Business Model' [39]. The second row of Zachman's Enterprise architecture, also called the 'Business Model', refers to the business owner's view and therefore does not contain technical implementation information but does contain business processes as the 'function' component of the business model.

2.4.6 Internal and External Business Model

From the preceding definitions, it can be seen that two very different viewpoints exist on the definition of business model. In order to use both of them we need to differentiate them by creating the appropriate terminology. For the purposes of this document, Internal Business Model refers to the information that is necessary to describe the workings of a business to enough detail as to make them implementable, akin to an architectural diagram of a building. This model is not usually made public as the knowledge of its details would arguably enable would be competitors to replicate the business. This model is also called the business logic, or simply logic, of a firm. On the other hand, the External business model is the description of the business that is made available to the public. This may be thought of as a product of the internal business model, analogous to a 3D rendering of a building. The external business model is concerned with information other entities such as customers or business partners will need in order to engage in transactions with said business, usually containing a rationalization of the value proposition that a business offers. This model can

also be called the business description as it correlates with an outsider's view of the company. From this point on, the term business model is used to refer to what is defined as the internal business model or the business logic of a firm.

2.4.7 Clarifying the verbs: Business Modelling, Business Process Modelling, and Business Model Design

Since the definition of business model is such a complicated issue, one can expect that the use of the relevant verbs will also be troublesome. While it is true that the use of the verbs is counterintuitive, there is much less disagreement in literature on what the proper use of the terms is.

Business model design refers to the activity of designing a company's business model. It is part of the business development and business strategy process and involves design methods. The unexpected part is that the verb phrase business modelling is usually synonymous with business process modelling. In the words of Osterwalder et al [36]:

"Part of the confusion comes from the expression "business modeling" being used mainly for the activity of business process modeling, which is the activity of modeling processes"

When someone wants to refer to the creation of a business model in the 'external' view of the first definition given above, the appropriate term is 'business model design'.

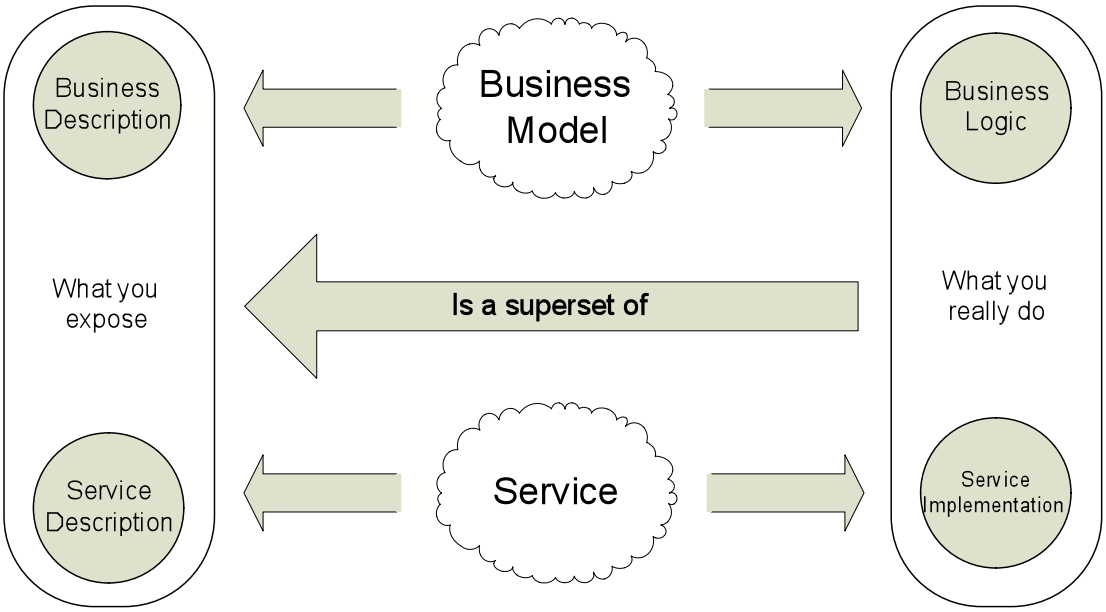

Figure 8. Business and Service terminology relations

2.4.8 The Connection Between Services and Business Models

As has been noted, the business/service pair displays the same dichotomy as the internal and external business models on the creation of value. Since the creation of value is the reason of any business to exist, there might be more to this parallel than is immediately obvious. In a sense, the first business description focuses on the external view of the business. Similarly, the definition of service is concerned with the 'what' of the service offerings. Similarly, the internal business model definition essentially sees the business model as the grouping of all relevant business elements (events, actors, activities, data model and perhaps processes) that lead to the creation of value, in

the same way that the service implementation does. These parallels can be conceptualised by positioning these definitions within the Zachman framework. We can consider the fundamental difference between a business model and a service to be the degree of immersion in technical details with the service being more technology-oriented while the business model is more concerned with the view of the owner. Given this separation, it can be said that a service is what resides in the third row of the Zachman framework, while the business model is embodied in the second row. It can therefore be said that a service is in fact the implementation of a business model on a technical architecture. This connection is a fundamental principle of the work presented here.

2.4.9 Combining Business Processes and Business Rules

The two aspects of business modelling presented, namely business processes and business rules, may seem incompatible at first, different tools of reaching the same aim, that of modelling the behaviour of a business. In fact, separate software systems exist that implement the each concept. Business Process Management Systems (BPMS) focus on managing business processes, while BRMSs are concerned with business rules, mostly the production rule type. BRMSs are usually limited to answering inquiries about decisions while BPMSs are focused on executing processes. Recently however, a number of approaches have appeared that attempt to combine the two approaches to create hybrid information systems.

The most widely practiced approach is that of using a BRMS as a decision service within a business process. This approach is quite straightforward and can be shown to offer benefits in simplifying the business process. A good example of such a combination has been presented by Vanthienen and Goedertier in [40]:

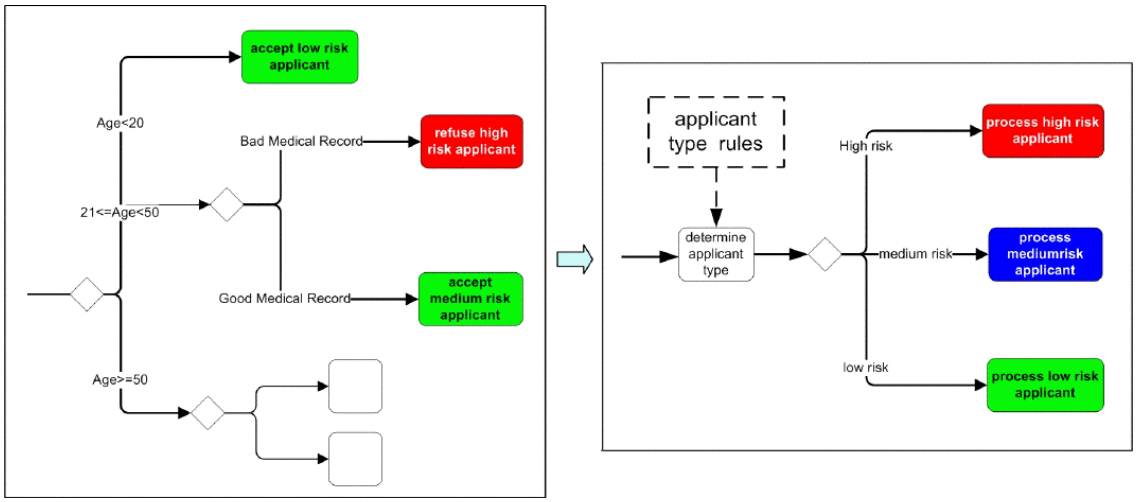

Figure 9. How Business Rules Define Business Processes [40]

Even though, confining rules within a process seems to be a safe approach from the BPMS vendors' point of view, it does not seem to cover all the possible cases, nor does it use all the advantages of business rules as a technology. In particular, there seem to be a number of cases when rules have particular effect on how business processes operate and therefore cannot be contained within them. A good example of that is a legal constraint such as the US Sabarnes-Oxley act, but also internal policies of a corporation that may affect more than one processes. A number of papers [41][42][43][44] have explored the possibility of having business processes that are shaped by rules.

It is quite clear that business processes are concerned with the “how” to accomplish a specific business goal. This focus on how makes their connection to the mindset of an imperative programming paradigm quite explicit. Having the owner specify business processes would be considered a violation in the declarative programming style. However, all types of information systems must accomplish goals and therefore need to sequence certain steps in a specific order. To achieve this, we need to disassemble processes into more elementary components. A very useful observation is made by Osterwalder et al [36]:"

"When modeling business processes procedurally, modelers inevitably make a number of modeling assumptions that are not present in the earlier specified requirements, this is called the assumption bias of a model. Therefore procedural models inherently risk to be over-specified as they are likely to impose more restrictions [...] than is strictly required."

The gap between requirements and executable processes is to be filled by the platform in a declarative system. However, once this step is made, there is no theoretical reason to support the existence of pre-specified processes at all. The processes can be generated on demand, based on the goals expressed by the user at run-time, in the context of logic-based business rules that the owner has entered at design-time. By analysing business processes into user goals, owner requirements, and execution decisions, we can assert that an information system that displays business process behaviour without having explicitly defined business processes is feasible.

2.4.10 Declarative Business Model

Since we want to avoid having the owner define bespoke business processes, a declarative business model can be defined as a set of atomic and independent but compatible constraints that together define behaviour well enough to be used for the construction of a business that is representative of the owner’s design. With this definition in mind, any deficiency of the implemented instantiation of the model in the eyes of the owner, assuming the model was applied correctly, implies that an aspect of the system was not specified in the stated requirements and is therefore a deficiency in the model itself. Additionally, from this point of view it can be claimed that a business process is not a primary design concern but rather the product of attempting to achieve specific goals within the constraints set by the business owner.

2.4.11 Limits of Business Modelling

At this point it is useful to address an issue that is bound to occur when one considers declarative business models. The idea of a declarative business model may evoke attempts to create systems that have a general understanding of reality, an elusive holy grail of Artificial General Intelligence research. This project is not an attempt to approach that goal. By discussing an approach to automating research in the limited field of yeast metabolism, a very good argument is made by Goedertier [45] that business is in fact a limited domain and not an open system such as wider reality and therefore in this limitation lays the power to model and automate it.

There exists a different argument that can be made to the same effect. Imperative information systems have until now been able to achieve the desired behaviour, therefore implicitly encoding the logic of the business. There is no reason to claim the same task cannot be accomplished by explicitly encoding the same requirements that would otherwise have been embedded in the information system. In practice, the history of the database system shows us it is feasible to move

from ad hoc solutions, to emerging patterns and from there to a standardized declarative platform for a now known domain.

2.5 Direct Influences

The following section discusses various approaches to information systems and relevant fields that have served as inspiration or direct influences in the presented architecture.

2.5.1 Relational Database

The relational database [46] is an architectural pattern that has stood the test of time. Since 1977, when the first version of system R was released by IBM, [9] it remains the default solution for data persistence in the vast majority of applications. The relational database can be seen as an extraction of emerging patterns in the field of data persistence. Seeing the commonalities in multiple ground-up implementations of custom persistence systems, the creators of the relational model, principally E.F.Codd, spent over a decade designing the relational database as a refinement of these patterns while simultaneously placing them on a firm theoretical foundation.

The most interesting element of the Relational database and System R in particular is the SQL querying language and the SQL data description language (SQL-DDL) since their use determines the rest of the architecture

The SQL-DDL allows the users to define a schema according to which the data is structured. The SQL querying language allows the users to make queries in the context of these structures. The result of the SQL languages is that the user does not have to perform any imperative programming tasks in order to gain the full generative power of the relational database, the same property we want to generalise for all information systems. In order to achieve this, the relational database pattern depends on the relational database management system (RDBMS) which is a generic platform, written by a central vendor and used in its unaltered form for all database instances, again a pattern we rely heavily upon.

The approach presented in this document has a strong connection to the relational database. In a sense, it could be considered a generalized form of the relational database system, extending the declarative abstraction from the data domain to the entire information system. Should the user desire to use it in a limited capacity, the system should be usable for the purpose of working with a schema and store data which can be queried with a declarative language, much like the relational database today. Additionally, it should provide web API and graphical user interface with no additional effort. This however would be a very limited use case for the proposed system

2.5.2 Business Rules Approach

The ‘Business Rules Approach’ is the result of many years of work from the ‘Business Rules Group’ that also led to the development of the SBVR standard. The Business Rules Approach focuses on logic-based type of business rules. According to this approach, business rules can be considered to be atomic and independent elements of the business logic. One definition of business rules that can be used to better clarify the concept is the following:

“[a] business rule is a specific, formal statement of a single term, fact, derivation, or constraint on the business” [23]

Central to the business rules approach is the Business Rules Manifesto [47], written by Ron Ross, widely regarded as 'the father of Business Rules'. This manifesto is also included in the SBVR Specification [48], signalling the intent to harmonize SBVR's use with the principles of the Business Rules Approach.

The following excerpts from the Business Rules Manifesto are relevant to this discussion:

"Article 2. Separate From Processes, Not Contained In Them

2.1. Rules are explicit constraints on behaviour and/or provide support to behaviour.

2.2. Rules are not process and not procedure. They should not be contained in either of these.

2.3. Rules apply across processes and procedures. There should be one cohesive body of rules, enforced consistently across all relevant areas of business activity.

…

Article 4. Declarative, Not Procedural

4.1. Rules should be expressed declaratively in natural-language sentences for the business audience.

…

4.3. A set of statements is declarative only if the set has no implicit sequencing."

From the quote above, the relationship between business rules and business processes is made clear. Additionally, it can be seen that the Business Rules Approach prescribes declarative usage of business rules.

2.5.3 Business Rules Approach to Application Development

One thing that the Business Rules Approach does not attempt to accomplish is the generation of complete information systems. This field was explored by the work of C.J. Date in his book "What not How, the Business Rules Approach to Application Development" [8], having the endorsement and encouragement of Ron Ross who wrote the foreword. The book describes a system that exposes the data model and completes it through rules such as constraints and inferences. The system described is devoid of processes only to be managed through an automatically generated user interface and a querying language. The approach presented is a natural extension of the relational model that is the cornerstone of most modern databases. The aim is to eliminate coding and represent business logic solely through declarative business rules. The book however does not discuss how to avoid processes, integration with other components or services and talks only briefly about the user interface. Nevertheless, many useful concepts are illustrated as well as the general vision of fully declarative information systems. In the book, a very useful classification of business rules is also presented:

- Rule
  - Constraint
    - State Constraint
    - Transition Constraint
    - Stimulus/Response
  - Derivation
    - Computation
    - Inference

The work of C.J. Date is a very important foundation as it brings together the world of database systems and the Business Rules Approach under a unified paradigm. The approach presented in this document is in many regards based on this work of C. J. Date.

2.5.4 The work of Alan Kay on reinventing Personal Computing

The most recent work of computing pioneer Alan Kay which tries to re-establish personal computing by redefining the software stack it runs on from the ground up. In his work, "Steps Toward The Reinvention of Programming: a Compact and Practical Model of Personal Computing as a Self-Exploratorium" [49], he summarizes his vision. According to this approach, software systems should possess the following properties:

Self-Explanation and Self-Disclosure: The system should be able to explain each error or operation to the user. Consequently, the system's operation should be transparent to the user.

Self-Similarity: All Objects Are "The Same" and Recursively Embeddable. This property breaks down the walls between local and remote components by modelling the local part of the system as a network of components itself. This allows the infrastructure in place for the distributed aspects of the system to be reused for the internal operations therefore simplifying the design of the system considerably.

Programming in "Meanings": From the same paper,

"A newer idea that is moving towards the mainstream is that specifications should be executable and debuggable. We want to go even further to "ship the specifications"

This principle is what enables self-explanation and self-disclosure. A system built on executable specifications can point to these specifications in order to justify its operation and interactions with the user.

2.5.5 Naked Objects

Finally, another approach to the problem of simplifying information system development, coming from the field of object-orientated programming is the 'Naked Objects' pattern and associated framework. The pattern was proposed by Richard Pawson in a series of publications including his doctoral thesis in 2003 [50]. The naked object pattern is based on three principles.

Behavioural completeness: All objects in the domain model are to be modelled in a way that encapsulates all relevant behaviour. According to the naked objects pattern, this is the only necessary code required to model an information system.

Expressive Systems: The naked objects framework uses the code of the domain objects in order to generate the entire information system, including the persistence layer and user interface. The interface generated however is not like the usual process oriented interfaces but rather gives the user direct access to the objects (hence the title of the pattern) and their associated functionality.

Users as Problem-Solvers, not Process-Followers: Giving the users access to the objects of the domain model rather than predefined processes requires the newly empowered users to learn a new way of interacting with the system. They must access the objects in order to modify the information they need which means they must first understand the domain model. This increased learning curve has been acknowledged by the creators of the pattern who recognise it may not the most suitable interface for every possible information system.

2.6 Other Relevant Efforts

In this section various approaches are covered, that even though they have not directly influenced the work presented, have many similar elements and therefore offer support to the path taken.

2.6.1 Database management Applications

When discussing the relational database, no mention of graphical user interface was made. This is because the typical RDBMS is not focused on the end user but rather is built for programmatic access. This creates a market for third party vendors to provide graphical interfaces for many popular database products. SQLyog[51] is a very popular open-source front end for MySQL[52] databases. It uses the schema data as described in SQL-DDL and makes the best attempt it can to automatically generate a user interface. Further than that, Navicat[53] allows users to customize the front end by entering metadata, additional to the information contained in the DDL. This allows usable applications to be created from little more than a database. Finally, PHPMyAdmin[54] carries out the same task but creates a web interface instead of a desktop one. In a sense, the combination of these features can be seen as a primitive web-based generative information system that allows only primitive, atomic actions to its users.

2.6.2 Web 2.0 data management applications

A different approach to generative information systems is offered by a number of Web 2.0 applications. These applications attempt to make the construction of an application as simple as possible so as to make it accessible to the average office user. Although they invariably store their data in some kind of relational persistence layer, they hide this from the end user by offering declarative modelling tools that operate on the level of terms and relationships instead of talking about technical concepts. Additionally, they can handle more complex schema information than what a relational database can. The limitation of these tools is that past the data modelling stage, they either run out of expressive capabilities or resort to imperative scripting solutions for the next development stages, therefore reverting to the current development paradigm. Nevertheless, their progress on declarative modelling is worth discussing and observing as examples of where a branch of the industrial mainstream is headed.

DabbleDB[55] is a very interesting application that offers perhaps the most advanced capabilities in this area. An innovation is their capability to use a spreadsheet as a starting point on top of which the user adds metadata to slowly move the flat spreadsheet structure to a more database oriented structure while abstracting the technical information away. Another quite innovative aspect is the

hint to a declarative user interface paradigm [56]. More specifically, they include a feature where the user can upload a logo to be used for determining a suitable interface colour scheme. In order to determine the colours that the user interface will adapt to, the logo is analysed and by using colour theory, a suitable colour combination is extracted and applied to the user interface. While the makers of the software do not explicitly mention declarative user interfaces, their application is a very good example of a move in this direction

Blist [57] is yet another start-up attempting to create an online office productivity tool that utilizes declarative elements. An interesting aspect is its claim to be merging the database and spreadsheet worlds in a unified paradigm. In fact it goes further than that and includes a rich type system that determines the integration with the data from the user interface down to the persistence layer. This feature allows users to create columns in their spreadsheet/database that have one of many types that can be usual database primitives such as numbers or strings but can also be higher level types such as star ratings, pictures and web links. These types have an equivalent presentation layer that utilizes the semantics of the types to offer an impressive user experience. Other than that, Blist is quite a limited product, not aspiring to create a full information system but being content with an intuitive but basic data management solution.

Similar to DabbleDB and Blist are products such as Iceberg [58], Zoho Creator [59] and the discontinued AppML framework [60], each one offering its own innovative variation on a similar theme.

2.6.3 Web Frameworks

With the increasing prevalence of web programming, a new class of tools has emerged. These tools, sometimes called web frameworks, aim to automate one or more aspects of web programming. They can aim to automate persistence, user interface matters, process control, or a number of other areas at the same time. Some web frameworks integrate smaller ones to build complete solutions that claim to drastically reduce development time for a web application.

A currently very popular web application framework is Ruby on Rails [61]. This framework contains a number of features that are interesting from the viewpoint of a generative information system. One such feature is the declarative domain model that it uses. A Ruby on Rails application is based on a domain model that the programmers define and which is used to auto-generate the database structure. In this way the programmers don't have to interact with the database at all. Another interesting feature is what is called 'scaffolding'. Based on the domain model, forms are created that allow manipulation of the data in the database through a web browser. By combining the two features together, a very basic web application can be built entirely through the domain model, without getting into imperative code. Finally, Ruby on Rails, with its latest version 2.0 has integrated full support for REST. At the same time, support for WS-* web services has been removed from the built-in libraries. These three features can be said to have equivalents in the generative information systems approach presented here.

2.6.4 Executable UML

Another interesting effort is executable UML (xUML) [62]. xUML uses a subset of the UML notation in order to allow for unambiguous specifications of information systems. A platform then uses these specifications to produce a working application. In this regard xUML is similar to the proposed approach. The weakness of xUML is in its eventual need for imperative code, usually in the form of

an Action Language, which it requires in order to handle more advanced scenarios. This is an aspect that is not desirable as it may present obstacles to business users.

2.6.5 Collage

Also of interest is a project by IBM research named Collage [63]. This project is interesting in its focus on declaratively creating distributed applications. Its usage of the Recursive MVC pattern and RDF as a unifying data model are design choices that can be useful in a possible implementation of generative information system. Especially the usage of Recursive MVC is reminiscent of the modelling of a generative information system as a digital ecosystem of other information systems and the related self-similarity principle of Alan Kay's work.

2.6.6 DSLs

Domain Specific Languages (DSLs) [64] are said to be a fairly large step forward for business users. The main idea is that once a program is written, key elements are extracted and exposed to business users through a translation layer that makes them more similar to natural language than computer code. Depending on the level of implementation, this can be done by utilizing simple regular expressions or something as complicated as a dedicated parser. The relevance with the work presented here is that the prevalence of DSLs proves the usefulness to business users of even a very crude interface to control the behaviour of the program.

2.6.7 Adaptive Systems

Adaptive systems [65] as Peter Norvig and David Cohn called their work, is a field of artificial intelligence that deals with creating software that can adapt to the context that it finds itself in and respond appropriately in situations unforeseen by its creator. Unfortunately, other than the referenced paper, not much is public about their work. The company it was done for, Harlequin Systems, has since been bought and the fate of the project is unknown with its main contributors moving on to other projects. While the work mostly deals with agents and was discovered relatively late in the process of this PhD project, a lot of inspiration can be drawn for future extensions, such as the emphasis on probabilistic reasoning and the aspects related to the performance of the system as a software component.

2.7 Conclusion

Generative Web Information Systems are a composite concept. Some of the component meanings can be inferred from the name itself, others only by keeping in mind the above discussion. An application of the concept of generativity to information systems, they are motivated by the conceptual paradigm of Digital Ecosystems but pragmatically implemented to be usable on today's World Wide Web. The declarative approach to business modelling and radical rethinking of the information system are the key elements that differentiate this approach from others that have tried and failed to achieve similar goals in the past. Of the many approaches have been proposed that approach the concept of a Generative Web Information System, none so far has stretched to cover the full scope of what a Generative Web Information System entails. By taking inspiration from the successful decisions taken by these efforts while focusing on the generative information systems vision has provided substantial input to this PhD project.

3 Stakeholders, Requirements & Foundations

Having surveyed the landscape of information systems and the efforts that are relevant to Generative Web Information System architecture, we can now lay down some foundations about the approach that will be taken. Initially, the stakeholders of an information system and the requirements of each of these will be described. Following from the requirements, some initial design principles and key technologies will be established based on which the architecture will then be developed.

3.1 Stakeholders

In order to proceed with defining our generative information system architecture, it is useful to first conceptualise the information system in more detail. The stakeholders are divided into the primary and secondary categories. The primary stakeholders are these which the system has been built to aid and be used by, specifically the owners, the users, and the partner systems. The secondary stakeholders are these which contribute to the system's operation but are not served by the information system, such as business analysts and developers. The secondary stakeholders are usually hired by the owners and their part should greatly be reduced but not extinguished in a generative information system, with some control being transferred to the primary stakeholders and enforcement of proper separation of concerns.

The ultimate stakeholder of an information system is its **owner**, who funds its construction. The owner may be a single person or it may be a board of directors of an enterprise. In any case, the owner should be an entity capable of making final decisions about the information system, as it exists primarily to represent their vision for the operation of the business. The role of the owner is to set the vocabulary and data models as well as the behavioural requirements or constraints on the information system. In other words the owner defines the context in which the system operates, the additional constructs which it interacts with, and the allowable or obligatory behaviours within this context.

More often than not, the owner does not monopolize the interaction with the information system. A number of **users** are given access privileges so as to manipulate the information held therein. These users may be internal users (e.g. employees in a business) or external (customers, suppliers, partners etc.). These users of the system have a somehow limited access to the system through policies as dictated by the nature of their relationship with the business and the task that they are supposed to carry out.

A special kind of user is the **partner system**. The partner system is a remote information system that wishes to interact with the local system. This system can in one sense be considered isomorphic with a user. Many of the same principles apply. The partner system has specific elements that it can interact with and specific policies governing those interactions. However the partner system is an automated agent. As such it is not able to dynamically adapt to changes in the local information system the way a human user would respond to changes in the user interface. For this reason, the relevant metadata must be made available so that automated agents can adapt their behaviour dynamically.

A different kind of stakeholder is the **developer** of the system. This party's role is to stand between the business and technical layers of the architecture and perform the translation necessary. The hardware architecture is agnostic to the business model of a specific service and at the same time the business owner is not interested in the technical workings of the underlying infrastructure. The implementer's role is to bridge the two spheres of concern. The way in which this is done currently, the explicit transcription of requirements into custom code, is problematic as it impedes system adaptability. However, even in a radically different architecture, there is still a role for a technical mediator.

Finally, current information systems require a business **analyst** as a bridge for the communication between the owner and the implementers of a system. One part of the analyst's task is to help the owner express the requirements. Many times the owner's view of the system will not be adequate or detailed enough therefore holding potential ambiguities. Other times the owner's view will be outright inconsistent. In these cases the analyst must help the owner first streamline and describe the business itself to an appropriate degree. The second task of the business analyst is to design a system, including the appropriate processes that will satisfy the owner's requirements and communicate this design to the implementers. Additionally, during the development process the analyst's task is to constantly align the project's progress to the owner's objectives.

In the day to day operation of the system, the most important roles are those of the owner and the user and partner organizations if any. These are ultimately the stakeholders whose productivity and cooperation the system aims to enable. The distinction between these categories can sometimes become blurry and this is an aspect that must be taken into account in any information system design. On the one hand, the owner of the system is almost always a user as well. It is very difficult to conceive a system that is defined by someone who does not have a need to contribute to the information being held therein. On the other hand, when the user or partner is granted access to information within the system, especially when there are creation and editing privileges involved, the user can exercise a certain amount of control, and therefore be considered a type of owner of said information. This ownership may extend to the user making the data available to third parties and defining additional logic or constraints that affect interactions with the information made through the user's access. Although this type of ownership is subject to the ultimate owner's granting the relevant privileges, it is still an ownership scenario that must be taken into account.

3.2 Requirements

Having described the stakeholders of an information system, we need to plot the requirements each one of them would have of a generative information system.

A prime requirement shared by all stakeholders is to allow each to carry out their function without needing to defer to other stakeholders. This is an application of the separation of concerns principle. The generative information system as described in this document is built to take this principle much further than the current mainstream information systems. In fact, the separation of concerns is seen as the primary requirement for generativity and therefore is pursued with priority over other requirements.

3.2.1 Common Requirements

As discussed in the introductory section, the overarching aim of the generative information system is the **separation of concerns** between the stakeholders. This is a requirement that cuts through all stakeholder groups. The owners should be able to specify the logic of the information system without needing the aid of the developers. Developers should be able to add capabilities to the information system without applying them on a case-by-case basis. Users should be able to interact with the system in any allowable way without the intervention of the owner or the developers. Partner systems should be able to continuously integrate without manual intervention.

3.2.2 Owner

The owner is the main beneficiary of the information system. It follows that most requirements revolve around the owner's needs. The owner is served by making the system evolution process as agile and costless as possible. This is a general goal within information systems and a specifically aimed advantage of generative information systems. Additionally, we seek to allow the system to easily find and connect to other systems and also be discoverable over a digital ecosystem by users who seek a service similar to the one offered, again to the benefit of the owner. Symmetrically, the owner should also be able to compose services from the digital ecosystem as parts of the information system itself.

In the generative information system, we seek to directly **execute the owner's specifications**. This goal forces us to aim for a modelling medium that is both accessible to the owner and also allows a computer system to extract the meaning embedded in it. We therefore need to express a number of requirements relevant to the modelling language and methodology. The modelling methodology that will be used needs to allow for expression of requirements in a way that requires no non-essential information and also does not require the same information to be stated in more than one place. Enforcing this requirement will give us a modelling methodology that drastically reduces the possibility for inconsistency and error. When non-essential or predictable information is required, sometimes called boilerplate code, the possibility that this will somehow be entered incorrectly arises. Additionally, we need a methodology that will not require redundant repetition of information. Each element of specification should be stated only once. Besides the obvious multiplication of effort in restating the same information multiple times, the possibility for error, inconsistency, and difficulty to update also appears.

At the same time, the modelling language used should allow a machine to extract the meaning intended by the owner. In order to achieve this, the language must **allow unambiguous expression** of exactly one solution. If the potential for ambiguity exists, then the system must chose among multiple alternative interpretations and the choice made may not always be predictable or consistent with the owner's desires.

Finally, the modelling paradigm must allow the owner to audit the specifications at any time. When the specifications are directly linked to the operation of the system, this equates to the elimination of the distance between the owner and the operation of the system. Ideally, the owner must also be able to write or edit the specifications without assistance, but at the very least the ability to **audit and verify the intent** in the specifications must exist, even if external aid is required to make changes.

Moving forward to the connectivity features of an information system, the system is expected to be able to **operate within a digital economy**. Consequently, the ability to interoperate with other information systems and be able to handle changes in either end of the connection smoothly is desired. Additionally, the system must be able to expose the business's services within the digital economy and have these services be discovered by interested parties. This of course does not apply to every instance of an information system but is an ability that is required as a general principle to be utilised when required.

3.2.3 User

The user group consists of anyone who interacts with the system through a user interface. Since users are exposed to the system on a daily basis, they are the most vulnerable to productivity loss due to misalignment of the planned use cases with their real needs from the system. The generative information system should be capable of **handling unforeseen requests** from the users by implementing a radically different interaction style, especially in environments where the user is expected to accomplish variable tasks through interaction with the information system.

Frequently, the users get alienated from the information system due to its opaque nature. The system's behaviours are a mystery to the users who react with returning to pen-and-paper solutions and general aversion to computerised information systems. Due to this, the use of a modern information system by unspecialized users is typically problematic and specific training expenses have to be incurred. In this case, the users get accustomed to a specific interface and set of capabilities, further constraining the information system's ability to change. This brings to the forefront the need for **transparency** of the information system to the user. The user should always be able to determine what the reaction of the information system was, and how this response was justified.

3.2.4 Partner Systems

Partner systems are automated information systems that want to interact with the generative information system. Symmetrically to the requirement of the owner, partner systems want the system to be **discoverable** within a digital ecosystem so they can discover the most relevant service to each query. As Metcalfe's law states [66], each new participant in a network increases the value of the network for all participants.

Past the discovery state, the information system should provide adequate information so that the partner systems are able to perform **automated integration and synchronization**. This means that the system should describe its services in a machine-readable format so that partner systems should be able to interface in the first place, but also keep adjusting to changes in the service over time.

While partner system considerations may seem irrelevant to information systems that are not ecosystem-facing, it should be kept in mind that singular information systems are modelled as ecosystems of interconnected information systems themselves. This makes discovery, integration, querying, and transaction considerations relevant for all instances of generative information systems.

3.2.5 Business analyst

The business analyst is a role whose very existence is based on the structure of the current development process. In a situation where the owner is much better connected to the information

system, the business analyst role would either be advisory or nonexistent depending on the capabilities of the owner. When there is no development team and software lifecycle to manage, the only remaining task is to assist the business owner in expressing, clarifying, and disambiguating the requirements. It follows that an owner who is capable of expressing unambiguous requirements should not require a business analyst, as an owner who runs an 'analogue' business does not require constant assistance from a business consultant. The business analyst has therefore got no stand-alone requirements, separate from those of the owner.

3.2.6 Developer

The requirements of the developer must be taken into account when they are related to the requirements of the primary stakeholders. For example, should developers request a familiar programming environment or support for a well-known technology, this should not be taken into consideration if it contrasts with the requirements of the owner. The development of the generative information system is an area where things work radically differently from the traditional software development paradigm. Even so, developers have an active part to play as owners are not expected to replace programmers or interface designers. Therefore what is required to include the role of developers in the generative information system is an **extension mechanism** that can incorporate assumption-neutral solutions to generic problems that can then be invoked from a declarative modelling paradigm.

3.3 Principles

Having documented the requirements that our stakeholders have of the generative information system, the next section presents the design principles that are being utilized in order to fulfil them.

3.3.1 Decoupling of Constraints from Goals

Traditional business processes can be seen as a pre-specified sequence of steps to satisfy a specific goal while adhering to contextual constraints. In order to allow both higher-order modelling by owners and the pursuit of valid but unplanned goals by users, the constraints and goals need to be represented separately instead of being inseparably compiled together in a process form. This yields the added benefit of deferring the setting of the goals to run-time rather than deciding on the reachable goals at design-time.

The goal/constraint decomposition is reminiscent of the mantra of logic programming which states [67] that:

Algorithm = Logic + Control

In this formula we can see that the algorithm is analogous to the business process while constraints correspond to the logic element. What is missing is the control part. The control component is defined in [67] as:

"...the control component only affects [the algorithm's] efficiency. The efficiency of an algorithm can often be improved by improving the control component without changing the logic of the algorithm."

The equivalent of the control component must therefore be a component which actualizes the constraints of the business logic while being able to evolve independently of them. In our case, the platform that executes the rules against the dataset and determines the system's behaviour accordingly plays the role of the control module. Applying the approach of logic programming unifies

a number of generative information system influences under a uniform paradigm. Modelling constraints independently provides an ideal use case for logic-based business rules which are intended to encode an atomic declarative business policy rather than an imperative sequencing step in a business process. The connection with databases is made even more explicit in [67]:

"A similar thesis that database systems should be regarded as consisting of a relational component, which defines the logic of the data and a control component, which stores and retrieves it, has been successfully argued by Codd"

Additionally, the radical shift in user experience draws from the Naked Objects framework which sees users as Problem Solvers, not Process-Followers.

3.3.2 Minimal Information

A trade-off of using more expressive languages and abstractions is that more code or information is needed to achieve a basic objective. This is a widespread criticism for the JavaEE platform for example. Whereas there is great power to harness, the amount of work required to get to an elementary result is too great compared to simpler frameworks which may not have as much potential power. The principle of minimal information seeks to identify the least amount of information that is necessary to unambiguously specify an information system. Inspiration can be drawn from the 'convention over configuration' and 'don't repeat yourself' (DRY) idioms found in the Ruby on Rails framework.

'Convention over Configuration' states that if an aspect of a system is not specified, the conventional behaviour will be assumed. This allows complete systems to be built simply by specifying the aspects of their operation that diverge from the conventional rather than starting from a blank slate and specifying everything each time. This principle is necessary to keep the amount of information used to describe a generative information system low, while providing as much expressive power as necessary for a given task. Boilerplate code and unnecessary information serves to distance an owner from the information system by creating a need for a specialist to manage the additional complexity.

DRY is a principle that encourages succinct expression of requirements. It mandates that each piece of information should be expressed in only one place and referenced wherever necessary. Needless repetition of information such as code is vulnerable to inconsistency and introduces resistance to change. While procedural programming and object orientation have been motivated by a similar pursuit, they have not been able to wipe out redundancy. The declarative paradigm, with its focus on 'what not how' descriptions, may be better suited to fully apply this principle.

While both idioms are used in Ruby on Rails, the intent is to build a system that can take their implementation much further, as with separation of concerns.

3.3.3 Self-Explanation

As transparency is of paramount importance, the principles of **Self-Explanation** are adopted as detailed in the work of Alan Kay. At the owner's level, transparency is pursued by allowing the owner to audit and verify the logic that determines the behaviour of the system.

At the user's level, when there is a logic violation we draw from the business rules approach which stresses that 'the rule should be the error message' therefore directly exposing business logic with

user interaction. A logic violation occurs when an information alteration is itself inconsistent with the business logic of the information system. Contrarily, a data violation occurs when an alteration to the information system state is legal on its own, but in the context of the current state, causes an inconsistency. In this case, additionally to the elements of business logic that are potentially violated, the potentially conflicting data are presented to the user. As part of the effort to be self-describing and accessible to stakeholders, a natural language approach to modelling is also preferred.

3.3.4 Self-Description

In order to facilitate exposure to a digital economy and seamless integration with partner systems, the information system must provide self-description of the services it offers. To accomplish this, an owner-approved subset of the business logic must be published as a description of the service.

The amount of logic published is a trade-off between efficiency and competitive advantage. A high amount of logic published allows potential clients to better filter their requests and therefore avoid needless queries. For example a loan service that publishes the fact that it does not grant loans to clients over 60 years old saves its servers the burden of rejecting unfit claims since the clients can pre-vet their requests. On the other hand a service that publishes a high amount of its logic as a description may be in danger of replication by competitors if the competitive advantage of the service lies in the logic itself. Not publishing however may not be an iron-clad defence as interrogation may reveal the internal logic of a service as the Search Engine Optimization community currently does with Google's ranking algorithm.

3.4 Key Technologies

Having defined a declarative information system and how it fits within the theoretical framework of a digital ecosystem, specifying an approach to declarative information systems requires instantiating the elements of the definition into more specific technologies. This allows separation of our architecture and implementation of a declarative information system from the general concept of a declarative information system.

3.4.1 Declarative Language – SBVR

A new and promising approach to human-computer interfacing is OMG standard Semantics of Business Vocabulary and Business Rules (SBVR [48]). As defined by [69], "SBVR provides a way to capture specifications in natural language and represent them in formal logic so they can be machine-processed". This allows users to be able to verify the requested service composition by directly reading the structured natural language used by SBVR which can then be parsed and executed by a machine.

Since SBVR is a way to capture specifications, there is no technical barrier to following a specific programming style. Specifically, one could conceivably use it to specify the structure of a process therefore conforming to an imperative programming style. However, the SBVR specification [48], quoting the Business rules manifesto [47] states:

"Separate From Processes, Not Contained In Them. Rules apply across processes and procedures. There should be one cohesive body of rules, enforced consistently across all relevant areas of business activity.

"Declarative, Not Procedural. Rules should be expressed declaratively in natural-language sentences for the business audience. A rule is distinct from any enforcement defined for it. A rule and its enforcement are separate concerns."

While SBVR can be used in an imperative style, the full benefits of the Business Rules Approach that spawned it can only be attained when it is used declaratively. Work on expressing information systems declaratively with the use of business rules has been presented in [8]. This work predates SBVR but provides essential guidelines to its utilization in an information system.

3.4.2 Distributed computing architectural Style – REST

Most approaches to service-oriented architecture in literature assume a foundation of Web Services built on the WS-* stack including specifications such as SOAP, WSDL, XML Schema and others. It is to be expected that this paradigm is used in literature given the activity around it in industry and standards organizations in the past few years. Essentially the WS-* stack represents the modern expression of RPC-style distributed computing, usually tunneled through the HTTP protocol which serves as a transport.

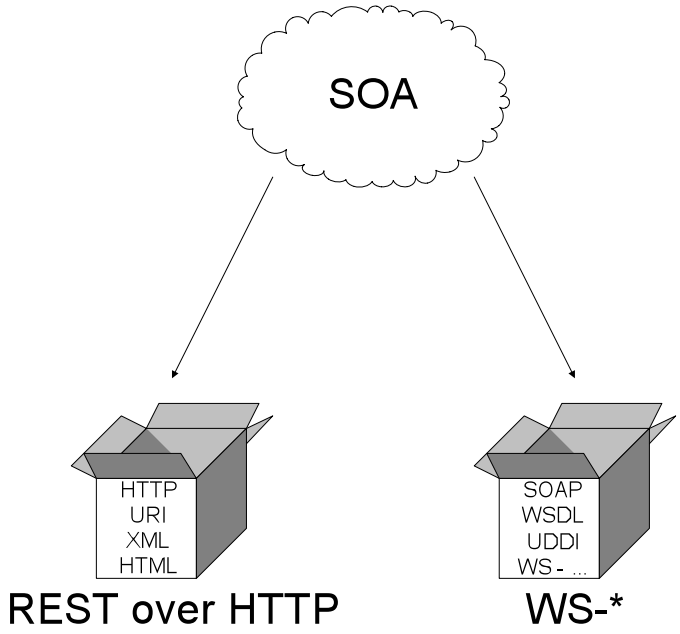

Figure 10. Different types of Service Oriented Architectures

However observation of uptake in the wider Internet suggests that a different paradigm is becoming a de-facto standard and recently industry seems to be noting this. The APIs of major web applications from Google[70], Amazon[71] and many others are built using a different architectural style termed REpresentational State Transfer (REST). Even WS-* heavyweights such as Microsoft and IBM are working on projects that utilize REST [72], [73], [74]. Also recently a number of standards have been approved that are compliant with and based on the REST style. [75], [76].

Contrary to what may be assumed, REST is not a new development. It was first defined in 1999 by Roy Fielding in his PhD dissertation [77] as a term to describe the architectural style used by the web. Consequently, HTTP 1.1 was released to better align the web with the principles of REST. While many have since championed REST as a competing web service paradigm to the WS-* stack, it has

only recently begun to be more seriously considered with the publication of works such as [78] and the apparent lack of expected adoption for WS-* technologies outside the corporate firewall.

The main concept of REST is the resource that is identified by a URI, the uniform resource identifier. Resources are to be accessed by a universal interface of well defined methods that should be resource-agnostic and therefore have the same, standard effect on all resources. In the case of HTTP 1.1, the methods include GET, PUT, POST and DELETE [79]. Other protocols such as WebDAV define methods such as LOCK and UNLOCK, suitable for transactions on resources. This Spartan interface is in contrast with the WS-* approach of defining a multitude of unique procedures the semantics of which must be rediscovered for each new service by its clients.

4 System Architecture

It is clear that SBVR was not intended [48] as a language from which to directly produce applications, at least to begin with. SBVR, as a declarative language, focuses on modelling the 'what' of a system, rather than defining the 'how' of its implementation. However, a given declarative model that is specified by its owner constrains the set of potential solutions that can implement it. Each new element of information added to the model reduces the number of compliant solutions. Within the set of compliant solutions, if two implementations have a difference observable by the owner, such that one is acceptable and one is not, then the model is underspecified. It must therefore be enriched with the additional information that retains the acceptable solution while excluding the unacceptable one. By iteratively repeating this process, we can arrive at a model that identifies only potential implementations that are acceptable to the owner. In practice, the specification can only identify acceptable solutions at a level of granularity afforded by the expressivity of the language it is written in. With this caveat in mind, we use SBVR as the best balance between expressivity and user-accessibility (through SBVR Structured English). Large part of this PhD project is to explore the extent to which solutions can be automatically produced by operational rendering of SBVR. Possible limitations that are encountered in expressing the specifications can act as feedback to the language design itself.

Current approaches to producing applications from SBVR models treat it as a workflow or code generation problem. This inevitably results in facing the tension between the declarative and imperative programming paradigms. When generating code, processes must be defined and programmed, however such processes are not natively defined by SBVR and this is so by design. Therefore attempts at code generation [80] must either arbitrarily select the processes that must be implemented, supplement SBVR with a workflow definition language such as XPDL and BPMN or extend SBVR itself to make it capable of specifying processes. The latter results in models that use SBVR as a verbose process language attempting to do what visual alternatives achieve far more concisely. Even then, the code that they orchestrate is still missing, at which point they revert to the need for a human programmer to fill in the gaps, a problem also faced by model-driven approaches like Executable UML in the past.

The alternative, which is pursued in this thesis, is to treat the model itself as the code to be executed and interpret it at run-time, possibly caching any decisions that can be reused. We view the static constraints of an SBVR model as defining the possible worlds (or states) that the data of an information system can describe. Additionally, dynamic constraints define the allowed transitions between these states. From this starting point, we explore how information systems can be generated, in the general pattern described by C.J. Date [8].

Figure 11 visualises the conceptual architecture that this chapter focuses on. The SBVR model, composed of terms, fact types and rules is what defines the two run-time aspects of the system, interface and persistence. The interface follows the RESTful principles and is designed to operate over HTTP, addressing all elements of the model and their instances as resources. The model also determines the structure of the database, through a mapping of the SBVR meta-model to the relational meta-model. User interaction takes place on the horizontal axis, originating at the HTTP

API, getting applied to the data, and if any rule violations occur, the update is rolled back and the violated rule returned as an error. The following sections go into details of the mapping from SBVR models to a relational schema, an HTTP API, the implementation implications of SBVR's modalities, as well as process-like behaviour within this system.

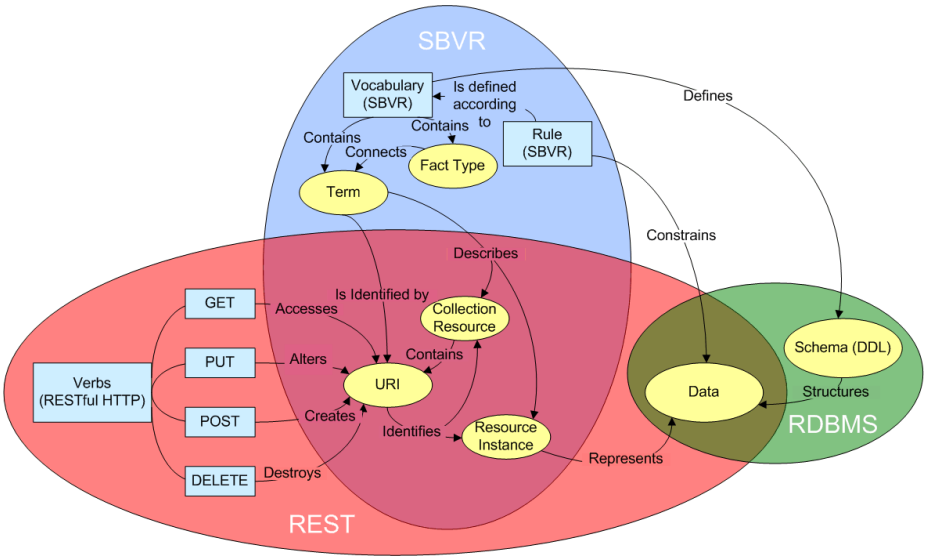

Figure 11. Connections between REST, SBVR and Relational Databases

4.1 Using Relational Databases for Persistence

While an SBVR model is an abstract construct, it defines the space which instances of terms and fact types are allowed to occupy when the model is itself materialized into an information system. This structure can be made explicit by extracting it and imprinting it onto a relational database. Relational databases, besides being the dominant persistence technology for information systems, are also an excellent candidate for persisting SBVR-based information systems because of their declarative nature. Relational databases are interfaced through SQL, a declarative language that defines the data structure and queries for a relational database. When discussing data structure, we focus on the SQL data definition language (SQL-DDL). While relational databases do not exhibit the expressivity that an SBVR model can, it is feasible to generate an SQL-DDL data model from an SBVR model. In this way, the maturity and performance of the many SQL databases can be harnessed while simultaneously using the integrity constraint checking as a basic model checker. The more advanced cases will of course still need to be checked against the SBVR model directly. Generation of a relational database schema has been referred to previously in [81].

4.1.1 Generating a relational schema

A basic schema to persist data that populates an SBVR model can be made by simply representing terms and fact types as tables. By giving each term table an automatic 'id' field, we can then represent fact type instances as a combination of the relevant ids, as many-to-many relationships are currently represented in RDBMSs. A 'name' attribute can also automatically be added to each term table to allow naming of instances. A binary fact type such as type '**student** *is enrolled for* **course**' would be represented as a table with two fields, one for the student_id and one for the course_id. These two elements of information are enough to represent the fact that a particular student is enrolled for a specific course. The same pattern can be applied to other fact types, including unary fact types and n-ary fact types with n≥3. Rules would not be encoded on the schema,

but instead turned into queries that when run against a dataset would return a Boolean result on which depends on whether the rule is consistent with the data. The proof-of-concept that has been implemented for this PhD project uses the patterns discussed above to generate a schema. While the result is functional, a lot of expressivity and performance of the relational model are left on the table. The following section discusses the generation of an optimised schema that uses much more of the relational model.

4.1.2 Generating an optimised schema

To infer an optimised database structure from an SBVR model, we must follow a more involved process. To begin, we construct a graph where each term and fact type is represented as a node. The edges link fact types with the terms they build on. Next, we need to define the relationship between each of the nodes.

For Date [82], defining a relationship requires integrating two aspects, one for each party in a relationship. The single-perspective relationships that Date considers are at most one (0..1), exactly one (1), one or more (1..*), and zero or more(0..*). While more detailed relations such as 0..5 etc. could be considered, there are diminishing returns to increasing levels of granularity, but also such relationships are also beyond the expressive capacity of the relational model.

In our graph, the edges initially connect fact types with terms. Since each fact type instance refers to exactly one term instance for each link, the relations of interest are those from the perspective of the terms. In the absence of constraints, a term instance can be referred to by multiple fact type instances (facts). The default edge label is therefore zero or more (0..*). So the fact type '**student** *is enrolled for* **course**', in the absence of relevant constraints, would be represented as a node with 0..* edges to the **student** and **course** nodes.

When rules exist that affect the cardinality of an edge, such as 'It is obligatory that each **student** *is enrolled for* exactly one **course**', the relation between the term **student** and the fact type '**student** *is enrolled for* **course**' gets more constrained, in this case to an exactly one relationship (1).

Once the relationships are identified, we can begin to differentiate what will eventually become tables and what will be attributes for these tables. To begin, unary fact types, such as '**student** *is under probation*' become Boolean attributes of the term they are connected to, in this case **student**. This is because they unambiguously say something about the term they are connected to, each term can only have one instance of that value, and this value can only be true or false. Similarly, we instantiate attributes from binary fact types which indicate that the one fact type role has a designation in an attributive namespace for the subject concept represented by the designation used for the other fact type role (e.g. **student** *has* **name**).

For other binary fact types, the simplest solution would be to represent them as tables on their own and leave enforcement of the relations between the data items to the SBVR model execution engine which will wrap the database. However, databases are highly optimized and their integrity constraints checking could take a lot of the burden off of our implementation, reusing the mature RDBMS software. So to represent different relationships we have five patterns for generating the equivalent SQL-DDL schema fragment for two nodes A and B. These generally include generating tables and using the Primary Key (PK) and Foreign Key (FK) as well as Nullable and Unique to express the relations specified in the graph.

The patterns, for two tables/nodes A and B are:

- Pattern I: PK of B is a FK in A with Uniqueness Constraint and is Nullable. If B has no other attributes, instead of an FK it becomes an attribute of A directly.
- Pattern II: PK of B is a FK in A with Uniqueness Constraint. If B has no other attributes instead of an FK, it becomes an attribute of A directly.
- Pattern III: PK of B is a FK in A and is Nullable. If B has no other attributes, instead of an FK it becomes an attribute of A directly.
- Pattern IV: PK of B is a FK in A. If B has no other attributes, instead of an FK it becomes an attribute of A directly.
- Pattern V: Intermediate Table A_B with PK of A and B as FKs and joint PK. If either A or B have no other attributes, instead of an FK they can become attributes of A_B.

We then apply these patterns as specified by Table 1. The cells that simply specify a pattern directly are those whose relationship semantics are exactly expressed by the results. The cells that merely specify 'Use' of a pattern are those whose semantics are not directly expressible in SQL, so a looser approximation needs to be used, with the rest of the input validation needing to be done by applying the SBVR constraints directly. We can observe that these cells are the ones related with the 1..* type of relationship which SQL cannot cover. Finally the cells that specify reverse use of a pattern are simply those where the appropriate pattern is the identified pattern with B and A substituted for each other. Due to the approximate nature of the relational model, an enclosing SBVR model execution engine must have sole write access to the database, in order to maintain consistency of the data. Alternatively, triggers could be implemented within the database itself for the more strict constraints, however their database-specific syntax and the difficulty of identifying the violated rule advise against this approach.

Table 1. Appropriate database patterns to express fact types as relations.

Edge with A / Edge with B...	0..1	1	1..*	0..*
0..1	I	Reverse II	Use Reverse III	Reverse III
1	II	Same Table or Use V/II/Reverse II	Use V	Reverse IV
1..*	Use III	Use IV	Use V	Use V
0..*	III	IV	Use V	V

For n-ary fact types with $n \geq 2$, such as **Student** *is marked with* **grade** *for* **course**, the relational model does not provide any way to represent this relation between terms, other than creating a new table. So for any combination of edge labels connected to the fact type, we use a separate table having the primary keys of the relevant terms as a foreign key. Another aspect of the data model that needs to be considered is the data types for the stored attributes. The 'SBVR Meaning and Representation Vocabulary' [48] gives us a number of data types such as (quantity, number, integer, text) that can be mapped to SQL primitives. This however puts a strict requirement on the terms that carry a value to belong to a concept type that specialises one of these data types such that it can be inferred.

Terms that do not define a data type can still be represented as tables and defined in terms of their characteristics and connections. Terms that are defined to range over a fixed set of values, such as the term **grade** which could be defined as [**A** or **B** or **C** or **D** or **F**], can be translated as an SQL ENUM data type to avoid creating a new table. Finally, the issue of primary keys remains. While this could be inferred through attributes that have a uniqueness property or SBVR reference schemes, performance of the database may suffer when using textual keys. For this reason, each table gets an integer auto-incrementing id attribute added, which becomes the primary key of that table. This can be omitted when the table contains a unique integer, such as a code number.

With the steps discussed above, we can algorithmically infer a relational database schema from an SBVR model that uses most of SQL's expressivity to optimise access to the data. However, the expressivity gap must always be considered, and each new element of data that enters the database needs to be verified not only against the integrity constraints of the database schema, but also against the SBVR rules that are relevant to the terms related to the data item. For instance a rule that uses multiple fact types such as '*it* is obligatory that each **module** that a **student** *is registered for, is available for* a **course** that the **student** *is enrolled in*.', cannot be expressed within the database schema. The process described in this section could potentially be implemented within a model transformation framework. QVT would be a candidate due to SBVR's serialisability in XMI, but the transformation would also require the existence of a suitable XMI target for relational databases.

4.1.3 Converting an SBVR rule to an SQL Query

Given the expressivity gap that exists between the SBVR model on the one hand, and the relational schemas produced by both the naïve and the optimized conversions on the other, we need additional measures to verify the consistency of the information in the database. To accomplish this, we convert the rules to SQL queries designed to verify the state of the dataset by essentially asking the question: "Is this rule constraint consistent with the state of the database?" Figure 12 shows the conversion of one rule from our example into an SQL query. The produced query will of course vary depending on the schema that it is applied on. This means that the rule generation algorithm must take into account the optimizations that have been applied in generating the schema. For our proof of concept, we work from the non-optimised conversion presented in section 4.1.1.

The first step in converting a rule to an SQL query is to transition from SBVR Structured English (SBVR-SE) to SBVR Logical Formulation (SBVR-LF). A parser has been implemented that does exactly that for a significant subset of SBVR, and is described in detail in chapter 7. What is important to note, is that once the logical formulation is attained, it cannot directly be converted to SQL without some pre-processing. A few important insights must be presented before the whole process of conversion can be made plain. Initial attempts at conversion used first-order logic as an intermediate step, however SBVR's mapping to ISO Common Logic is incomplete in the specification, and certain features such as the at-most-n quantifiers would require an extremely wordy conversion which although feasible is impractical for our purposes. This mapping has been published in [83], however the path taken is not useful for a practical conversion. The crucial insight that allowed bypassing ISO Common Logic was that SBVR-LF itself can be treated as logic, even though this is not discussed in the specification.

The other vital piece of the puzzle is the little-known relational division pattern which uses the SQL EXISTS keyword. A relational division query to find out if, for instance, a pilot can fly every plane in a given hangar, would be written as:

```
SELECT DISTINCT pilot_name
  FROM PilotSkills AS PS1
  WHERE NOT EXISTS(
    SELECT * FROM Hangar
    WHERE NOT EXISTS(
      SELECT *  FROM PilotSkills AS PS2
      WHERE (PS1.pilot_name = PS2.pilot_name)
        AND (PS2.plane_name = Hangar.plane_name)));
```

The above query is notable for its logic-like structure and its usage of the NOT and EXISTS keywords that map to logical negation(¬) and the existential quantifier (∃). What it lacks is the universal quantifier(∀) which in face cannot be directly expressed in SQL. As a result, any logic structure to be mapped to SQL must have this quantifier eliminated. To accomplish this, we convert every ∀□ into the equivalent ¬∃□¬, directly on the SBVR logical formulation. An additional step is then required to eliminate double negations that may occur from the substitutions.

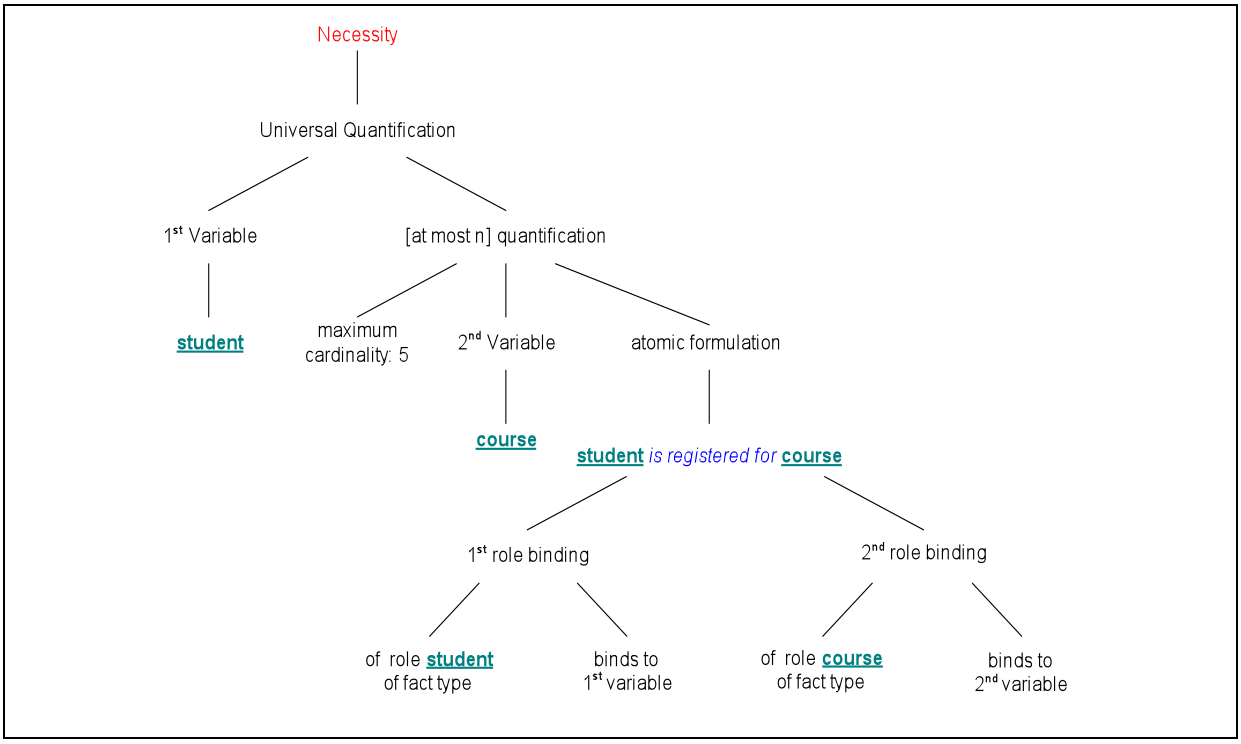

Figure 12. A rule encoded in SBVR Logical Formulation

In our example, the rule 'It is obligatory that each student *is enrolled for* at most 5 courses' yields the logical formulation seen in Figure 12. If we were to use a Propositional Logic-like syntax:

∀□:□□□□□□□□ ∃*0..5*□:□□□□□□□ □ □□□ □□□□□□□□□ □□□□ □

The at-most-n quantifier is introduced in the SBVR specification and can be seen above as seen above as ∃*0..5*. By eliminating the universal quantifier, we arrive at the following:

¬∃□:□□□□□□□□ ¬∃*0..5*□:□□□□□□□ □ □□□ □□□□□□□□□ □□□□ □

The negation of the at-most-n quantifier (¬∃*0..5*) could be converted into the at-least-m (where m=n+1) quantifier (∃*6..*) but it is not currently of any practical benefit to do so. The resulting logical formulation bears increasing similarity to the relational division pattern presented above.

What remains is to describe the mappings from the logical formulation constructs to SQL fragments. For the existential quantifier, we use the following SQL fragment:

```
EXISTS (SELECT *
          FROM <term> AS var<n>
          WHERE (...))
```

Here essentially the SELECT statement is used to introduce the relevant variable so that it can be later bound to.

The at-most-n quantifier maps to the following SQL fragment:

```
EXISTS (SELECT count(*) AS card FROM <term> AS var<n>
           WHERE (...)
           GROUP BY NULL
           HAVING card <= 5))
```

This fragment uses '`GROUP BY NULL`' to state that the '`HAVING card <= 5`' clause should apply to the whole result, as SQLite does not support a '`HAVING`' clause without a '`GROUP BY`' clause. Other quantifiers (existential, at-least-n, exactly-n, numeric range) have similar mappings.

Atomic formulations map to the following SQL fragment:

```
EXISTS (SELECT *
          FROM <fact_type> AS f
          WHERE var1.id = f.<role1_id>
            AND var2.id = f.<role2_id>)
```

Here, the `WHERE` clause declares the variable bindings. The assumption is that the appropriate quantifiers envelop the fragment such that the variables that are referred to have values to bind to. This is in line with the way the SBVR logical formulation is structured so a legally structured SBVR rule covers this requirement.

The negation operator simply maps to the SQL keyword `NOT`. Also, the '`SELECT`' keyword is prepended to every query.

Applying the above, the result of the conversion of the example rule 'It is obligatory that each **student** *is enrolled for* at most 5 **courses**' to SQL is:

```
SELECT NOT EXISTS(
  SELECT * FROM student AS var1
  WHERE NOT EXISTS(
    SELECT count(*) AS card FROM module AS var2
    WHERE EXISTS(
      SELECT * FROM student_is_registered_for_module AS f
      WHERE var1.id = f.student_id
      AND var2.id = f.module_id)
    GROUP BY NULL
    HAVING card <= 5))
```

Converting the more complex rule 'It is obligatory that each student that *is registered for* a module *is enrolled in* a study programme that the module *is available for*' into a query yields:

```
SELECT NOT EXISTS (
  SELECT * FROM student AS var1
  WHERE EXISTS (
    SELECT * FROM module AS var2
    WHERE EXISTS (
      SELECT * FROM student_is_registered_for_module AS f
      WHERE var1.id = f.student_id AND var2.id = f.module_id
      AND NOT EXISTS (
        SELECT * FROM study_programme AS var3
        WHERE EXISTS (
          SELECT * FROM module_is_available_for_study_programme AS f
          WHERE var2.id = f.module_id
          AND var3.id = f.study_programme_id
          AND EXISTS (
            SELECT * FROM student_is_enrolled_in_study_programme AS f
            WHERE var1.id = f.student_id
            AND var3.id = f.study_programme_id))))))
```

In this example we see a new structured English formulation which is signified by the keyword 'that'. In this role, the keyword 'that' introduces a restriction on the variable that precedes it. In our example, the variable 'student' is restricted by an existential quantifier that introduces the variable 'module' and ranges over the atomic formulation 'student *is registered for* module'. In terms of SQL, this translates to an injection of two more nested SELECT statements in the stack of nested statements that would otherwise occur. The same pattern occurs once more in the example.

It is interesting that this nesting formulation in SQL departs from the tree structure of SBVR's logical formulation. This is because while SBVR-LF is organised as a tree, latter nodes are assumed to have access to variables declared earlier in the formulation, even if the declaration was in another branch entirely. Formulating this as an SQL statement however returns an error as the SQL engine does not give access to variables declared on other branches. The solution to this problem is to nest every clause inside the previous one so that it has access to all variables previously declared, as seen in the example above.

4.2 Generating a RESTful interface from an SBVR model

When considering how SBVR models can be operationalised, there is the issue of shifting between consistent states. The verbs that SBVR allows, in the form of fact types, are declaratives that describe a state and as such cannot be used to actively cause a shift from state to state. Even dynamic constraints which are not yet natively supported by SBVR can constrain but not cause transitions. What is needed is an architecture that allows users to express, in a standardized manner, the changes they want to affect on the data of the information system. In contrast to the less disciplined Remote Procedure Call (RPC) style used in most WS-* standards, REST is based on a number of explicit architectural constraints that govern interactions. While linking URIs and terms brings SBVR closer to the web, the user is still without means to cause change in the state of the system. REST however indicates that resources should be manipulated through a uniform interface. In the case of HTTP this interface includes operations such as GET, PUT, POST and DELETE. The uniform interface is in fact the only way in which a client is intended to interact with the resources

that a service makes available. The rationale behind this constraint is that if the operations are insufficient to accomplish some functionality, which is an indication of the existence of more resources that need to be identified. This is in contrast with the programmer's instinct to overload the interface with additional methods as one would do in an object oriented environment where both endpoints are under the programmer's control. By constraining the interface to a fixed set of methods, the interface designer is forced to extend the vocabulary of the application. This is an insight that can directly reflect on the modelling methodology and reveals the benefit of considering the run-time behaviour at design-time. By forcing the modeller to consider the model as an executable artefact, accessible from a constrained interface, they may discover entire new areas that need to be modelled that would have otherwise been overlooked, leaving room for ambiguity.

A number of other constraints are in effect when the REST architectural style is considered, including statelessness and the 'hypermedia as the engine of application state' (HEAS). Statelessness instructs that each request to the server contains all the information needed for the server to understand the request without need for session information on the server side. HEAS requires that resources use links to point to each other such that the clients can incrementally discover the API and any change on a resource URI will not trigger a catastrophic failure of the client but rather a recovery procedure during which the client will rediscover the new identifier the same way as the original identifier had been discovered. These constraints form an architectural style that is state-oriented rather than process oriented. By focusing around 'be' and not 'do' type interactions, REST can be considered a declarative architectural style, very well aligned with the design principles behind SBVR.

A basic design principle of REST is that all important 'things' should be named, which perfectly echoes SBVR's term-orientation. Since SBVR vocabularies have a namespace URI, and terms are unique within a vocabulary namespace, it is trivial to assign a unique URI to each term. The following URI template [84] would be sufficient to accomplish this:

http://domain.org/{vocabulary}/{term}

However the question arises of what these URIs will produce as a response when requested from the application's HTTP server. Since the model-level term can be seen as a collection of data-level instances, we can return this collection as a resource. The Atom Publishing Protocol [75] is being increasingly used as a general-purpose format for representing collections and can be used to grant our interface with significant standardised functionality, such as exposing collections as standard feeds and editing collections or instances. Having discussed the issues of representing collections, we can now look at the instances. Following the unique identification patterns discussed in the relational schema generation, we can use a URI template such as the following to generate the URI for an instance of a term, like so:

http://domain.org/{vocabulary}/{term}.{identifier}

The serialisation of the content of a term can be subject to the content negotiation processes that HTTP specifies, but as a baseline, an XML or JSON serialisation can be assumed as a standard. Providing an XSLT stylesheet which defines the transformation of the XML to HTML can also aid towards readability by humans and set the foundations of a user interface layer.

Another fundamental constraint of REST is that of 'Hypermedia as the Engine of Application State' (HEAS). This specifies that URI-named resources should link to each other so that they can be discovered by a new client with minimal initial information, or rediscovered in case of URI change. This also naturally overlaps with SBVR's assertion that fact types connect terms. In this sense we can use the fact types as links to instances of the terms that the fact type builds on. So, the representation of each term must also provide links to URIs that represent the set of instances of the fact type that concern the term in question. A fact type of the form term-verb-term would be represented as:

http://domain.org/{vocabulary}/{term□}-{verb}-{term□}

If the verb consists of multiple words, those are joined with underscores to avoid encoding problems. It is worth noting that the characters dash (-), dot (.), and underscore (_) are all legal and have no special meaning under RFC3986 [85] that governs URIs. Instances of fact types can be identified by appending '.{identifier}' as with terms. This implies that each instance of a fact type has an associated identifier, independent of the term instances which it connects.

Filtering can be applied to the URIs of terms and fact types by appending the suffix '*filt:{property}{=|!=|~}{value}'. A semicolon (;) can be used to chain multiple filters in a single 'filt' suffix. The operator (=) indicates equality, (!=) indicates inequality, while the tilde (~) corresponds to the SQL operator LIKE. In a similar fashion, sorting can be applied to the URIs with the use of the '*sort:{property}({ASC|DESC})' suffix.

Due to the arbitrary nature of filtered and sorted URIs, embedding them in a hypermedia system seems at first glance contradictory. We have a resource that must link to a non-specific URI, representing not a single resource but a family of resources, one for each potential query. In an exclusively HTML-based world this could plausibly be overcome with the use of HTML forms. Drop-down menus could be used with which the user selects the operators and properties, and then another input, dependent on the data type of the property, to define the value that the property is constrained with. The form then converts, either directly or through processing via JavaScript, the input into a URI which is executed either with a GET or a POST operation. Besides the limited array of operations, (the upcoming HTML 5 standard will remedy this) this seems to be a workable solution for including in the hypermedia graph large families of resources whose number is large enough to make it impractical to enumerate and link to them.

The shortcoming of this approach is that it renders the queries inaccessible to the world of machine-oriented interactions, as automated clients have no means of discovering them. What is needed is a method similar to that of HTML forms that is also machine-friendly. The solution that has been arrived at is to define a new media type {x-application/query} which contains the results of a query, but also links to a JavaScript function that transforms a media type-defined set of parameters into a URI that can be executed against the service. This fits into the code-on-demand type of RESTful interaction. The script can be either embedded in the resource or linked to at another location. In this approach, the script plays the role of the form, transforming arguments to a URI. By introducing a new media type, the clients that implement it will know all the relevant semantics including the purpose of the function and the nature of the arguments that are to be used in it. Also, since the server controls the function, the URI-space remains in its control. The server can change the way it formats its URIs and, as long as it changes the transformation function correspondingly, the clients

should automatically adapt to the new format. Even if there are bookmarked URIs, the clients should know how they were produced in the first place so that they can retrace their steps in the case of breakage.

The querying infrastructure described above has been developed throughout a number of related MSc and undergraduate student dissertation projects [86, 87, 88]. Each of these has gone much further than the above in developing a querying language, but only the most concrete and necessary ideas have been embedded here, with more features to be included in the future as the need arises.

Table 2. Applying HTTP operations on Collections and Instances

	Student		Study Program		Course	
	Collection	Instance	Collection	Instance	Collection	Instance
GET	+	+	+	+	+	+
PUT		+		+		+
POST	+		+		+	
DELETE	+	+	+	+	+	+

Having defined identification of collections, instances and queries with URIs, we can now examine which HTTP operations apply on these URIs. Table 2 shows a basic application of HTTP operations on the resources defined by the student term.

Thus, knowing the URI of the student collection, we can not only see a representation via GET, but also create a new student via POST, identify a specific student via hyperlinks, and then modify the student instance through PUT or remove the student via DELETE. Access to data and ability to modify it is of course subject to authorisation from and authentication of the user making the data retrieval and modification.

4.3 Utilising Modalities

SBVR supports constraints of two different modalities, Alethic and Deontic. The SBVR specification [48] describes their difference as follows:

"Alethic modal logic differs from deontic modal logic in that the former deals with people's estimate(s) of the possible truth of some proposition, whereas deontic modal logic deals with people's estimate(s) of the social desirability of some particular party's making some proposition true."

In this sense, it can be said that the alethic model defines the map of the territory that the deontic model navigatesn. In order to analyse an SBVR model, we separate the rules into two models, alethic and deontic according to their modality. After checking each model for internal consistency, we infer the relationship between the sets of allowed states of the two models.

The interaction between the alethic and deontic modalities is not discussed in the SBVR specification, but in order to produce executable code, it is an area we must examine. Following the notation in chapter 10 of [48], we want to ensure that situations do not occur where $\mathrm{O}\mathrm{A}\ \&\ \sim\mathrm{A}$ for any proposition A. A proposition cannot be made true if it is not possible for it to be true, so the propositions deemed obligatory by the deontic model must be possible in alethic model.

If some propositions obligatory in the deontic model are not possible in the alethic model and all the propositions possible are obligatory, we consider the deontic model to be superfluous as it adds no information about desirability beyond what is known to be possible. If the states allowed by the two models are disjointed, we consider the overall model to be invalid, since the deontic model would only be aiming for states that are not deemed possible by the alethic model. If the two models partially overlap, the model can be executed, but the user should be made aware that a subset of the states deemed desirable by the deontic model are unreachable.

After inferring the relationship between the two sub-models, we take their intersection as the effective model which we attempt to execute. Differences in modalities mean that there may or may not be recourse to the model owner in case a specific rule is violated depending on whether it is deontic or alethic respectively.

The choice of modalities supported by SBVR has been made by soliciting expert opinion rather than being sourced from an objective aspect of modal logic. As such, it is understood that new modalities can be added if they are useful and do not overlap with the semantics of the existing modalities. As part of this PhD project, we have introduced an experimental modality of desirability. Rules under this modality do not represent hard limits, but rather soft targets which produce reminders but are not otherwise enforced. This new modality is not necessary for the operation of the system but is rather an extension that adds functionality. Therefore, a traditional SBVR model is still perfectly compatible with the Generative Web Information System engine being implemented. An example of the use of this new modality would be a rule such as 'It is desirable that at most 20 **students** *are registered for* each **module**'. There may already be a rule that places a hard limit on the number of students, such as 'It is obligatory that at most 25 **students** *are registered for* each **module**'. However, once a certain module has surpassed 20 students, the users will receive a message which they can discard. These rules can be used to provide additional soft guidance instead of producing error messages which cannot be bypassed. The desirability modality depends on the judgement of the human users who judge whether an alternative course of action is possible or whether this change of state is necessary.

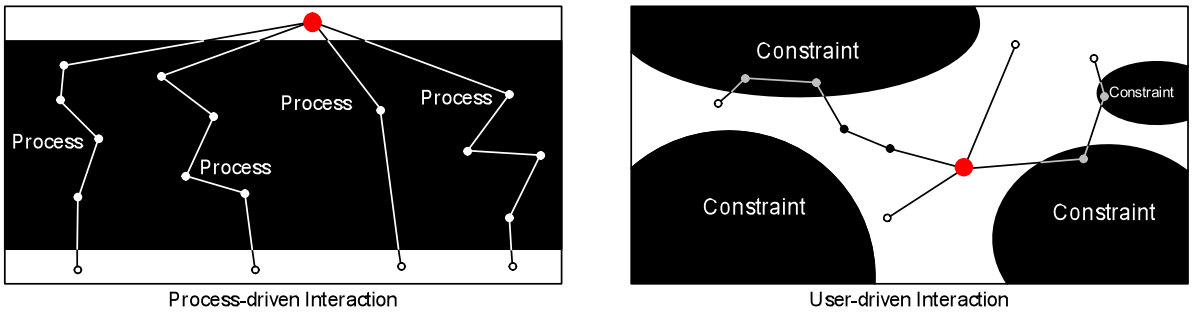

Figure 13. Visualising Process- and User-driven interaction models

4.4 Exhibiting Process-like Behaviour

The capabilities of the information system described so far are limited to satisfying sequential operations that satisfy the SBVR model over a basic RESTful API. However, fundamental to information systems is the ability to perform processes that make multiple state alterations as their result. On a first reading, there is a fundamental tension between processes and declarative

specifications, as processes focus on the 'how' rather than the 'what' which declarative models specify. To resolve this, we have been inspired by the motto of the logic programming community [67] which states that *Algorithm = Logic + Control.* By approaching the process as a simplified algorithm and the SBVR model as the logic, we can reformulate this as *Process = Model + Control*.

Perhaps the best way to explain how a Generative Web Information System exhibits process-like behaviour is by starting from traditional process-driven systems. In the standard case, an information system allows every user to execute pre-programmed processes which result is specific changes to the system's state. The principle is that 'everything that is not explicitly allowed is forbidden', with processes defining the allowed steps. During the course of the process execution, the changes made to the system are validated against the implicit model that the processes were designed against so that the resulting update does not leave the system in an inconsistent state. Note that there is nothing preventing a badly designed process from leaving the system in an inconsistent state under certain conditions. It is up to the process designer to make sure this does not happen. Assuming well-designed processes, the user can be certain that the interaction with the system is safe. This mode of operation however, depends on an underlying assumption, which is that the processes that are made available to the user at design-time, are the ones that the user will need at run-time. Any need for allowing unforeseen changes to the system state requires additional development effort. Also, updates to the business logic of the system require revision of all relevant processes, including some that may not be obvious. If this update procedure is not carried out correctly, the system risks being left with processes that assume different models and may lead to inconsistencies in the data.

The advantage that our system has is that there is direct access to a formalised model of the business. This allows a much more open approach that could be described as 'everything that is not explicitly forbidden is allowed'. In this approach, the user can request a state change, and the system can respond with the exact rule that the resulting state would violate. This is an application of the business rules motto 'The rule is the error message' which seems to be quite effective in our case. Notice that this mechanism can also observe the violation of dynamic constraints such as avoiding the progression of marital status from 'married' to 'single' instead of 'divorced' or 'widowed'. The user can then amend their request with additional operations aimed at mitigating the violation. As the process iteratively continues, the user will either realise that their request is untenable within the constraints of the current model, or they will formulate a request that satisfied both their requirements and the system's constraints. Our solution is to introduce the meta-process as seen in Figure 14, which at its core examines the state resulting by each action of the (authenticated) user, and determines whether it will result in the system being in a consistent state, one where no rules are violated. In case of violations, we return the rule that has been violated as part of the response.

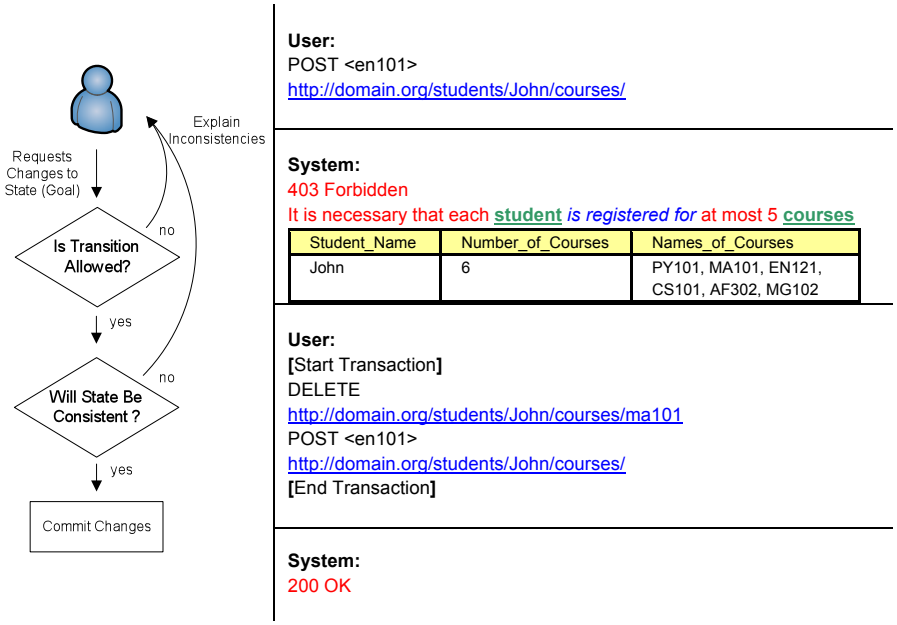

Figure 14. The meta-process control structure

This results in a declarative process-less system which can nevertheless exhibit process-like behaviour. Its main run-time difference with process-driven systems that is that it allows users to perform any allowable process instead of specifying at design-time the processes that the designers forecast will be useful to users. The design-time implication is that such systems are capable of naturally adapting their behaviour to changes in the model instead of requiring manual revision of their individual processes. The run-time result is that the user is able to intelligently respond to requests that its designers had not thought of. This allows the information system to adapt to the way the business evolves rather than constraining it.

5 RETRO: A RESTful Transaction Model

In the previous chapter, we have discussed the need for a transaction model to accompany the Meta-Process. When multiple operations need to be carried out with guarantees of atomicity, introducing transactions is the standard solution. However, since our system is rooted in the architecture of the web, we needed a transaction model that was congruent with it. However review of the literature revealed only fragmented mentions of the possibility for such a model, with no real candidates fleshed out, but also no real substantiation of the position that transactions are incompatible with the nature of the web.

We therefore have started research efforts in the direction of enabling RESTful transactions to take place over the web. The contribution of the resulting model is larger than simply enabling Generative Web Information Systems. RESTful transactions are useful in aligning with formal Web Architecture a number of use cases that happen over the web today, but which do so by bypassing the constraints of REST, and therefore, besides bearing the related impedance mismatch, giving up the related benefits a RESTful architecture offers.

The resulting model is useful for applications other than Generative Web Information Systems, so it is presented in an implementation-agnostic manner. That noted, the transaction model presented in this chapter is integral to GWIS architecture.

5.1 General usefulness of RESTful Transactions

Within the REST community, the topic of transactions in combination with REST is a difficult one to breach. This is in part because of the association of distributed transactions with the WS-* set of specifications, as well as the lack of a clear justification for the need of such capabilities. In our research so far, we have found that the RESTful approach significantly simplifies the transaction model compared to an RPC-based approach, simultaneously making it more powerful. The next section attempts to explain the need for transactions in a distributed system such as the web.

A common conception among the REST community is that if transactional needs arise, a resource should have been defined, the updating or creation of which will cause the desired modifications to the resources the transaction would have needed to modify. So in the case of transferring funds from one bank account to another, a solution would be for a new transferRequests collection to be created by the designers of the system that transferRequest resource representations are POSTed to. Each transferRequest representation would link to the source and destination account URIs, and also contain the amount to be transferred. This paradigm in its strict interpretation, however suitable for a range of problems, cannot cover all needs for transactional interactions in a distributed system.

In the common case of the shopping cart, a buyer adds items to a virtual shopping cart. The purchase of the group of items is then processed on checkout. In RESTful terms, the shopping cart and the items in it are expected to be identified as resources by a URI. The processing and purchase of the items in the shopping cart is then transactional as the semantics of ‘purchase these 4 items’ imply atomicity, being distinct and stricter than the non-atomic ‘buy as many of these items as possible’. In the case where these items are all necessary for a common purpose (e.g. a DIY project

or cooking based on a recipe), best-effort is not enough as missing one item is enough to render the purchase useless to the buyer. Taking the shopping cart metaphor a step further, in a physical store, an item added to a physical shopping cart is not available for anyone else to purchase in the interval between placing the item in the shopping cart and checking out at the cashiers. These semantics may not be practical for every online store, but in situations of high-value, limited availability, highly specific items (such as seats in a flight) and trusted buyers (such as travel agents), they may make sense or even be necessary. This implies a need for a locking mechanism to 'mark' the items as being considered for purchase by a buyer, and therefore not available to other buyers, at least for a given time window. As we elaborate the shopping cart model, we see it gradually becoming an ad-hoc transaction model with the need for atomic semantics and a locking mechanism. Regardless of the duplication of effort, if this model is not well designed, it will fail to implement the semantics correctly and therefore allow adverse effects contrary to user expectations.

Additionally, in situations where the items needing to be purchased are not all available from a single vendor, but there is still need for atomicity, such as a travelling arrangement containing a return flight, a hotel booking, and a car rental for the duration of the stay and perhaps a sightseeing tour booking, there is need for a standardized transaction model that can coordinate a purchase across multiple vendors without the need for an opaque intermediary, such as Expedia in our example. While the above examples focus around purchasing and actual monetary transactions, the computer science concept of transactions has far wider applications in distributed systems such as in many collaboration scenarios.

As is common with disruptive technologies, REST over HTTP is evolving to compete with WS-* in increasingly advanced usage scenarios such as Google's GData and the Sun Cloud API [89, 90]. The RETRO model aims to be part of the next wave of REST evolution by helping define a RESTful transaction model that is designed to operate over HTTP. To date, usage of REST has remained at the level of serial sequences of operations, each succeeding or failing atomically. While its advantages have made it the dominant web services paradigm on the web, the WS-* stack provides the only standard for transactions [91, 92].

5.2 Transaction Basics

The general term 'Transaction' has been introduced by Gray [93] and is defined by the four properties contained in the ACID acronym: Atomicity, Consistency, Isolation, and Durability. These properties guarantee that a system is maintained in a consistent state, even as transactions are executed within it concurrently. This includes the situations where one or more transactions fail to commit.

The fundamental property of transactions is that if one or more operations of the transaction fail, it should automatically undo all previous actions and return to the original consistent state. This property is called atomicity. A transaction that is started when a system is in a consistent state may make the state temporarily inconsistent, but it must terminate by producing a new consistent state. This temporary inconsistency may not affect other concurrent transactions, so as to maintain the illusion that each transaction runs in isolation. It follows that concurrent execution should not cause application programs to malfunction, which is the first law of concurrency control [97]. Finally, transactions should be durable, meaning that once a transaction is complete, its results should be persisted permanently.

When dealing with a sequence of transactions (one transaction executed at a time), each transaction starts with the consistent state that its predecessor ended with. If all the transactions are short, the data are centralized in a main memory, and all data are accessed through a single thread, then there is no need for concurrency. The transactions can simply be run in sequence. Real-world interactive systems however, often require execution of several transactions concurrently. Use cases such as distributed environments or dynamic allocation of resources to external developers illustrate this need.

5.3 A RESTful Transaction Model

The main motivation behind RETRO is to design a transactional model that can guarantee transactions with the ACID properties, while remaining within the constraints of REST over HTTP. To accomplish this, the model operates over resources and also every significant entity in the transaction model is made into a resource itself that is to be manipulated through the uniform interface of HTTP. To satisfy statelessness, a user locks resources serially, at which point copies of the resources are created within the transactional scope. The client can then manipulate those copies through the uniform interface as they would if they were interacting with an isolated service. Any resources that would be created as a side-effect of these manipulations can also be created within the transactional scope and reachable through hypertext links from the manipulated resources. The requests of the client are stored as resources themselves and become the history of the transaction. Once the transaction is committed, the new state of the locked resource copies is applied to the original resources atomically.

5.3.1 Creating a Transaction

Transaction Collection (Tc): To create a new transaction its representation must be POSTed to the transaction collection. The server should then return a '201 Created' response, along with the URI for to the new transaction resource. By applying the GET operation to transaction collection resource itself, a transaction owner will receive in a collection format links to the transactions they own as a response, after successfully authenticating.

Table 3. Available Operations for Tc

GET	Returns the collection of transactions owned by a user, after authentication.
POST	A client can create a new transaction by POSTing its representation. If successful, the server should respond with a '201 Created' status as well as the URI of the new transaction resource.

Our model uses a number of collections. The media type of these collections is not specified here, although the ATOM Publishing Protocol is being examined as a suitable collection handling format. However, any collection representation, as long as it covers the hypermedia requirements of this model, can be used.

Transaction (T): The transaction resource should be represented by a dedicated media type (e.g. application/x.retro-transaction+xml). It should specify the elements in Table 4 as well as the operations that can be applied to them.

Table 4. Elements of T

TransactionCollectionURI: The URI of the parent collection
OwnerURI: The URI of the owner's handle
TransactionLockCollectionURI: link to the transaction's collection of locks.
TransactionHistoryCollectionURI: link to the history collection
CommitTransactionURI: link to the resource that when POSTed to, executes the transaction
TransactionStatus (Active | Committing | Committed | Aborted): The status of the transaction, one of 4 options.

Table 5. Available Operations for T

GET	Returns the representation of the transaction resource.
PUT	Updates the state of the transaction resource.
DELETE	Aborts the transaction and deletes all the resources associated with it

These 3 elements identify the resources vital to the execution of a transaction. Any operation on a transaction resource should be applied only after authenticating that the client is the owner of the transaction. The owner can GET the transaction resource to locate the resources relevant to it, PUT to it to update its state or DELETE it to abort the transaction and delete all its subordinate resources. The transaction OwnerURI element should link to a URI that the server understands as an identity and can execute an authentication protocol on. Conceivably, this could be simply a URI on the server itself or even an OpenID [94] handle on a remote server. Finally, the owner of the transaction can commit it by using POST on the CommitTransactionURI. This pattern is akin to a button being pressed on a machine, inspired by a similar pattern used in Sun Cloud API [90, 95, 96]. If the request reaches the server successfully and the server is able to commit the transaction, it will respond with '202 Accepted'.

The same functionality could have been implemented by adding a TransactionExecutionRequests collection that an individual TransactionExecutionRequest resource can be POSTed to and lead to the execution of a linked transaction. The addition of an extra collection and all the associated resources however was deemed overkill as monitoring the state of the transaction can be done directly through the TransactionStatus element in the transaction resource and making individual requests into separate resources did not yield any real benefits. This design decision can be reversed however, is sufficient reason is found to do so.

Transaction Lock Collection (T-Lc): The transaction lock collection contains links to the locks that belong to a specific transaction. New locks can be created by POSTing a new lock representation to this collection.

Table 6. Available Operations for T-Lc

GET	Returns the collection of locks subordinate to a transaction
POST	A client can create a lock by POSTing its representation. If successful, the server should respond with a '201 Created' status as well as the URI of the new lock resource

5.3.2 Placing a Resource in the Transaction Context

Lockable resources (R) are the resources on which the transactions operate on. To satisfy the hypermedia constraint, every lockable resource should indicate it is lockable and link to the transaction collection. Also, ideally, any resource that can be served by an HTTP server should be potentially lockable, conditional on the resource owners' desires, regardless of media type. One way to do that would be by using custom HTTP header level. This is clearly a more general solution as it would allow resources of all media types to be lockable. If we want to avoid mingling with custom headers, we can focus on XML media types and opt for a XML fragment that can be included in an XML representation of a resource that provides the links HATEOAS requires to start a transaction. This approach could potentially be extended to other formats that can accommodate hyperlinks but probably not to binary files such as images or zip archives for which a different mechanism to indicate lockability must be devised. The inclusion of the following XML fragment indicates that a resource is lockable:

```
<lockable>
    <link rel="lock_collection" href="http://example.org/resource/locks/">
    <link rel="transaction_collection" href="http://example.org/transactions/">
</lockable>
```

Figure 15. (R) Example XML Fragment

Making available resources for locking is solely the responsibility of the server. However, the server must have the ability to keep resources stable once they have been locked. This means that if the manipulation of other resources will lead to the alteration of the state of a locked resource, those resources should behave as if they were locked as well, or at least to the degree that they affect the locked resource, to avoid causing consistency violations.

Lock Resource (T-L): The lock resource is represented by a dedicated media type (e.g. application/vnd.retro-lock+xml), and should contain the elements in Table 7.

Table 7. Elements of T-L

LockedResourceURI: a link back to the resource that this lock is applied to.
TransactionURI: a link to the transaction that controls the lock.
Type: "S" or "X" depending on the type of the lock.
PrevLockURI: a link to the previous lock in the lock sequence.
Timestamp: Server's timestamp when the lock was granted.
Duration: Indicates the interval that the lock has been granted for.
ConditionalResourceURI: A link to the representation of the resource that will come into effect once the lock is committed.
InitialResourceURI: A link to the representation of the resource at the time of locking. This copy will serve as an archive when the transaction is committed.

As expected, a transaction cannot lock a resource that is locked by another transaction. However, if two or more transactions want guaranteed read-access to a resource, they are not going to change the resource state and therefore this can be accommodated to increase concurrency. The type element can take one of two values, X or S, corresponding to the available lock types. X stands for XLOCK: eXclussive Lock, and S stands for SLOCK: Shared Lock. To place a new lock, the server must

authenticate the user as the owner of the transaction that is referenced by the lock. Table 8 shows the lock compatibility rules.

Table 8. Legal lock sequences

		Mode of Preceding Lock	
		Shared	Exclusive
Mode of New Lock	Shared	Yes	No
	Exclusive	No	No

The length of time of effectiveness that is granted to a lock is dependent on the maximum length of time that the server is prepared to grant a guarantee to the client. Once the duration of the lock expires, the lock is aborted. The lock resource can only be operated on by GET. Deletion a lock can only occur when a transaction is aborted or committed, to avoid violating the 2PL [93] protocol.

Table 9. Available Operations for T-L

GET	Returns the state and links of the lock.

The result of the standard GET operation on the locked resource does not change until a lock of type X is committed. In this sense, the locks and transactions are transparent to the GET, the result of which does not change until a transaction that affected the resource has committed. At that point it reacts as if a simple PUT was applied just then. This was a specific design objective. Meanwhile, PUT and DELETE operations return a '405 Method Not Allowed' HTTP response for the duration of a lock's effect. This behaviour maintains backwards compatibility, with the understanding that if a transaction-enabled client requires further guarantees on the future state of the resource, the client should seek to secure a lock on the resource. Consequently, the semantics of GET are unaffected, as a GET on a resource does not guarantee that the state will remain unchanged for any period of time after a response is sent. Regarding the POST operation, if a resource only has SLOCKs placed on it and the server can guarantee there are no concurrency conflicts such as changing the state of the resource, the POST operation can be satisfied.

Resource Lock Collection (R-Lc): The R-Lc contains all the locks that have been applied to a specific resource. Under normal operation, the locks should be in sequences that follow the compatibility rules stated in Table 8, rendering the transaction well-formed. The inferred rules constrain the set of allowed transaction histories. Histories that satisfy the locking constraints are called legal histories. We use the 'PrevLockURI' element of the lock resource to create a linked list of. The client can retrieve the lock collection via GET to determine if the resource is locked. An empty collection indicates an unlocked resource. The role of R-Lc is to allow clients to determine whether a resource can be locked by a new transaction. By retrieving the lock collection a client can determine whether the resource is locked, and what type of lock it has. By the expiration timestamp on the lock resource, the client can estimate when a locked resource will be available again for locking. To emphasise its intention to be solely for retrieval and not POSTing of new locks, we have made the resource read-only for the clients, although this is not strictly necessary. The locks reflected in this resource are the ones that are submitted to T-Lc and reference the lockable resource that the R-Lc is connected to.

Table 10. Available Operations for R-Lc

GET	Returns the resource's collection of locks.

5.3.3 Manipulating Resources within the Transaction Scope

Lock resources link to two other resources: the Initial and Conditional Resource representations, which are created when a lock resource is created. While the initial resource representation is a copy purely for future archiving reasons, the conditional resource representation is the resource that acts as the 'shadow' of the locked resource, with any changes made to it potentially being applied to the original locked resource when the transaction is committed and therefore the lock released.

Initial Resource Representation (L-IR): A resource that is of identical media type as the locked resource and stores its initial state. The initial representation is archived together with the lock to represent the change caused by the commit of the lock and enable rollback.

Table 11. Available Operations for L-IR

GET	Returns the representation that was the initial state of the resource before locking

Conditional Resource Representation (L-CR): A resource that is of identical media type as the locked resource and is created with the same state as well. The conditional resource representation essentially contains the state that will be applied to the resource once the XLOCK is committed. If an L-CR is produced from a SLock, it will only respond to GET and therefore exists purely as a guaranteed point of reference for the length of the transaction.

Table 12. Available Operations for L-CR

GET	Returns the representation that will be committed if the relevant lock is committed.
PUT	Creates a new conditional state that will replace the current state of the locked resource once the linking XLOCK is committed.
POST	Reacts as the original resource would have reacted, but its side-effects are contained within the transaction scope.
DELETE	Deletes the conditional state. If the XLOCK is committed, there will be no write action performed.

Conditional Resource Representations produced by an XLock can be manipulated like the resources they are a copy of would be, in an isolated service. This means that PUT and DELETE operations can be applied to the conditional representation freely. The resulting changes will be reflected on the original resource when the transaction is committed. One property of the locking system is that it can lock resources that do not exist yet, simply by referring to them as the ResourceURI of an XLock. This is similar to the creation semantics of PUT. While this may be counterintuitive, it allows the creation of resources directly within a transaction. The only requirement from the server is that it reserves the URI until the end of the transaction. Any updates to the L-CR of the not-yet-existent resource will become the state of the newly created resource at the time of commitment of the transaction. Correspondingly, the L-IR is null. Finally, the conditional representations can be operated on by POST as well, but any side-effects, such as modification, deletion or creation of resources should be limited within the transaction scope. In fact, for any operation on a conditional resource that the server allows to succeed, resources affected should appear within the transaction

scope as conditional resources themselves and the appropriate locks should be added to the transaction. The server can continue to allow its state-space to be forked, to the extent where there are no real-world (and therefore irreversible) side-effects and also no potential conflicts if the forks were all to be merged at the same time. The locking system and its compatibility rules make sure this is the case. Also, the trust that the server places on the client may play a part in the amount of the state-space that a client is allowed to place locks on and for how long.

At this point, the links between conditional representations must be considered. Since conditional representations of resources in a transaction are interrelated and valid only within the context of each other, their hyperlinks must point to each other, for the duration of the transaction, as long as the linked to resource is also part of the transaction. A resource that is not part of the transaction can be linked to normally. This also implies a post-processing step that must be made by the server to replace all links to transactional conditional representations with their final and original URIs when the transaction is committed.

Transaction History Collection (THc): This collection contains all the operations performed on the conditional resource representations, serialized into URI-named resources themselves. For any point in time, it represents the path taken from the set of initial representations to the set of conditional representations and at the time of commit represents all the operations that are to be applied to the main state-space of the service. This collection is read-only for anyone who wants to verify what the path has been so far. Its main purpose however is for archival reasons, as it represents the sequence of operations taken starting from the consistent state before the transaction to the consistent state after it.

5.4 Archiving

Once a transaction is executed, all its related resources are archived for durability purposes. This includes all the initial representations of the resources involved, all the operations applied to them as the transaction history, and the resulting conditional representations as the end state of the affected resources.

5.5 Model Overview

Having defined all the resource types, it is easy to see that an interconnected network arises. Figure 16 displays the interconnections of the resource graph. It can be observed that having a URI for R is enough to locate all other resources in the network. The URI of a given T is only returned as a response to the initial POST operation on Tc performed by the transaction's owner.

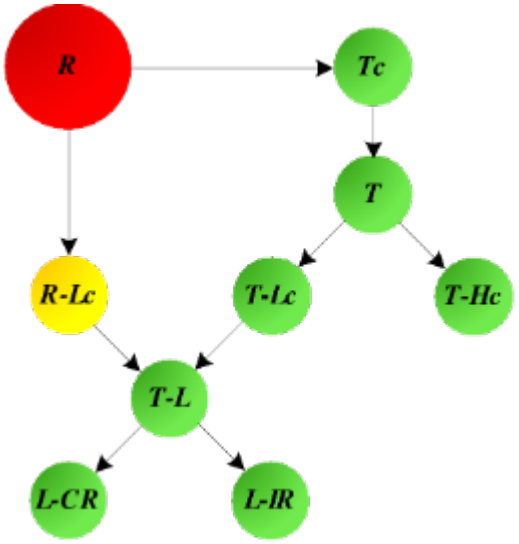

Figure 16. Resource Hypermedia connections

Tables 13 and 14 summarize all the relevant resource types that comprise our model together with a short description and a list of the allowed operations.

Table 13. Transaction model resource types

Lockable Resource (R)	A resource that locks can be applied to
Resource Lock Collection (R-Lc)	The collection of locks that apply to a particular resource
Lock Resource (R-L)	The representation of a specific lock
Conditional Resource Representation (L-CR)	The potential representation of a locked resource, once its lock is committed
Conditional Resource Representation (L-IR)	The initial representation of a locked resource a the time of application of the lock
Transaction History Collection (THc)	The collection that contains all the operations which make up the steps of transaction up to this point
Transaction Collection (Tc)	The collection of transactions on the server
Transaction Resource (T)	The representation of a specific transaction
Transaction Lock Collection (T-Lc)	The collection of locks connected to a specific transaction

Table 14. Allowed Operations on the Resources

Resource	Allowed Operations
R	GET, [By XLOCK owner: PUT]
R-Lc	GET, POST
R-L	GET
L-CR	GET, [By XLOCK owner: PUT, DELETE]
L-IR	GET
Tc	POST
By T	GET
T-Lc	GET, [By transaction owner: DELETE]

5.6 Examples

The section that follows contains examples of transactional interactions between clients and servers. The first two examples focus on a simple service with two resources (R1, R2) while the third example

focuses on a single client executing a transaction over multiple servers. Based on the current model, more advanced concurrency (multiple clients and multiple servers) can be depicted by combining the functionality seen in these examples.

5.6.1 Example 1: Creating a Transaction

The example in Figure 17 focuses on a client attempting to perform a transaction on a server, placing the locks. It also shows the response that another client will get when it attempts to modify a locked resource.

Figure 17. Creating a transaction

Client	Operation	Resource	Response	Description
A	GET	R1	200 OK	GETting R1 to extract location of TC
A	POST <new transaction>	TC	201 CREATED {Location: T1}	Creating a new transaction
A	POST <LOCK {type:X, resource: R1}>	T1-LC	201 CREATED {Location: T1-L1}	POSTing an XLOCK for R1 to T1-LC
A	POST <LOCK {type:S, resource: R2}>	T1-LC	201 CREATED {Location: T1-L2}	POSTing an SLOCK for R2 to T1-LC
B	PUT <new version>	R1	405 Method Not Allowed	Another client attempting to modify R1 and failing
A	GET	R1	200 OK	GETting the locked representation of R1
A	GET	R2	200 OK	GETting the locked representation of R2
A	GET	T1-L2	200 OK	GETting T1-L2 to extract location of L2-CR, the shadow of R2
A	PUT <new version>	T1-L2-CR	201 CREATED	Creating a conditional Representation of

5.6.2 Example 1.1 Aborting the Transaction

Figure 18 shows how the example in figure 17 could be continued if the client decides to abort the transaction.

Figure 18. Aborting a transaction

Client	Operation	Resource	Response	Description
A	DELETE	T1	200 OK	Abort: Deleting T1, L1, L2, L2-CR and Unlocking R1 and R2

5.6.3 Example 1.2 Committing the Transaction

Figure 19 shows how the example in Figure 17 could be continued if the client decides to commit the transaction. On commit, the conditional representation of an active XLOCK becomes the new version of the locked resource. The transaction resource and all other relevant resources are archived for future rollback as they constitute an effective recording of all the changes made by the transaction. Finally, R1 and R2 are unlocked for future use.

Figure 19. Committing a transaction

Client	Operation	Resource	Response	Description
A	POST	T1-Commit	202 Accepted	Commit: Replacing R2 with L2-CR, Archiving T1 and unlocking R1 and R2

5.6.4 Example 2: Concurrent Transactions

The example in Figure 20 deals with multiple transactions executing over the same resources. We can see that SLOCKs can be placed by both transactions on the same resource (R1) but XLOCKs cannot (R2). To XLOCK an already locked resource, the transaction must wait until it is unlocked. Once that happens, the second client can complete transaction T2 normally. Here, the transaction model enables concurrency and protects the two clients from adverse effects that their concurrent interaction would have which might have left them in an inconsistent state.

Figure 20. Concurrent transactions

Client	Operation	Resource	Response	Description
A	GET	R1	200 OK	GETting R1 to extract location of TC
A	POST <new transaction>	TC	201 CREATED {Location: T1}	Creating a new transaction
A	POST <LOCK {type:S, resource:R1}>	T1-LC	201 CREATED {Location: T1-L1}	POSTing an SLOCK for R1 to T1-LC
B	GET	R1	200 OK	GETting R2 to extract location of TC
B	POST <new transaction>	TC	201 CREATED {Location: T2}	Creating a new transaction
B	POST <LOCK {type:S, resource:R1}>	T2-LC	201 CREATED {Location: T2-L1}	POSTing an SLOCK for R1 to T1-LC
A	GET	R2	200 OK	GETting R2 to extract location of TC, verifying lockability
A	POST <LOCK {type:X, resource:R2}>	T1-LC	201 CREATED {Location: T1-L2}	POSTing an XLOCK for R2 to T1-LC
B	GET	R1	200 OK	GETting the locked representation of R1
A	GET	R1	200 OK	GETting the locked representation of R1
A	GET	R2	200 OK	GETting the locked representation of R2
B	GET	R2	200 OK	GETting R2 to extract location of TC, verifying lockability
B	POST <LOCK {type:X, resource:R2}>	T2-LC	403 Forbidden	POSTing an XLOCK for R2 to T2-LC. The operation fails, as the resource is already exclusively locked by T1
A	GET	T1-L2	200 OK	GETting T1-L2 to extract location of T1-L2-CR
A	PUT <new version>	T1-L2-CR	200 OK	Updating the conditional Representation of R2
A	POST	T1-Commit	202 Accepted	Committing T1-L2-CR to R2 and Unlocking R1 and R2
B	POST <LOCK {type:X, resource:R2}>	T2-LC	201 CREATED {Location: T2-L2}	POSTing an XLOCK for R2 to T2-LC
B	GET	R2	200 OK	GETting the locked representation of R2
B	PUT <new version>	L2-CR	200 OK	Updating the Conditional Representation of R2
B	PUT <new version>	L2-CR	200 OK	Updating the conditional Representation of R2
B	POST	T2-Commit	202 Accepted	Committing T2-L2-CR to R2 and Unlocking R1 and R2

5.6.5 Example 3: Multi-service Transaction

The example in Figure 21 shows a transaction over multiple services. In fact, it is an application of the 2 Phase Commit protocol [93]. This is equivalent to the travel arrangements service composition discussed earlier. Multiple services must be coordinated to avoid an incomplete booking. The example shows a highly ordered sequence; however the same result can be attained by interleaving or even parallelising the operations to the servers, as long as the operations to each server retain their relative order and the three commits are performed as the final operations of the transaction.

Figure 21. Multi-service transactions

Client	Server	Operation	Resource	Response	Description
A	B	GET	R1	200 OK	GETting R1 to extract location of TC
A	B	POST <new transaction>	TC	201 CREATED {Location: T1}	Creating a new transaction
A	B	POST <LOCK {type:X, resource: R1}>	T1-LC	201 CREATED {Location: T1-L1}	POSTing an XLOCK for R1 to T1-LC
A	B	GET	R1	200 OK	GETting the locked representation of R1
A	B	PUT <new version>	T1-L2-CR	201 CREATED	Updating the conditional Representation of R2
A	C	GET	R2	200 OK	GETting R1 to extract location of TC
A	C	POST <new transaction>	TC	201 CREATED {Location: T2}	Creating a new transaction
A	C	POST <LOCK {type:X, resource: R2}>	T1-LC	201 CREATED {Location: T2-L1}	POSTing an XLOCK for R1 to T1-LC
A	C	GET	R1	200 OK	GETting the locked representation of R1
A	C	PUT <new version>	T1-L2-CR	201 CREATED	Updating the conditional Representation of R2
A	D	GET	R3	200 OK	GETting R1 to extract location of TC
A	D	POST <new transaction>	TC	201 CREATED {Location: T3}	Creating a new transaction
A	D	POST <LOCK {type:X, resource: R3}>	T3-LC	201 CREATED {Location: T3-L1}	POSTing an XLOCK for R1 to T1-LC
A	D	GET	R3	200 OK	GETting the locked representation of R1
A	D	PUT <new version>	T3-L2-CR	201 CREATED	Updating the conditional Representation of R2
A	D	POST	T3-Commit	202 Accepted	Commit: Replacing R2 with L2-CR and Archiving T1
A	C	POST	T2-Commit	202 Accepted	Commit: Replacing R2 with L2-CR and Archiving T1
A	B	POST	T1-Commit	202 Accepted	Commit: Replacing R2 with L2-CR and Archiving T1

5.7 Verifying the RESTfulness of the model

The RESTfulness of a model depends on the degree to which it satisfies the constraints of the REST architectural style.

5.7.1 Resource Identification

Contrary to most transaction models, our model explicitly identifies locks, transactions, owners and conditional representations as explicit, linkable resources. In fact, every significant entity in our model is represented as a resource in order to comply with this constraint.

5.7.2 Resource manipulation through representations

In order to comply with the manipulation through representation constraint, a model has to interface with the resources solely through their representations. Our model complies with this as updates to resources are performed by applying the uniform interface on the URI of the resource. For locked resources, the representation that will replace the locked resource when the transaction is committed is kept in a special resource named conditional resource representation, which is a resource itself.

5.7.3 Self-descriptive messages

The self-descriptive nature of messages in REST depends on the assumption that both client and server are aware of the same media types and the semantics which they enclose. This refers to both the contents of a resource as well as the results of a protocol operation on one of the links presented by the resource. In this regard, our model introduces two new media types, one for locks and one for transactions. Given that client and server are aware of these media types, the messages can be considered self-descriptive.

5.7.4 Uniform Interface

The model complies by using the operations provided by the HTTP protocol and also by maintaining the properties of each of the operations.

GET: This operation is used only for retrieving representations of resources and is therefore safe. No changes in the state of the server results from the use of this operation. Since it is safe, it is also idempotent as multiple invocations of this operation have no adverse effects on the state of the application.

POST: Although this operation implies no guarantees of safety or idempotency, it is a frequently misused method, as a tunnel for remote procedure calls. Our usage of POST consists of creating new resources when posted to a relevant collection with the specific URI of the new resource created by the server, or invoking a resource to trigger a message to the server such as ‘commit transaction’. By using this operation for these very specific use cases, we avoid its misuse.

PUT: This operation is meant to create or update a resource at a predetermined URI and guarantees idempotency. In our model we have used this operation purely for updating resources. The idempotency property remains as repetitions of the same operation do not cause unforeseen results.

DELETE: Similarly to PUT, this operation implies idempotency. Our model utilises DELETE as a terminating operation (e.g. for transactions) in a manner complies with the idempotency requirements.

5.7.5 Hypermedia as the engine of application state

Special care has been applied in order to avoid breaking this often misunderstood constraint. The model has been designed so that a client aware of only the relevant media types and given a single URI can interact with a server implementing the model. The URI can be of any lockable resource on the server, or of a transaction. Both these resources provide enough information for any relevant resource to be discovered at run-time. This allows for extremely loose coupling as the server is free to alter the naming scheme of its resources at any time, and the clients can be expected to adapt without any deliberate updating process. In fact, throughout this document, no specific URI structure has been referred to or proposed, specifically to avoid the assumption that the transaction is driven by URI structure and therefore out-of-band information.

Having defined all the resources and their interconnections, it is easy to see that a network arises. Figure 16 displays the interconnections of the resource graph. It can be observed that having a URI for R is enough to locate all other resources in the network. The connection from Tc to T is different from the other connections as there is no GET ability for the Tc resource. The URI of T is returned as a response to a POST operation on T performed by the transaction's owner.

5.7.6 Statelessness

In our model, each request by the client includes all the information required to carry it out at the server. Authentication information is expected to be negotiated separately for each request. This produces a system that needs to keep no session information between operations.

5.8 Conclusion

Work on RETRO continues, the next step is to look at long-running transactions. The reason we have focused on ACID as the first step is to develop a basic model we can extend to long-running transactions in the future, but also to show that REST and Transactions can and do mix. Whether or not we have accomplished our second goal remains to be seen.

6 Composing Generative Web Information Systems

As described thus far, Generative Web Information Systems have a number of features that are useful in the context of a distributed system, such as a machine-accessible API and support for transactional interactions. However, we have not yet covered how GWISs use these capabilities to produce complex services by integrating with other web information systems, generative or not. In this chapter we examine the competing visions and propose criteria for a way forward and propose an integrated architecture built on RESTful Web Services described by SBVR as a modelling language, and proceed to show how a number of advanced use cases can be built upon this core foundation.

6.1 Visions for the Future of Web Services

Since its inception, the Web has radically expanded its limits to include ever more people, and cover an increasing part of their everyday activities. While initially based on a very straightforward architecture, the aspirations of its users soon exceeded the limits of what was possible within it. So a number of parties started extending that architecture to enable their own use cases. The architecture was gradually revised to include more and more of the low-hanging fruit, and reached a stable point with the release of HTTP 1.1, URI (RFC 3305) and HTML 4. Recent efforts to revise that architecture tend to be practitioner-driven, intending to enable new uses and standardize widespread practices, without causing fundamental changes to the way the Web works. Examples of this are HTML 5 and the HTTPbis efforts. At the same time, at least three camps have been jostling to define the long-term vision for a Web where machines will be able to participate as users just as humans do on today's Web.

The competing approaches are Representational State Transfer (REST), the Semantic Web and Linked Data Movements, and WS-* web services. Each one descending from a different tradition, using divergent vocabularies, and emphasising different use cases and considerations, with the result being a dialogueue where the sides often fail to communicate constructively.

The WS-* web services efforts [98], also known as Big Web Services [78] are rooted in enterprise systems vendors and the history of that industry. Starting out as vendor-specific Remote Procedure Call (RPC) technologies, many of these consolidated in the 90's leaving CORBA and DCOM as the only major competitors in this space. With the introduction of SOAP, the companies in both camps worked together in building up the WS-* standards. The focus of these efforts was in service description, discovery, composition and transactions, as well as governance and reliability. The work of Jim Waldo [99] already in 1994 however, stressed the problems with RPC-based approaches to distributed computing. In recent years most of the implementation of WS-* technologies has been limited to the interior of the corporate firewalls, having failed to convince practitioners on the open Web. Indicatively, the public UDDI Business Registry that was set up to work as the lynchpin of the WS-* service ecosystem was shut down in 2006 [100]., while Amazon Web Services that have both REST and WS-* APIs saw a 85%-15% traffic split in favour of REST already in 2003 [101]. In 2006, Google abolished their SOAP Search API and retired it in 2009, in favour of the RESTful equivalents [102]. While many of the specifications were arguably compensating for the complexity of the WS-* overall architecture, it must be recognised that the needs of enterprises have not been fully met by competing approaches thus far, leaving the technological need unmet.

The Semantic Web[103] and more recently the Linked Data[104] movement have been focusing on expressing or annotating data on the web with machine-readable semantics. With a strong Artificial Intelligence and Formal Methods background, the Semantic Web community has a focus on ontology and aims for novel inferences to arise from this Web of Data. Like the WS-* effort, they have produced a stack of standards such as RDF, RDFS, OWL, OWL-S which until recently have had relatively limited adoption. The effort by W3C to recast HTML into XHTML which would have allowed for more natural integration with the Semantic Web efforts has recently been abandoned in favour of HTML 5. Similarly, the early versions of RSS, based on the Resource Description Framework (RDF) have been rejected in favour of plain XML-based approaches. Controversial elements are the assumption that all RDF triples in a graph are equally trustworthy, as well as the use of HTTP as a read-only protocol, both very limiting in multipurpose distributed systems like the Web. Recently, Linked Data has achieved some success with owners of large data sets such as government organizations, and there are now vast collections of interlinked triplets, forming what is called the Web of Data. Even so, the technology does not seem to have hit the mainstream, as a 'killer app' that can impact the daily lives of millions has not emerged.

Representational State Transfer has already been presented in chapters 2 and 3, but it is worth reviewing its properties in this context. As a vision for the future of Web Services, its ambition may appear limited at first, compared to the other two visionary approaches. Its focus is on the more earthly goals of scalability, loose coupling, linkability and ease of maintenance. The result of REST on each of these can be seen on todays' Web to great effect. As a result there has been a large recent shift of developer mindshare away from WS-* standards and towards RESTful Web Services. Perhaps the most well known usage of REST as a machine-to-machine interaction paradigm is the increasing use of feeds (RSS[105], ATOM Publishing Protocol[75]) not only for their native blog subscription scenarios, but as a generic mechanism to communicate updates between services. Recent advances include podcasting[106] and real-time feeds[107].

An immediate comparison can be made between the granularities of the three approaches. WS-* web services are the most coarse grained with the fundamental element being a 'service endpoint'. Knowing the historical background helps to explain this, but in the context of the Web, limitations such as not being able to use hyperlinks becomes a big drawback. REST focuses on the concept of a 'resource' as identifying a unit that the service deems important enough to name with a URI. A RESTful web service can and usually does break down to multitude of interconnected resources. Finally, the Linked Data approach is to focus on the RDF triple as its primitive element. While this is arguably more granular than a resource, there has been very little work on identifying individual triples. As a result, HTTP is used to read collections of triples, and SPARQL [108] and the newer SPARQL Update [109] being used as means to make queries and updates on collections of triples at SPARQL endpoints. The SPARQL concept of an 'endpoint' is reminiscent of the 'service endpoint' of the WS-* approach, and can in fact be described with WSDL as a proper WS-* web service. As a result, this brings us back to the service level of granularity in terms of linkable entities.

Table 15. Comparing the Visions for the Future Web

Approach	Community Background	Target Properties	Primitive Object	Use Cases	Key Standards
Semantic Web	AI, Formal Logics	Data accessibility, Semantic annotation	Predicate (RDF Triple)	Data analysis, Automated reasoning	RDF, RDFS, OWL, SPARQL
WS-* Web Services	Enterprise Systems	Governance, Reliability, Security	Service / Procedure	Service Composition, Transactions	SOAP, WSDL, UDDI,
RESTful Web Services	Distributed Systems, Software Architecture, Web Development	Scalability, Evolvability, Loose coupling, Linkability	Resource	Web Sites, Feeds, Search Engines	HTTP, HTML, URI, ATOM

From the point of view of Generative Web Information Systems, the use cases aimed at by the WS-* and Semantic Web approaches are certainly desirable and useful. On the other hand, the RESTful approach is both congruent with the declarative nature of SBVR and backwards compatible with today's web. The challenge then is to approach the benefits promised by the WS-* and Semantic Web approaches, such as composability and inference, without giving up the RESTful aspect of the architecture. As with chapter 5, the approach presented here is useful beyond Generative Web Information Systems, but is especially beneficial when in combination with them.

6.2 Media Types

One area where the Semantic Web and WS-* approaches have put a lot more effort is that of service description. WSDL and OWL-S focus on exactly this task. Contrary to common perception, there is a place for descriptions in REST, and that is the media type. Roy Fielding in his seminal PhD dissertation writes:

> *"The data format of a representation is known as a media type. A representation can be included in a message and processed by the recipient according to the control data of the message and the nature of the media type. Some media types are intended for automated processing, some are intended to be rendered for viewing by a user, and a few are capable of both."* [77]

In the course of a sequence of RESTful interactions, media types should contain all the information that is not being communicated through HTTP. When there is a human driving the interaction, the semantics of the resource can be discerned from the content. When however machines need to act as clients, the semantics of a novel type of resource are not clear. This seems to be the canonical use case for the Semantic Web, and thus we can identify the media type as the appropriate place to communicate the semantics of the resource type. This is supported by modern uses of media types, such as ATOM, which use the media type designation to imply not only syntactic information but also semantic information by linking to the appropriate specification.

A holdover from the time when media types were sparse, media types are registered in the central repository of IANA that requires a time-consuming procedure for the inclusion of a new media type. This introduces a single point of control and potentially, failure. Some defend the IANA barrier to entry as a means on non-proliferation of media types, as they believe this barrier will encourage thought about the act of creating a new media type. Once the low-hanging fruit of what can be done with wide-reaching media types is exhausted though, this barrier becomes one against the non-proliferation of use cases for RESTful systems, and an invitation to use out-of-band information in protocol specification. Additionally, media types are not required to be formally described, and it is not clear what information should be included in a media type description. This means that clients should implement media types on a one by one basis, which leaves RESTful clients constrained to a design-time determined range of understandable media types. It's easy to see how this can limit the run-time potential of a client, whether that is a browser or, more importantly, an automated client supposed to operate with a degree of independence. Search engine spiders are probably the most common example of this category.

The proposed solution is to introduce a single new high-level media type (`application/vnd.sbvr-described+xml`) that enables resources that use it to describe themselves at run time. The `+xml` suffix should be replaceable with `+json` or other equivalents as necessary. This allows clients that implement it to react to unforeseen types of resources, in keeping with the general spirit of the Web. In fact, in the spirit of trying to eradicate out-of-band information, this approach would offer a drastic step forward, as it would offer an in-band way to communicate information that was not known to the creators of the client at design-time. To specify such a media type, we must first examine what the role of a media type is and how these elements can be described in as formal a way as possible.

A relatively non-controversial role that media types play is that of declaring the serialization of the representation. JSON and XML are both popular serializations on the machine-oriented Web. Other media types such as HTML or even PDF and JPG are often found in more human-oriented applications.

What XML, JSON, and HTML define is not only the meaning of characters in a textual representation. They go beyond mere syntax in defining a meta-model around which the information contained in the representation can be organised. If the difference between XML and JSON was merely syntax, the two formats would be convertible to one another by means of trivial transformations, but this is not the case. A core difference of the two is in the expressiveness of the meta-model, and thus in some cases XML is an option where JSON is not suitable. This is due to XML's more expressive meta-model, and in spite of its higher bandwidth overhead.

The ATOM and ATOM Publishing Protocol [75] specifications define three new media types, even though they are all serialised as XML. This indicates that there is a need for extra information beyond the serialization and meta-model information for a RESTful application protocol to be complete. One type of such information is the schema. There is a need to know which elements are allowed to occur and in which arrangement. But beyond that, if the content is going to be consumed and not merely displayed as text to a human, then there is a need to define the meaning of each element and how it is meant to be processed. A vital example of this is the hyperlink element. Without defining which element acts as a link, a hypermedia system such as the web would be decidedly

incomplete. In the case of JSON, additional proposals have been put forward to We have identified four types of information that the media type can convey: These are syntax, meta-model, schema, and semantics.

Figure 1 shows the body of a resource which is of media type *application/vnd.sbvr-described+xml* while Table 2 shows the information that is communicated by the description it links to. While the table displays stylised SBVR-SE statements for presentation reasons, in practice the description would be serialised in the XML Metadata Interchange (XMI) format, as defined in the SBVR standard.

```
<?xml version="1.0" encoding="UTF-8"?>
<link rel=”sbvr-description” href=”http://domain.org/school/student/model.xmi” />
<student>
          <id>3465</id>
          <firstname>John</lastname>
          <lastname>Smith</lastname>
          <is-under-probation value=”false” />

          <link rel=”is-enrolled-in_modules”
              href=”http://domain.org/school/student/3465/is-enrolled-in/modules” />

          <link rel=”is-registered-for_course”
              href=”http://domain.org/school/student/3465/is-registered-for/courses” />

          <link rel=”is_marked_with-grade-for-course”
              href=”http://domain.org/school/student/3465/is-marked-
              with/grade/for/courses” />
</student>
```

Fig. 22. Example XML Serialisation of SBVR-described resource

As we can see the resource links to its description so that a client that is unaware of the metadata that is available can access it through standard hypermedia. The syntax is covered by the specification of XML as a serialisation format. The meta-model portions of the description are partially covered by XML. If one considers SBVR to be the semantic meta-model, then that is specified directly by the media type also. This leaves schema and semantics. Firstly, we can see that the vocabulary gives us the terms that are allowed to occur. Secondly, the fact types define which terms can appear directly below or above which other terms. By defining student as the root node of the graph, we can start to see the formation of a tree. To cover the possibility of cycles, the XML representation contains only one level of information about the student, pieces of information directly related to the student entity. Other, more structured pieces of information can be requested by following the hyperlinks provided, as in the example of the list of courses the student is enrolled in. Finally, constraints such as “It is necessary that each **student** *is registered for* at most five **courses**.” allow the client to have specific expectation with regard to the cardinality of elements that it expects to find, both in the current resource and further along the hypermedia graph. Of course, SBVR has far more expressivity than simple cardinality rules, which allows it to natively express constraints that are difficult or impossible with competing approaches.

Table 16. Instance of an SBVR model subset describing a single resource.

Terms	Fact Types	Rules
Student **Module** **Course** **Grade** **A** or **B** or **C** or **D** or **F** **First name** **Last name**	**Student** *is under probation* **Student** *is registered for* **course** **Student** *is enrolled in* **module** **Student** *has* **first name** **Student** *has* **last name** **Student** *is marked with* **grade** *for* **course** **Module** *is available for* **course**	It is necessary that each **student** *is registered for* at most five **courses**. It is necessary that each **module** that a **student** *is registered for is available for* a **course** that the **student** *is enrolled in*. It is necessary that each **student** that *is under probation is registered for* at most three **courses**.

Rule-based schemas may seem unusual in the context of popular schema languages such as XML Schema, RELAX-NG or Database schema languages such as SQL-DDL. Lee & Chu [110] in their work comparing schema languages conclude:

"In our study, we have found that support for constraints in the schema language (e.g. Schematron, DSD) is a very attractive feature."

Schematron, a declarative, constraint-based language praised for its expressivity. Schematron calls its constraints 'assertions' and each is expressed in two forms, one machine-readable (XPath query), and one in natural language. The two however are only linked by the judgement of the schema developer and have no formal relationship. SBVR serialisations such as Structured English that can produce natural language sentences from a logical formulation may be an improvement. Finally, SBVR models contain definitions for each term. This is a link to semantic information that can be used by the client. Some of these definitions resolve to other terms, some to natural language, and some to outside sources. Unfortunately exposing semantics for novel concepts in an automated fashion is an open problem, but this approach removes all other layers of complexity and exposes this problem directly, making it accessible to various lines of investigation. We have thus far shown how media type information can be communicated with the introduction a new SBVR-enabled media type, and how this media type can cover the possible requirements that a media type may have to cover. This is not to say this media type can cover all possible needs, but a large spectrum of previously hard use cases is indeed made much more accessible.

Once we can have resources exposing models as a description, the question of consuming the exposed models arises. To provide an answer for this question we need to examine our assumptions about the client. If the client is a human, then the schema can be useful in understanding and shaping expectations about the content of the described resource, especially if read in a structured natural language form. This is a use case that derives from SBVR's unique advantages and is therefore hard for other architectures to match without additional effort on the part of the developers. If the client is a an automated agent of some kind, then we must assume that it intends to apply further processing to the data, or expose it to other agents for further use. Currently, simple agents can be written to consume a specific type of resource or a group of resources, with all

addressing and media type information hard-coded into them. Any change in the resource media type or addressing scheme however is liable to disrupt the operation of the client. Agents that implement the Hypermedia constraint of RESTful design can survive changes in the URI-space of a well-designed service; however changes in the media type leave them equally exposed as their naïve counterparts. The work presented here aims to address exactly this problem.

Clients consuming SBVR-described resources would have several more levels of flexibility. Since rules and fact types are built on terms, if the terms are understood by the client, any change in the higher-order constructs can be compensated. This already introduces a large increase in flexibility. If for instance students were suddenly allowed to take on only 4 courses instead of 5 by a change in a relevant rule, the SBVR-enabled client would have no problem adapting, and even checking another database for errors if necessary. A client that understands the meaning of this changed constraint can even act as a proxy between its users and the original resource, returning the rule as an error when it knows that the description will be violated if a request is made. This can introduce large efficiencies for distributed systems by eliminating needless roundtrips. Also, if new terms are introduced that are defined in terms of other existing terms, this too can be comprehended by the client. When new terms are introduced with external or informal definitions, the client can know that it has reached its limits. If the change is additive, it can possibly try to work with the understood subset of the resource, however if a necessary term is removed or changed, operation cannot continue. In these cases where a non-understandable term appears, the system can have recourse to a human, but with a much more graceful explanation of the problem than a simple hard failure, or even silent failure, as is the norm with today's systems. As discussed above, this approach pushes interoperability all the way to the limit of the ontology problem, and even exposes it itself to be addressed without interference from other often entangled problems.

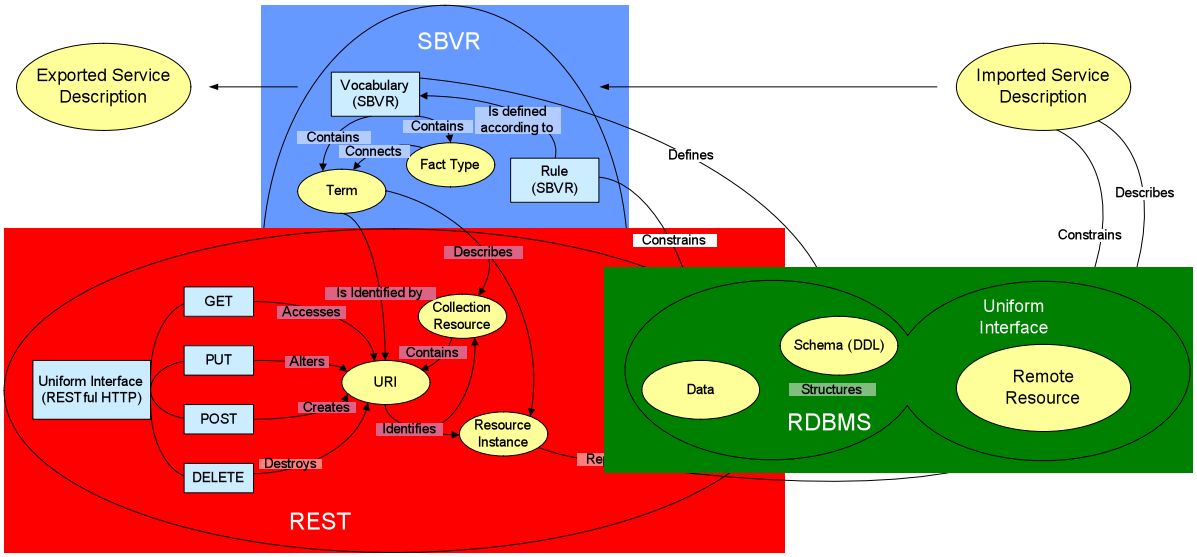

Fig. 23. Extended Generative Web Information System Architecture

6.3 Model Propagation

The previous section discussed ways of exposing and consuming rule-based descriptions of resources. Thus far we have not made assumptions about the internal operation of the nodes, but this architecture has been developed as an interaction paradigm for Generative Web Information Systems (GWIS), which use SBVR as their internal modelling language as well as external resource

description language. It is important to note that while GWIS are the canonical example, in the spirit of backward compatibility, anything that can be done with a GWIS can be done in a conventional information system. The internal architecture of a GWIS simply allows it to do so more efficiently, utilising the full strength of the new paradigm. Figure 2 portrays the architecture of a GWIS, extended to produce and consume SBVR-described resources. What we see in this figure is that service descriptions can be embedded into the internal model of the consuming service itself, and exposed to other users as a resource provided by the consuming service, what we call a remote resource. If a remote resource is presented as writeable, then the consuming service has the responsibility to relay those updates back to the source.

This essentially is a rough description of what happens with mash-ups, where a resource or part of a resource is incorporated into a third-party service. SBVR-description would cure mash-ups of their largest problem, which is that their operation is dependent on stability of internal resource schema and resource addressing on the part of the resource owner. This of course makes some homogeneity assumptions about the services that are mashed up which is not always the case, since many are often out of the reach of the mash-up creator.

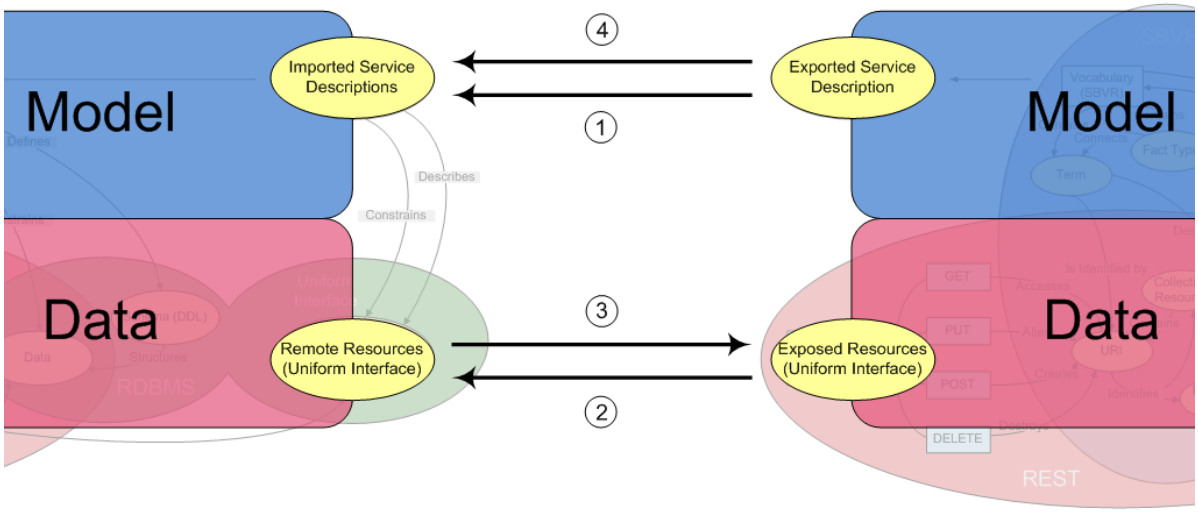

Fig. 24. Model Propagation Workflow

It is important to note that in Figure 24 we can see that the information system is a two-tiered system, where both model and data are dynamic. The model exposes itself to others that can incorporate it into their service models, and it can change when the models it itself depends on change. This is usual behaviour for data, but highly unusual for models which are supposed to be fixed at design time and need long and involved development cycles to be revised. Figure 3 shows how models and data of two linked services interact. First, the resource descriptions are imported into the model of the consuming service (1). Based on those descriptions, the consuming service can read data of the resources that are described (2). If the resources are writable, it can also attempt to make updates (3). Finally, if a discrepancy is noted between the cached model and the behaviour of the resource, the consuming service concludes that an update has been made on the resource structure and can seek to update its own description, by repeating the first step (4).

The exact mechanism by which inconsistencies are noted in the behaviour of the resource is the same as the meta-process presented in chapter 4, but this time in the context of a distributed, machine-to-machine interaction. When the client makes a request for a change to the source, the client expects this to be a legal change to the best of its knowledge. This could be false for data-related reasons (e.g. the student already has 5 courses) or it could be false for model-related reasons, meaning the model has changed such that the description that the client was working with is no longer valid (e.g. the student can now only register for 4 courses). In the second case an update event is triggered, where the description is downloaded again and updated inferences can be made by the client. In terms of the request that is already en route, the source of the request will need to amend it to be consistent with the new model, if it still wants it to go through.

This of course only covers changes to the model that make it more restrictive, but not ones that make it more permissive. For instance, if a sixth course was suddenly allowed, the consuming service would not notice this through this mechanism. What this mechanism guards against is inconsistent updates, and this is achieved. To deal with the case of more permissive updates to descriptions, a number of mechanisms can be adapted such as publish/subscribe or periodic pinging, aided by HTTP's inbuilt facilities for caching such as the `Cache-Control` header and the and optimistic concurrency afforded by ETags and conditional operations. It is important to note that these optimistic approaches can be applied without worrying about inconsistencies arising only because the case of potential inconsistency has already been covered as described above.

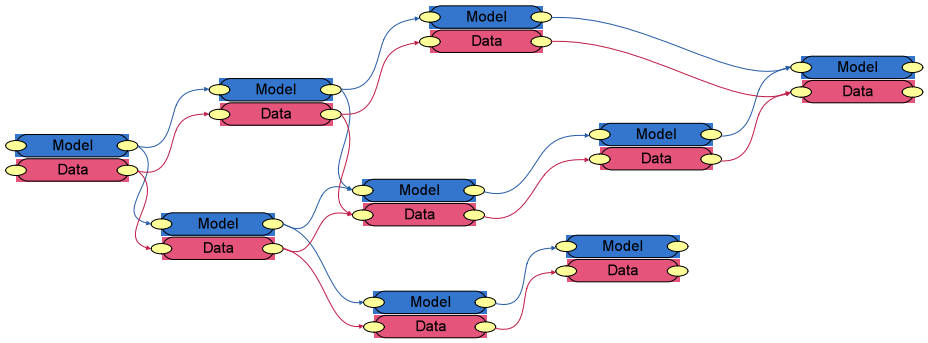

Fig. 25. The Web of Models overlaid on today's Web of Data. As data updates flow from clients to servers, updates in the models propagate in the reverse direction

6.4 Service Composition & Transactions

Having covered resource description, model incorporation and model propagation, we have the building blocks to discuss the higher-level use cases that are common topics in the Semantic Web and WS-* communities. Having incorporated logic-based rules into our architecture it is relatively simple to see how an inference engine would operate over the information in this system. In fact, inference is fundamental to the way the Generative Web Information Systems operate in determining whether an update is allowed and whether it will leave the node in a consistent state.

Chapter 2 has discussed the various types of service composition and they are all considered here. A typical example of service composition is the well-known travel scenario where flight and hotel can be booked independently but must be coordinated for their result to be of use. Once the correct combination of services is found, the execution must be made in a transactional manner to avoid partial success which leaves the user charged for some items without being able to use them if they depend on others that failed.

In our case, we can describe an augmented travel scenario that allows us to explore the potential of model propagation and the meta-process described above. To begin with, we assume various service businesses (airlines, hotels, taxi agencies, sightseeing services) that offer their products as resources in a RESTful ecosystem, and also offer SBVR descriptions for them. A Travel agent service can select quality providers, import their resources and expose them as resources available from its own API. Also, it can create higher-level 'travel package' resources which can act as a container for a number of provider resources (flight, taxi booking, hotel reservation) but also add extra business logic, concerning the consistency between the items in a travel package. So for instance all the items have to be related with the same destination, there is a single starting date for the trip, etc. Further, we can suppose that the travel agency services businesses that make the 'travel package' resource available to their employees. To do that they simply import the travel package resource description from the travel agency. However, each business has its own set of regulations about travel. Some may disallow certain locations, others may place limits on the total budget. These additional constraints can be added to the model such that an employee cannot book a non-compliant travel package through the API of the business. Through this example we can see three levels of business logic nesting which happens naturally. If we consider providers to the base services themselves (such as credit card processing) the nesting can extend to a fourth level or even further.

In order to see how such a scenario would work out in real life, it is necessary to examine the behaviour of the meta-process when operating recursively. Being in a distributed environment, model propagation is also in effect. To interpret the meta-process recursively, the final step, "commit changes", needs to be read as "propagate changes". So, if a number of resources are being written to as part of a travel package, once the business logic of the organization whose employee is making the booking is satisfied, the changes are propagated to the travel agency. It's important to note that the organisation's API does not return success at that point but waits for the response of the travel agency's API. Given that this is a RESTful environment, depending on the semantics of the operation applied, HTTP provides a number of options to deal with the case of no timely response, and these can be taken advantage of by our architecture. Also, HTTP responses such as `202 Accepted` allow the server to acknowledge receipt of the data and defer producing the result.

Once the request reaches the server of the travel agency, the meta-process gets executed again, this time at the travel agency's node, with the organisation's node as a client. The agency's logic is checked for violations, and again the prospect of success or failure is opened. It is important to examine why there is the possibility of failure. One case is that the service has changed its internal logic so the client (the organizational client in this case) is operating on a stale cached copy. In this case, the violated rule is returned as an error, and the client can inform their users. At the same time the client knows that it needs to update the description of the relevant resources it's using so it can prevent the error in future interactions. In every case, the updates can then be propagated further until all relevant nodes in this 'web of models' (Figure 26) are aware of the new rules. The other alternative is that the request has violated a non-public rule. For instance, an insurance company may not want its business logic to be public knowledge and so does not publish the exact criteria by which it may reject an applicant. In that case a generic response is returned and propagates all the way to the end user who cannot do much to change the outcome other than reapply under more favourable circumstances, as would be the case in real life.

The second alternative for a request that is submitted to the travel agency is for the request to succeed. In that case the request will be broken down into parts and each provider will receive the sub-request that is relevant to them. But since the travel agency is interfacing with multiple providers, it needs to make sure that the requests either all succeed or all fail. It is not very useful to have a hotel booking if the flights cannot be booked. This is the transactional property of Atomicity. To guarantee this, we depend on RETRO which was introduced in the previous chapter.

This description of a service composition scenario is a smaller but updated version of previous work done by the author on declarative service composition with SBVR [111]. That work, in the same spirit as the work presented here, also examines the resolution of higher-level queries, which don't specify the exact services to be composed, but rather describe the requirements for the composition. In a sense, this is the next step for the work discussed in this section. The way to go about completing such requests is by using Constraint Logic Programming solvers. Since SBVR rules are essentially constraints, they can be used as requirements for such a component which would produce viable combinations of resources that could satisfy said requirements. Also, a user may choose to execute multiple alternatives in a given order of preference, with the lower-priority combinations serving as a fallback in case of failure of the preferred option.

6.5 Conclusions

In this chapter we have examined concepts and use cases from the three leading schools of thought regarding the future of the Web and constructed an outline for an architecture that can natively compose Generative Web Information Systems. We have used the concept of resource orientation, hypermedia, and constrained distributed interaction from REST, the ideas of service description, transactions, and the use case of service composition from WS-* and the idea of using formal logic semantics and the use case of knowledge integration from the Semantic Web community. All these are brought together into a coherent whole that is backwards compatible with the REST-based architecture of today's web.

A lot of work remains to be done, especially with regards to integrating more complex services, generating user interfaces, and prototyping the ideas discussed in this chapter. Overall, we feel that this unique combination of traits and use cases can genuinely contribute to the discussion about the future of the Web and highlight rule-based approaches in general and SBVR in particular as a powerful unifying paradigm that can expose new avenues for exploration in tackling complex problems.

7 Implementation

7.1 Implementation History

The implementation of a Generative Web Information System is a task that has been ongoing since before the start of this PhD project. The original ideas that matured into the GWIS concept are related to an implementation of a model-driven, web-based Content Management System (CMS) that started in mid-2005. That system, named Canvas, was later expanded to work as a Customer Relationship Management (CRM) tool for a different deployment. The ease of adapting the same engine for a task considered qualitatively different gave rise to the insight that many of the business information system categories are actually based on the same primitives. The original deployment of Canvas was for the website Opadoi.gr and is still used to this day.

Once the first version of the software reached its limits, a complete re-write for the second version began in late 2006. As an increasing number of ambitious features were planned, the effort fell victim to the 'second system effect', ambitious redesigns of working software never quite reaching completion. Despite the failure of the second version, lessons from both efforts were integral to developing the ideas in this thesis.

More recently, collaborations with two MSc students and an undergraduate have led to software which has explored certain aspects of GWIS architecture, especially on the RESTful API/Relational Database axis. Work with MSc student Jiannan Lu in 2009 [87] led to exploring querying relational databases over a RESTful API. The resulting HTTP DataBase Connectivity (HDBC) software was released as open source [116] and published as a poster at the WWW2010 conference [117]. In 2010, we applied these ideas to the real world. Collaboration with the Institute of Animal Health (IAH) at Pirbright and students Giedrius Gliozeris (undergraduate) [88] and Dimitris Mexis (MSc) [87] resulted in software to expose the data of IAH through a RESTful API, use of XSLT transformations to produce human-accessible web pages, and a back end for data management with hypermedia querying capabilities.

All the above projects have led to the maturation of ideas that have been described in the previous chapters, and also illumination of implementation issues for the actual proof of concept for a Generative Web Information System. Empowered by these, the final effort tentatively named 'SBVR with Sails' (SwS) brings together most of the relevant ideas, into a single, browser-based, demonstrable software package. The heart of SwS is the SBVR to SQL compiler, which is described in the next section.

7.2 SBVR to SQL Compiler

While the SBVR standard [48] has been in development for several years, no reference implementation of an SBVR parser has appeared in the open source domain, raising a barrier to entry for experimentation with the language. As such, to implement what was described in the previous chapters, a parser was necessary. With this motivation, we present in this section a compiler that uses as input a minimally augmented plaintext representation of SBVR Structured English (SBVR-SE) models and converts them first to SBVR Logical Formulation (SBVR-LF) and eventually to an SQL model, both SQL-DML (data model) and SQL-DDL (queries).

The architecture of the compiler involves 3 stages: Parsing, Preprocessing and Optimization, and SQL Code generation. The first step converts SBVR-SE to SBVR-LF. The second stage eliminates redundancies and reduces some elements that are hard to represent in SQL to others that are more natural. The third stage converts the processed logical formulation into SQL statements.

The compiler has been implemented in OMeta/JS [5],[6], an innovative variation of Parsing Expression Grammars (PEGs) that extends the pattern matching capabilities so that they can be applied to all stages of compilation, including lexing, parsing, optimisation via the visitor pattern and code generation. Each stage traditionally required different tools which use different languages and implementation patterns, significantly impeding experimentation with new languages.

7.2.1 SBVR-SE to SBVR-LF Parser

While SBVR-SE is not considered a normative part of the SBVR standard, it has become in many ways the most well-known facet of the language and the core of much of its perceived usefulness. So, while it should be noted that SBVR-SE is not the only way to verbalise an SBVR model, it is the de facto standard. As such, we have used it as the input of our compiler implementation. There has been significant discussion over the feasibility of parsing SBVR-SE into logical formulation, so we consider this part of our compiler to be a particularly useful contribution towards progressing that debate. The input of our parser is a plain text version of the model with minimal markup to denote whether each line should be interpreted as a rule, a fact type, or a term. An example model would be written as follows:

```
T: student

T: module

T: study programme

F: student is registered for module

F: student is enrolled in study programme

F: module is available for study programme

R: It is obligatory that each student is registered for at most 5 modules

R: It is obligatory that each student that is registered for a module is
enrolled in a study programme that the module is available for

F: student is under probation

R: It is obligatory that each student that is under probation is registered
for at most 3 modules
```

A line starting with ‘T:’ denotes a term, a line starting with ‘F:’ denotes a fact type, and ‘R:’ denotes a rule. Aside from this minimal markup, we do not require the terms, verbs and keywords to be marked up as is done in the SBVR specification itself. This may theoretically allow certain ambiguities but our grammar resolves those by favouring terms over verbs in case of conflict. In practice, the possibility of a conflict is vanishingly small as the words that can be used both as verbs and nouns

unmodified are few, especially in the formal business vocabulary concerned here. Additionally, for a conflict to arise, the tokens must be in a position where there is ambiguity about their intended role, an extremely unlikely case if at all possible.

The first rule of our model, when coloured as per the specification is this:

It is obligatory that each **student** *is registered for* at most five **modules**

According to the specification, an equivalent logical formulation is the following:

```
The rule is a proposition meant by an obligation formulation.
. That obligation formulation embeds a universal quantification.
. . The universal quantification introduces a first variable.
. . . The first variable ranges over the concept 'student'.
. . The universal quantification scopes over an at-most-n quantification.
. . . The at-most-n quantification has the maximum cardinality 5.
. . . The at-most-n quantification introduces a second variable.
. . . . The second variable ranges over the concept 'module'.
. . . The at-most-n quantification scopes over an atomic formulation.
. . . . The atomic formulation is based on the fact type 'student is registered for module'.
. . . . . The atomic formulation has a role binding.
. . . . . The role binding is of the role 'student' of the fact type.
. . . . . The role binding binds to the first variable.
. . . . . The atomic formulation has a second role binding.
. . . . . The second role binding is of the role 'module' of the fact type.
. . . . . The second role binding binds to the second variable.
```

The output of our parser is a Javascript nested array:

```
[rule,
   [obl,
      [univQ,
         [var,
         [num, 1],
         [term, "student"]],
      [atMostQ,
         [maxCard,
            [num, 5]],
         [var,
            [num, 2],
            [term, "module"]],
         [aFrm,
            [fcTp,
               [term, "student"],
               [verb, "is registered for"],
               [term, "module"]],
            [bind,
               [term, "student"],
               1],
            [bind,
               [term, "module"],
               2]]]]]
```

This output can be seen to be clearly equivalent to the above verbalization, with the first element of each array denoting the type of element in a shortened form. While the output can be mapped to the specification-mandated XMI format in a fairly straightforward manner, since the output is directly processed rather than stored or communicated to other tools, there has not a need to do so at the current stage.

It should be noted that our parser does recognise pluralisation of terms but only trivial forms which are produced by appending an 's' to the singular form. This is not a design limitation of the parser

architecture and a more sophisticated inflection library can be plugged in to cover more complex cases. However there are always exceptions for which a general inflector cannot provide. The ideal solution would be for the users to be allowed to declare the plural form as an attribute of the term in the model, which would override the default provided by the inflector, but this would require an extension of SBVR.

An element of ambiguity in the specification is about the exact form of fact types. While examples of term-verb (unary) and term-verb-term (binary) fact types are provided, there is also discussion of n-ary fact types where n is greater than 2. What is not clear is whether fact types ending with a verb and containing more than one terms (e.g. term-verb-term-verb) are considered valid. For the implementation of the parser we have considered such fact types valid as they increase expressivity without any obvious drawback.

Our parser does not implement the full breadth of the SBVR specification but rather a large and usable subset with a focus on expressing complex rules. The parser can be extended to include less common features of SBVR and indeed this is part of the future work planned. The features implemented are: declarations of terms and fact types, all modalities for rules, all quantifiers, and the keyword 'that' as a means of introducing atomic formulations that constrain variables. With this subset of SBVR, even complex rules such as the following can be parsed into the appropriate logical formulation:

It is obligatory that each **student** that *is registered for* a **module** *is enrolled in* a **study programme** that the **module** *is available for*

7.2.2 Optimizations and Pre-processing

Once the logical formulation is attained, it cannot directly be converted to SQL without some pre-processing, as not all the operators are directly translatable to SQL. Initial theoretical attempts at conversion used first-order logic as an intermediate step. However SBVR's mapping to ISO Common Logic is incomplete, and we found that certain features such as the at-most-n quantifiers would require lengthy conversion which although feasible is impractical for our purposes. This mapping has been published in [3], however the path taken there is not useful for a practical conversion. The crucial insight that allowed bypassing ISO Common Logic was that SBVR-LF itself can be treated as logic, even though this is not discussed in the specification.

Also, SQL cannot directly express the universal quantifier (∀). As a result, any logic structure to be mapped to SQL must have this quantifier eliminated. To accomplish this, we convert every occurrence of ∀□ into the equivalent ¬∃□¬, directly on the SBVR logical formulation.

Similarly, we replace all occurrences of the at-most-n quantifier (∃*0..*□) with the negation of the more easily expressed at-least-m (where m=n+1) quantifier (¬∃□*+1..*).

As a result of these substitutions, as well as the parsing of impossibilities and prohibitions as necessities of negation and obligations of negation respectively, many negations are introduced into the logical formulation. We then apply a double negation elimination rule as an optimization to remove these redundancies.

Finally, when the at-least-n quantifier carries a modality of 1, it is equivalent to the existential quantifier. Since the latter is more straightforward and efficient to express in SQL, we replace the

former with it as another optimization. Once the optimised Logical formulation is attained, it is converted to SQL according to the process described in sections 4.1.1 and 4.1.3 of this thesis.

7.2.3 Further Work for the Compiler

This implementation is the result of long effort at comprehension of the SBVR specification which presents a challenge unto itself. With the progress of this project has come deeper understanding of the structures and rationale of SBVR's design. We hope that the current result as well as that of the ongoing effort will be of use to other users of SBVR, either as an implementation or as a source of ideas on implementing SBVR. Next steps include expanding the coverage of the grammar, especially with the inclusion of logical operators, definitions for terms, more sophisticated inflection, and the support for the concept-type attribute which will allow for complex hierarchies of terms and highly expressive models as a result. The overall aim is to integrate the result into a web-browser based development environment for developing SBVR-based information systems according to the Generative Web Information Systems paradigm.

7.3 Realising the Platform - SBVR with Sails

The compiler may be the heart of the proof of concept but it is not very useful on its own. To become usable, a software package needs to be built around it that implements the remaining aspects of the system: Among other things, the Meta-Process, the transaction model and the user interface, and the RESTful API.

7.3.1 Technology Choices

The foundational technological choice was that of building the entire system to operate inside a web browser, and therefore be primarily implemented in JavaScript. On the client and user interface, this choice is predictable given the amount of web technology that is intricately entwined with this project. However, recent advances made it possible to go further. The Web SQL Database specification, which has been implemented in the Chrome, Opera and Safari browsers, provides a complete, transactional SQL engine inside each browser instance. This database is available to client-side scripts which can execute standard SQL commands against it. In fact, the engine behind the implementation is the well-known SQLite [118] embedded database. As such, the possibility of building a GWIS that operates entirely on the client-side, with no server connection, becomes feasible, and indeed has been the path chosen for this project. This was aided by the fact that OMeta/JS, the parsing language, is itself implemented on top of JavaScript, and compiles its input into JavaScript, making it and its output deployable on any browser engine, without the need for round-trips to a server. Finally, libraries like JQuery, JQueryUI, and JSON.js, have made development in JavaScript and the browser's DOM significantly easier than the reputation it had created for itself in previous years suggests.

Altogether, these technologies make it possible to develop and execute a GWIS that exists entirely within a browser, without even requiring plug-ins or extensions. This approach encourages the users to experiment locally, while also allowing for the end result to be uploaded to a server for access by more than one machine when it has reached a reasonable state of maturity. Along with the ongoing maturation of JavaScript as a general-purpose language, development in the browser has just started gaining a foothold [119], and deployment of complete applications in the browser has not yet been showcased elsewhere to the best of our knowledge, placing this particular paradigm firmly on the cutting edge of the move to browser-based software.

7.3.2 Architecture

While all parts of the software run on the client-side, to maintain the client-server division that will allow for later deployment to a real server, the components are split into two groups, one acting as client and one acting as server. The server aspect is the one that interfaces with the database. The only communication between the two sides is through the serverRequest() method, which takes arguments of an HTTP request and responds similarly. So essentially there exists a pseudo-HTTP Server which allows the user interface to be usable whether the server is local or remote.

To enable the two sides to operate independently, each has its own set of URIs it responds to. Since these URIs can get quite complex, an OMeta grammar has been written for each, which parses the URI into a tree for further processing by the appropriate component. The API of the server has been described in XX, but it is interesting to examine the API of the client, which is deeply interconnected with the decisions made for the user interface.

7.3.3 User Interface & Client API

The initial screen of the user interface (Figure 26) is composed of a modelling text area where the user is meant to type in the model they would like to execute. A number of options are present below the text area:

Load Model: Some example models are available to be loaded to the text area.

Execute Model: This option executes the content of the modelling text area. Once done, the in-browser database contains tables that correspond to the terms and fact types of the model, and the data management area of the user interface is activated. When a model is executed, the modelling area becomes disabled.

Reset: This option returns the system to its original condition, re-enabling the modelling area.

Backup DB / Restore DB: These options are purely to assist with development, and are used to protect the data of an information system while its model is being altered.

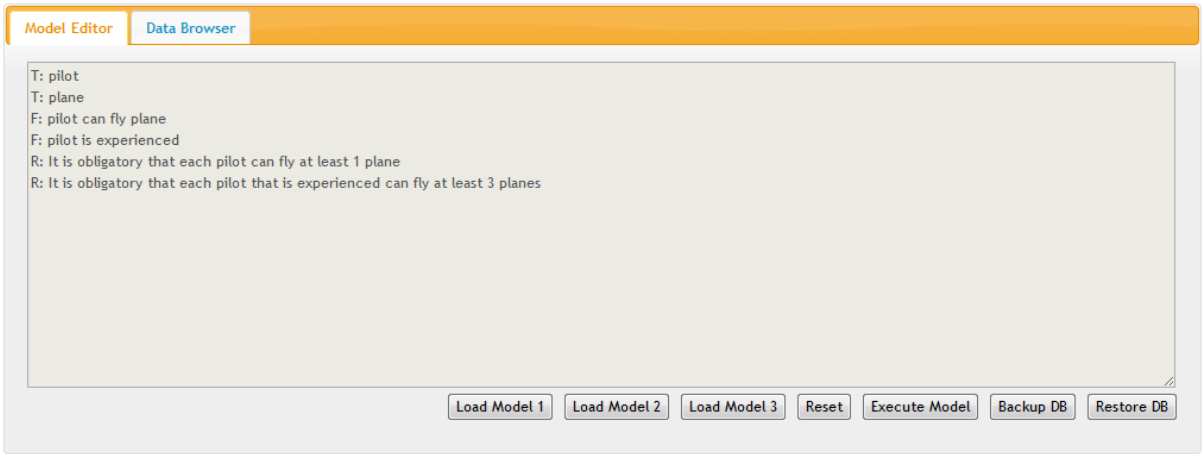

Figure 26. Model Editior

When designing the data-management aspect of the user interface for SBVR with Sails, a number of challenges come up as not only are there novel capabilities to expose, but also the unusual structure of the information system itself renders usual user interface patterns unsuitable. One challenge in

particular is that of arbitrary transactions. Process-driven information systems do not allow users to execute more than one action simultaneously but rather have processes that lead to the execution of a parameterised preset transaction on the back end once complete. Since we wanted to allow users to specify arbitrary transactions, we needed for them to be able to specify multiple actions on the same screen and execute them as a group.

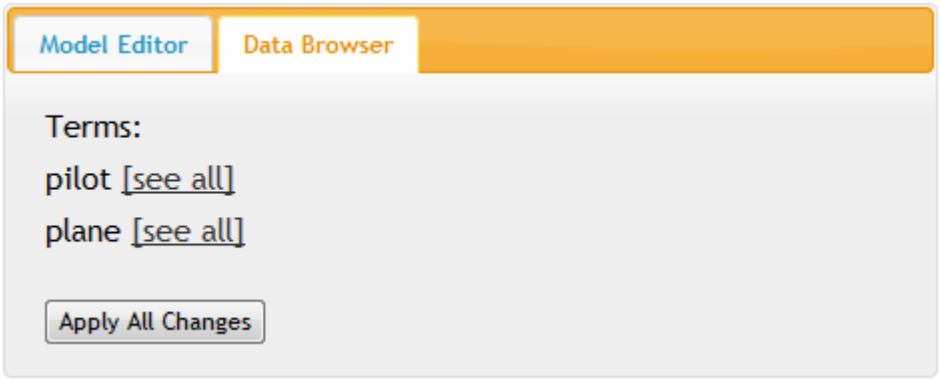

Figure 27. Root view

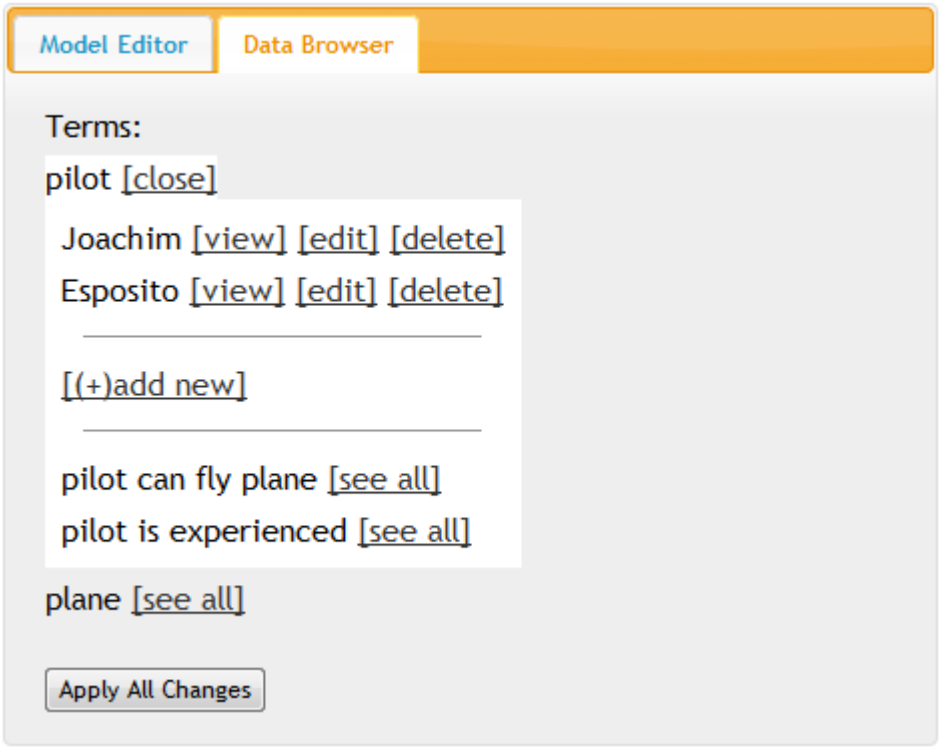

Figure 28. Expanding a term

To achieve this, a tree metaphor is applied to the entire interface. At the root level, the user sees a list of terms (Figure 27). By expanding a single term, all the instances of a term are visible, but also all fact types that involve this term and an 'add instance' option (Figure 28). Each term instance exposes the actions that can be applied to it: viewing, editing, and deleting. Editing allows the user to edit and submit the changed, but also deleting opens a nested dialogue that requires confirmation. Every dialogue that opens gets added inside the dialogue which spawned it. So multiple terms can be expanded at the same time, multiple instances can have actions attached, and

same for the relevant fact types (Figure 29). Besides the individual buttons that confirm one action each, there is also a button that commits all actions in one step, in other words, performing a transaction. The transaction is performed in the manner prescribed by the RETRO model, with a few simplifications since this is only a prototype.

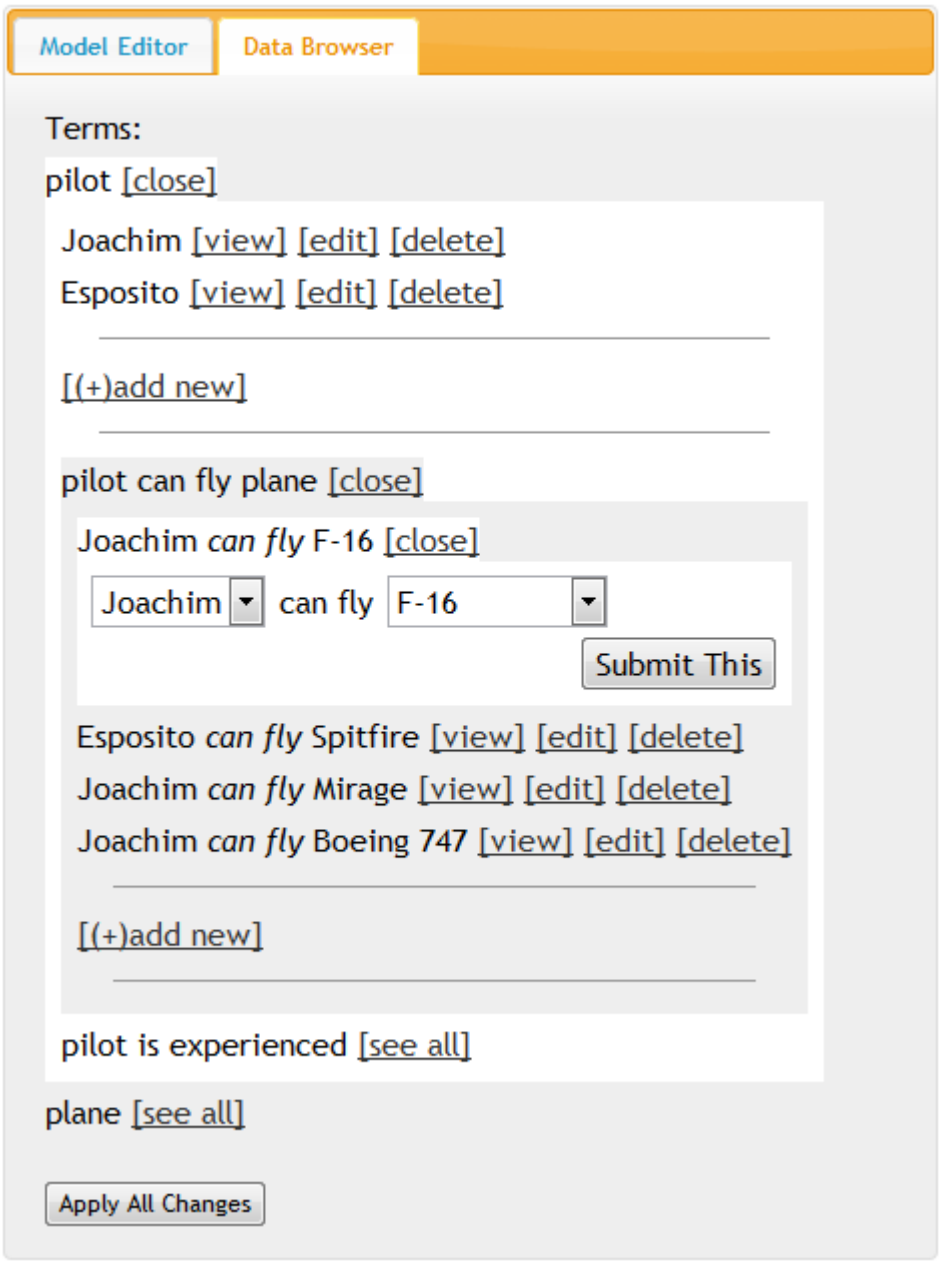

Figure 29. Multiple nested dialogues

Along with the user interface, a client API was developed that also reflects the features of the user interface. The relevant parts of the URI follow the sequence of the fragment identifier and an exclamation mark (#!). So, the URI that leads to the overall view of all data in the system, when the code is hosted on a windows machine is 'file:///C:/{...}/standalone.html#!/data/', where {...} is the path at which the HTML file which contains the application is located. The filename is followed by the characters '#!' and then the location within the application. From this URI it is obvious that the application requires no server-side program and can be launched directly from the file system. It can

of course be launched from a server also. A change in the URI beyond the fragment identifier raises an event within the running code. This means that the script running inside the page can process these events and produce its interface accordingly. In practice, it avoids reloading the page, as the browser interprets it as a change of state rather than a complete move to a different page. Since every state of the user interface is reflected on the address bar of the browser, it can be bookmarked and reopened later, with the exact same UI state appearing on the screen. From here on, all URIs will be printed from the '#!' onwards, which is the relevant segment. So, the URI that has a term expanded is formatted thus:

#!/data/{term}

If the add instance link is clicked, then a nested dialogue appears inside the term dialogue that allows the user to add a new instance. The relevant URI is:

#!/data/{term}/{term}*add/

To view a specific instance, the '.' character followed by the identifier of the instance is appended to the name of the term. So, the URI is formatted as:

#!/data/{term}/{term}.{id}/

This reflects a UI state where the term is expanded and within that dialogue another dialogue is present that displays more information on a specific instance. If the instance is to be edited, the URI is instead:

#!/data/{term}/{term}.{id}*edit/

If the user chooses to delete a certain instance, the URI becomes:

#!/data/{term}/{term}.{id}*del/

This URI does not result in the deletion of the relevant term, but only in the appearance of a UI element that the user can use to verify deletion. This is crucial to note, as using verb names in URIs and having those URIs cause side-effects on the state of the application is an error that developers who are unaware of the semantics of HTTP and REST often make. At first glance, seeing '*del' in a URI raises the flag of potential REST violation, however, as explained, this is not the case.

Since the URIs reflect the user interface, they also follow the format of a tree. Structuring a URI as a tree is a challenge since URIs are meant to be one-dimensional arrays delimited by the '/' character. To add this expressive capability, we use the characters '(',')', and ',', which are allowed by the URI RFC. So if a user wants to delete two instances of a term, the URI would be formatted as follows:

#!/data/{term}/({term}.{identifier1}*del,{term}.{identifier2}*del)

This would present the user with the opportunity to delete items 1 and 2 separately, or, by pressing the 'apply all' button, to delete both in a single transaction. If the user was to expand another term from this state, the new branch of the tree would be added to the URI for a result of:

#!/data/({term1}/({term1}.{identifier1}*del,{term1}.{identifier2}*del),{term2})

Similarly, every expansion of the user interface is reflected in the URI.

The client URIs, as the server URIs, are parsed by an OMeta grammar which produces a tree structure that the code then converts to the appropriate user interface or action. However, the tree structure of client URIs presents a particular challenge. When the links to perform an action are rendered, they must lead to a state where the appropriate node is expanded, but also all other currently expanded nodes are preserved. To achieve this, the current URI is parsed into a tree, the new node is added directly to the tree, and then the tree is processed by another grammar, called the 'Client URI Unparser' which renders the tree back into a URI to be used as the target for the action link. Consequently, when any node is expanded or collapsed, all links must be reprocessed in this way so that the user interaction is consistent.

We have discussed the Client URI and Server URI, and mentioned that the Client URIs can represent multiple nested dialogues. A peculiarity of JavaScript and the Web SQL Database standard is that they favour asynchronous interactions with external components and services, such as a database. Since a client URI may represent multiple queries against the server, the requirement for asynchronous interaction significantly raises the complexity of the implementation as the calls are themselves recursive. This essentially boils down to a pattern of recursive asynchronous invocation which required particular attention to maintaining the correct references. Since there is client-server communication, the reference to the object that is to be called on successful response must be passed to the server. However, multiple requests may have been made to the server at any time, and the context for each should have been maintained. The solution was to create an object for each call, store the context in object fields, and have the callback function as a function of the object. While difficult to design and implement, this pattern improves the concurrency behaviour of the client, as each object is executed in parallel rather than waiting for other calls on the same level to complete before beginning.

7.3.4 Implementing Transactions

A particularly interesting aspect of the implementation is that the transactional behaviour is partially implemented in SBVR. This helps show the capabilities of the paradigm, and also reduce the amount of code needed. Once more aspects of SBVR are implemented in the parser, a larger part, or even all, of the transaction model can be encoded in SBVR. For the moment, each model that gets executed on the server gets augmented with the following model that enables transactions:

```
T: resource
T: transaction
T: lock
F: lock is exclusive
F: lock is shared
F: resource is under lock
F: lock belongs to transaction
R: It is necessary that each resource is under at most 1 lock that
is exclusive
```

This model only partially implements the transaction model, with other aspects such as execution behaviour being dealt with programmatically. It is however important to note that the last rule of the model essentially encodes the locking logic that is described in Chapter 5, Table 8. Once SBVR

Concept Types are implemented for terms, Resource will become the basic type for each term, so that terms will inherit the behaviour and properties of the resource term, such as the ability to be locked. Currently, this link is maintained by additional code. Also, once logical operators for rules are implemented, the exclusive relationship between lock types will be expressible in SBVR with the rule 'It is necessary that each lock either is exclusive or is shared but not both'. At the current level of implementation, this also must be enforced in JavaScript code.
Another observation that is derived from this partial implementation is the excellent modularity that SBVR models offer. One of the observations from developing RETRO is that in traditional contexts it needs to be implemented specifically for each API that it is applied to, as it tightly couples with the relevant resource representations. However, when modelling at this high level of abstraction, this is not necessary. As a result, SBVR models are more modular than typical component-based software engineering patterns such as objects or libraries, at least in this particular instance.

7.3.5 A Simple Scenario

To make the operation of the system more concrete, a simple scenario is presented. Assume the following model has been executed within SwS:

```
T: pilot

T: plane

F: pilot can fly plane

F: pilot is experienced

R: It is obligatory that each pilot can fly at least 1 plane

R: It is obligatory that each pilot that is experienced can fly at
least 3 planes
```

There exist two pilots, Joachim and Esposito, and Joachim is marked as an experienced pilot and can fly 3 planes. Esposito can fly only one type of plane. This is a consistent dataset given the model above. The scenario is that we want to replace one of the planes that Joachim can fly with another one. If we try to delete the one plane before adding another, we end up in the situation shown in Figure 30. The system refuses the update as it would leave it in an inconsistent state. In particular, it would leave pilot Joachim with only two planes he can fly, therefore not qualifying him as the experienced pilot he is marked as.

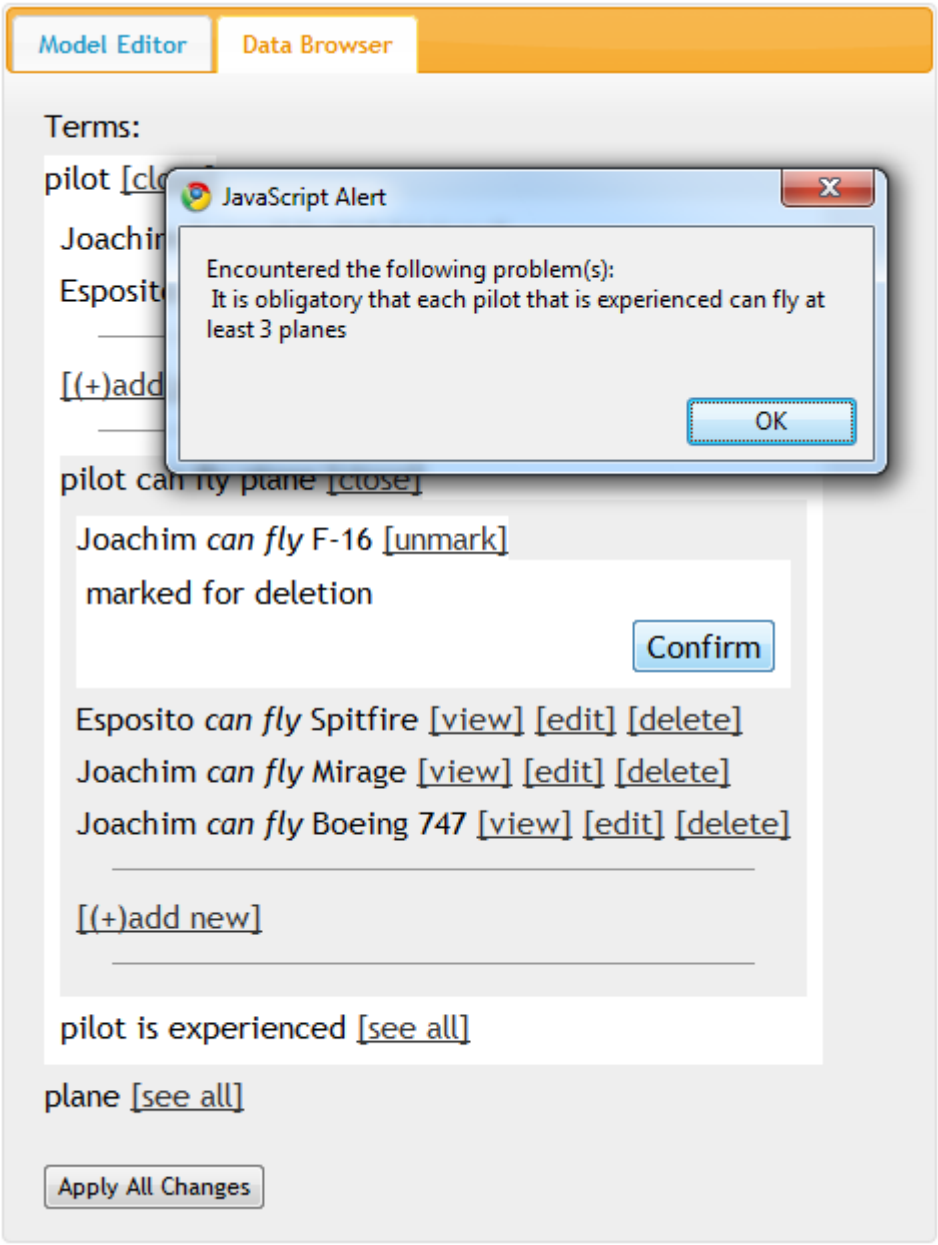

Figure 30. Returning a rule as an error message

The correct course of action is to set both updates in different branches of the user interface, and then execute them simultaneously by using the 'Apply All Changes' button at the bottom (Figure 31). This would instruct the user interface to create a new transaction on the server, create locks for the relevant resources, apply the updates to the conditional representations, and finally execute the transaction which applies the updates atomically on behalf of the user.

7.4 Conclusion

This chapter describes the implementation efforts related to this PhD project. While not exhausting all the ideas described in the previous chapters, they cover a large enough area to provide a confident proof of concept. The implementation process itself has been crucial in further shaping the concept of a GWIS, and next steps include adding concepts such as service composition to the

capabilities of SBVR with Sails. Another critical feature is for applications developed with SwS to be deployable to a server with a single click, such that they can be used by multiple users.

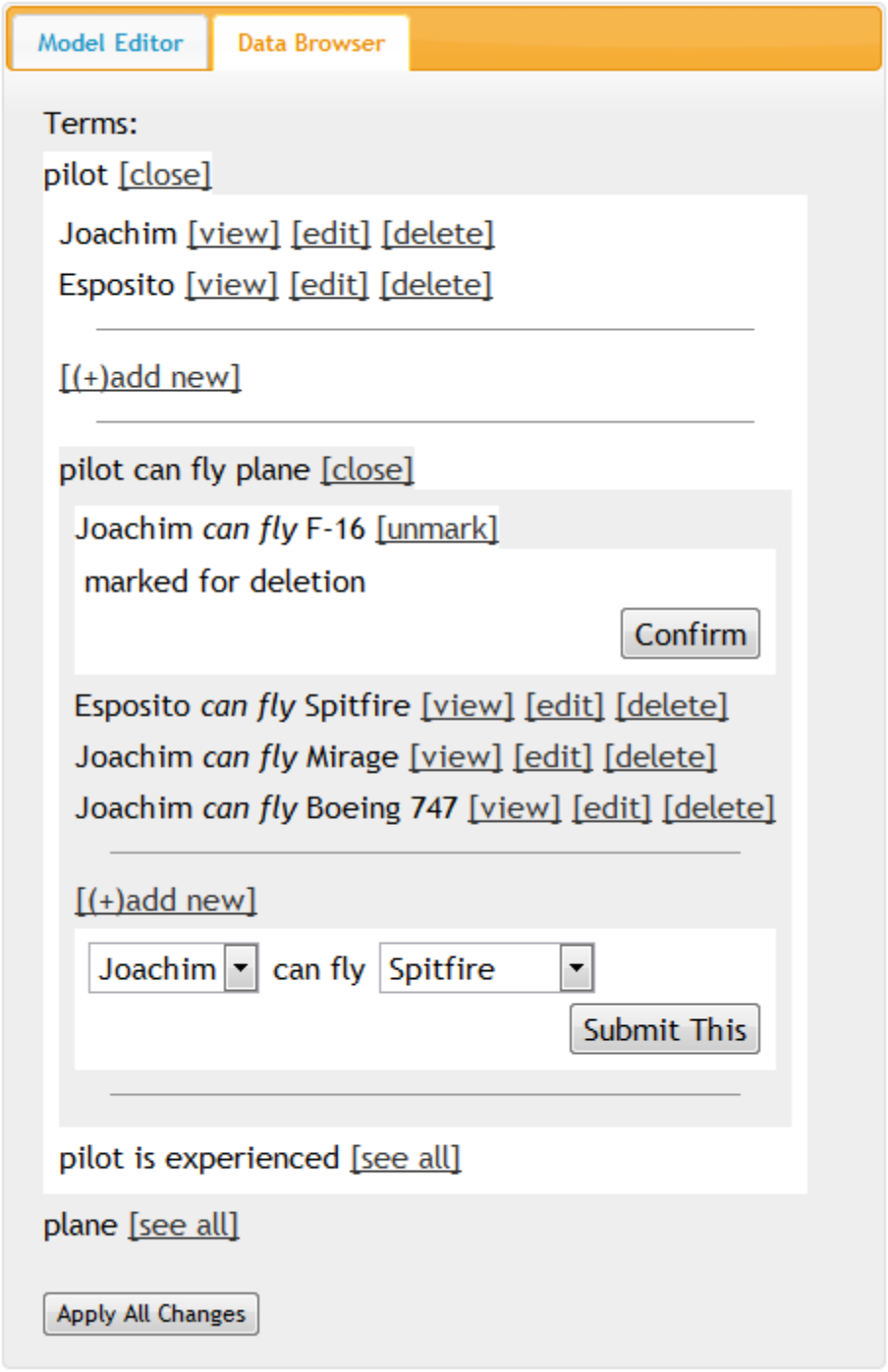

Figure 31. Applying multiple updates

Additionally, by releasing the code as open source, we hope to engage with the multiple related communities. The querying and transaction aspects are topics of interest to the REST community, the SBVR parser and compiler will help further active debates in the SBVR and model-driven engineering communities, while the in-browser development paradigm will hopefully provide an interesting use case for WebSQL, which as a specification is currently at an impasse.

8 Future Work

8.1 Maturing SBVR with Sails

The current status of the SBVR with Sails implementation is a proof of concept. While it covers the functionality needed for an existence proof, it does not have all the peripheral features that would make it ready for production use. The best way to bring it to that point is to actually work towards covering real-world use cases. Besides helping mature the software, such use cases have a way of illuminating further areas of research. Besides, this whole project was indeed an outgrowth of trying to apply modelling concepts to real-world web development projects with a view towards removing repetitive steps from the process. As was noted in the previous chapter, there are currently real-world deployments using earlier versions of the GWIS concept and they will make excellent targets for deploying SBVR with Sails. However, to get to that point, other simpler use cases have to be tackled first, such as managing lists of items, etc. which will require implementing features such as search, querying, ordering, and pagination functionality to the current setup, and of course the ability to deploy an application.

8.1.1 Evolving the Meta-Process

One criticism that can be made of the current interaction pattern is that the user needs to think and click more than what is necessary. While staying true to the principle of *Users as Problem-Solvers, not Process-Followers*, there is still information in the interaction, particularly in the violations, that is currently not being used in the user interface. For instance, if a user tries to mark a pilot as experienced when the pilot is not able to fly at least three planes, they will receive the rule 'It is obligatory that each pilot that is experienced can fly at least 3 planes' as an error message. However, when fully exploiting the feedback to improve user experience, we could produce a nested dialogue that will allow the user to add planes to the pilot that they want to mark as experienced, saving the user of the mental effort to find out exactly what needs to be done. For this to be effective there needs to be more advanced reasoning built into the meta-process than what exists now, but initial analysis shows it is feasible. Also, there needs to be more advanced integration between the meta-process and the user interface layer without compromising the separation between API and UI, something that appears to be a challenge.

8.1.2 Increased Self-implementation

As part of the implementation of the transaction model, we showed how part of it is implemented in SBVR and stated that in the future more of it should be implementable this way. This ambition is not limited to the transaction model, but extends to all parts of the system. Another instance where there exist the foundations of using SBVR with Sails to code itself is the user interface. The client makes requests to the server for the contents of the user interface as if it was querying a model. From there it's not difficult to see how an actual model of the user interface could be created on the back end that will replace these pseudo-calls with real ones. Of course creating a fully declarative user interface is a bigger problem than simply modelling the data that is reflected in it, but as with other areas, incremental progress in this direction, perhaps through combining SBVR with a templating language, will be useful in revealing exactly what the challenges are and whether they are fundamental or can be overcome.

While it would not be defensible to claim that SBVR with Sails is a complete programming environment at this level of maturity, as more and more aspects of it are coded in SBVR, it will either reveal the ultimate levels of what can be expressed, or proceed to the point where SBVR with Sails can claim to be a legitimate programming environment, having itself as its own use case. Both outcomes are quite useful.

8.1.3 Caching

Performance considerations have been beyond the scope of this project thus far. When deploying to real-world use cases however, this luxury is no longer available. Placed under load, the current implementation would probably not scale very far. The core of the problem here is that each request is evaluated individually against the model and database. This real-time interpretation approach is very beneficial for the flexibility characteristics of the result but cannot serve heavy traffic loads on its own. What highly-trafficked websites have found is that caching is a great help in this regard, and this solution is compatible with the GWIS architecture also. In this way results of common queries can be stored in a form that can readily be returned on receipt of a relevant request, bypassing the database. Of course, this applies to read operations and not necessarily write operations. However, for write operations that have been known to fail and no update has been made to the relevant data since, a similar approach can be used.

8.2 Constraint Logic Programming and Service Composition

In chapter 6 we spoke about service composition, but covered only a subset of the exploration that has been done in the area. Initial designs included a much more elaborate method which asked the user to give higher-level rules which were to be used as inputs to a constraint logic programming engine, which would then produce appropriate combinations that satisfy the constraints. Although in fact the earliest work in this PhD project was in that direction, it was later realised that this functionality is in fact a superset of that offered by GWISs, so it was best left as a future extension. This section summarizes that work. Figure 35 is an overview of the approach.

The proposed solution makes a technical distinction in the types of resources that can participate in a service composition. On the one hand there are bounded resources. These resources exist before being requested for by the user and therefore lists of available resources can be made to any query. For example a seat on a specific flight exists before being requested by the user. These resources are called bounded resources. On the other hand, there are resources that are created specifically based on a user's request. These resources are called unbounded resources. It is important to note that these resources cannot be queried through a generic query but have to be requested on a one by one basis. An example of such a resource is a taxi reservation. If one were to ask what taxi reservations could be made during a specific day, the list would be infinitely long due to the granularity of such a service, and therefore unusable.

8.2.1 Requirements Expression

Since our initial focus is on the 'Generating Multiple Effects' type of service composition, we should use SBVR as a means of expressing the desired effects and correlations between them. Within a RESTful architecture this translates to resources and constraints on their attributes. Therefore a two step process is needed. First the user selects the desired resource types from a repository. The interfaces of the services are expressed in SBVR and from them an aggregated SBVR vocabulary for the service composition is generated. Then, based on this vocabulary, rules are written to express

what combinations of these resources are acceptable with both absolute and relative constraints. Absolute constraints refer to properties of the combination itself whereas relative constraints refer to properties of a given combination of resources in comparison to all other available combinations.

8.2.2 Combination Generation

According to the vocabulary and rules that express the request, the combination generator should query the relevant providers and determine the combinations that satisfy the absolute constraints. Querying has been covered in chapters 5 and 7. Also, Microsoft Astoria [72] and Google's GData [70] cover the same ground with slightly different objectives and can serve as a reference for designs in this direction.

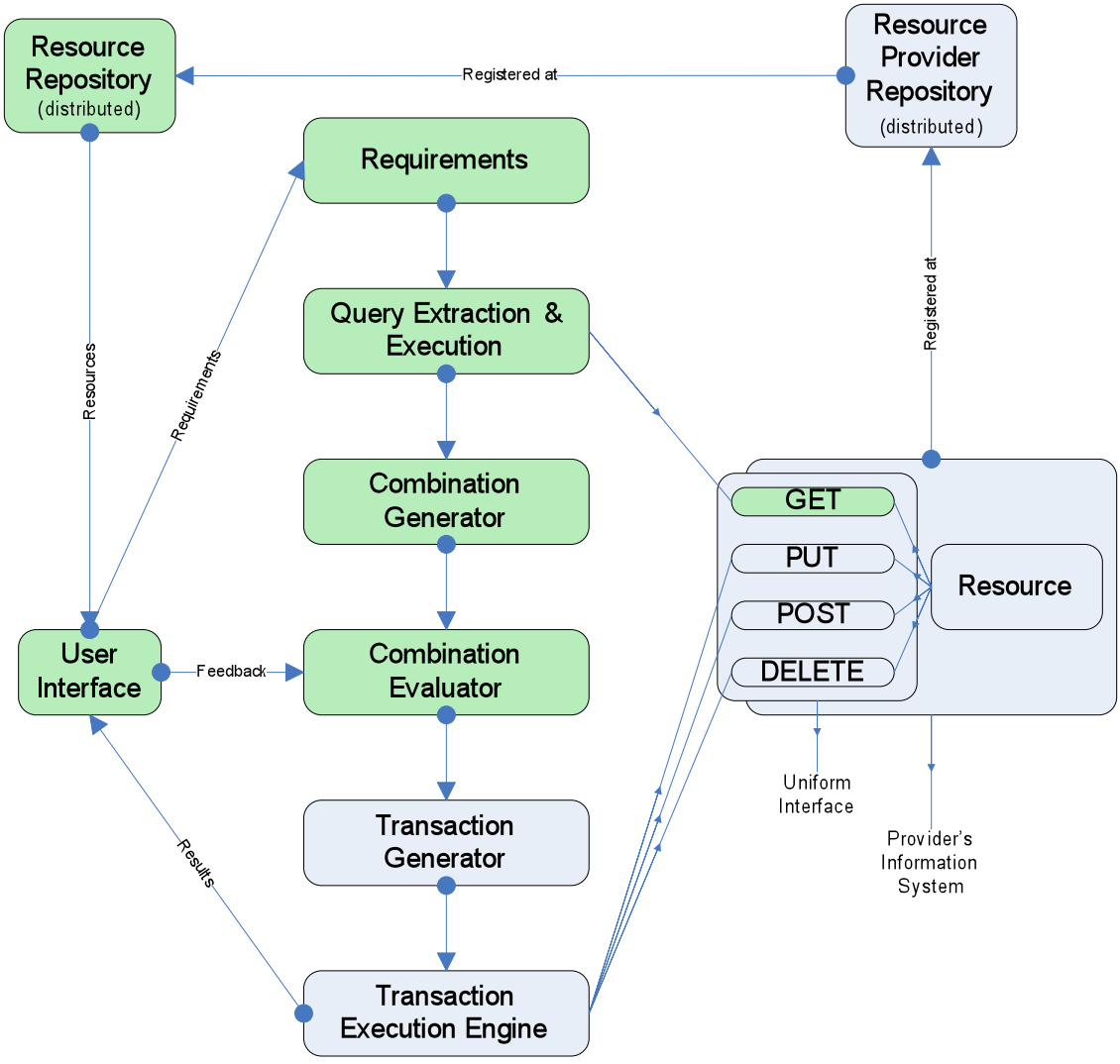

Figure 32. Service Composition Workflow

Determining the appropriate combinations requires identifying a suitable algorithm. Backtracking may fit the requirements; however there are more efficient ways to search the solution space for combinations. A Constraint Logic Programming system such as ECLiPSe[120] can better handle these requirements. It is important to note here that bounded and unbounded resources should be treated at different phases due to their different nature. At first, range queries are made to all bounded resources once and viable combinations of the results generated. Afterwards, the

unbounded resources are queried repeatedly, once for each generated combination of bounded resources. At the end of this process, a list of feasible combinations should be produced.

8.2.3 Combination Evaluation

After the suitable combinations are determined, they can be evaluated either manually or automatically. Automated evaluation depends on relative constraints entered during requirements expression whereas manual evaluation returns the combinations to the user for arbitrary ordering. This is of course preferred because in many cases users will have additional implicit requirements not stated which will cause them to alter the results of the automated evaluation. The two approaches can be combined by using automated evaluation as a step before manual evaluation therefore easing the user's work. However, if a composition is to be exposed as a service itself, manual evaluation is not feasible, since the user will be represented by a machine which will only be aware of the initial requirements and will not be able to provide more accurate insights relevant to processing the returned results. Manual evaluation remains a valuable option where the ability to apply it exists.

8.2.4 Strategies for generating the transaction tree

To generate a transaction tree from a set of combinations there are a number of strategies that must be followed. The strategies differ in the complexity of the transaction tree produced and therefore the amount of intelligence of the transaction model that they utilize. As more complex strategies are explored, it is made clear that a balance must be struck between the decisions that the transaction can take at runtime and the decisions that the user can take at design-time. Deciding this will determine where the barrier between the service composition and the transaction execution phases will be set.

Combination-centric transaction: The easiest way to create a transaction from a set of combinations of resources is simply to consider each combination a sub-transaction, coordinated in parallel and then use a serial alternative coordinator for the entire transaction, ordering the sub-transactions in order of preference of the combinations which they represent. Following this strategy gives us an elegant way to derive a transaction from a set of combinations.

However this simplicity may prove problematic as resources that may appear in multiple combinations may have to be locked and unlocked more than once as the transaction proceeds therefore creating inefficiencies for the provider and risk having a resource made unavailable in the time between unlocking and locking again. Additionally these repeated calls to resources may have performance implications that again may result in unavailability of other resources.

Factoring out overlapping services: In order to deal with this issue, we utilize a different strategy that allows us to factor out overlapping resources into separate sub-transactions. This avoids the possibility of multiple interactions with the same resource. The high level coordinator for such a transaction will be a serial coordinator. Sub-transactions will have similar structure to a combination-centric transaction without overlap. In order to achieve such a structure, we will make use of data driven coordinators to allow sub-transactions to communicate their results to one another. This will allow us to ensure that in any case, the result of a successful transaction is one of the initial combinations, which is the main goal of any transaction tree that is generated.

Deferring decisions: Having established the possibility of pursuing each type of resource as a separate sub-transaction, a new possibility is made available. The decision for the exact resource that will satisfy the requirement for an unbounded resource can be rolled over to the transaction execution phase. It is therefore possible to avoid rejection of a combination simply because an unbounded resource that was available at the time of querying is no longer available at the time of execution of the transaction. This would increase the probability that a transaction will commit successfully. In order to achieve a transaction tree that facilitates this functionality, we would have to extend the structure presented in the "Factoring out overlapping services" strategy by adding the ability for abstract resources to be added as separate sub-transactions. These sub-transactions would be connected to others with the use of data driven coordinators. Once the data has been delivered to the sub-transaction, the provider is queried and a suitable resource is selected.

Implementing this strategy requires modifying the process of service composition to allow creation of less specified results. In this regard it is not directly compatible with the service composition workflow as it stands but is presented as a future direction where the transaction model is empowered with greater decision power.

In order to illustrate the concepts presented above, this section includes a case study. The case study is based on the well-known travel scenario where a user is trying to fulfill a request for travel arrangements.

8.2.5 Service Description

Each resource in the ecosystem is described by an associated set of vocabulary and rules that serves as a service description for each of the providers of the resource. For example the Flight resource could be described with the statements in Figure 19. As seen in the example, a resource could also be constrained by rules.

1. **Flight** *has* **Departure Date/Time**

2. **Flight** *has* **Arrival Date/Time**

3. **Flight** *is from* **Departure Airport**

4. **Flight** *is to* **Arrival Airport**

5. **Airport** *is in* **location**

6. It is obligatory that the **Departure Date/Time** *of* the **Flight** *is before* the **Arrival Date/Time** *of* the **Flight**

Figure 33 - Example of a resource description in SBVR.

8.2.6 Service Selection

As the user selects the resources that are added to the composition, the associated vocabularies and rules are also added to the service requirements so that the rules about the service requirements themselves can be written using them.

a. Vocabulary

1. Travel Arrangement *is* a Service Composition
2. Travel Arrangement *contains* a Flight *called* Departure Flight
3. Travel Arrangement *contains* a Flight *called* Return Flight
4. Travel Arrangement *contains* a Hotel Booking
5. Travel Arrangement *contains* a Taxi Booking
6. Travel Arrangement *has* a date *called* Departure Date
7. Travel Arrangement *has* a date *called* Return Date
8. Travel Arrangement *has* a time period
9. Time period *extends between* the departure date *and* the return date.
10. Travel Arrangement *has* a location *called* home location
11. Travel Arrangement *has* a location *called* destination location
12. Travel Arrangement *has* an amount *called* total cost

b. Rules

13. It is obligatory that the departure date *of* the departure flight *is same as* the departure date *of* the travel arrangement.
14. It is obligatory that the return date *of* the return flight *is same as* the return date of the travel arrangement.
15. It is obligatory that the departure flight *is from* an airport that *is in* the home location.
16. It is obligatory that the departure flight *is to* an airport that *is in* the destination location.
17. It is obligatory that the return flight *is from* an airport that *is in* the destination location.
18. It is obligatory that the return flight *is to* an airport that *is in* the home location.
19. It is obligatory that the pickup location *of* the taxi booking *is at* the destination airport *of* the departure flight.
20. It is obligatory that the drop off location *of* the taxi booking *is at* the location *of* the hotel booking
21. It is obligatory that the home location *of* the travel arrangement *is* London
22. It is obligatory that the destination location *of* the travel arrangement *is* Athens
23. It is obligatory that the pick up time *of* the taxi booking *is* 30 minutes *after* the landing time *of* the departure flight
24. It is obligatory that the departure date *of* the travel arrangement *is between* December 1, 2007 *and* December 3, 2007.
25. It is obligatory that the time period *of* the travel arrangement *is between* 16 and 18 days.
26. It is obligatory that the total cost *of* the travel arrangement *is minimal*.
27. It is obligatory that the total cost *of* the travel arrangement *is less than* £850

Figure 34 - Example of a user request in SBVR.

8.2.7 User Requirements

Except for the vocabularies and rules related to the resources to be composed, a user's requirements also contain rules that link the service composition to each resource and also constrain the resources and service composition according to the need of the users. In our case study, the user's requirements are depicted in Figure 2. In the figure, statements can be seen connecting the resources such as 'Departure flight', 'Return Flight' and 'Taxi booking' to the service composition and

to each other. While statements 1-20 may seem tedious, they do offer a description of the service that can be verified by an untrained user, and can also be reused as a resource themselves. The most interesting statements are the ones from 21 onward that build on the previous ones to express complex constraints on the length and cost of the travel arrangement. It can be thought that a company may have Travel Arrangement service composition templates ready for its employees who then only cave to add the rules relevant to their specific instance of the travel arrangement and execute the service composition as is.

More details about this service composition system can be found in [111].

8.3 Self-Similarity and Fragments

Part of the future work for this project, is not only to implement service composition for bringing together remote systems, but to actually make it a fundamental aspect of Generative Web Information Systems, used to bring together elements of much finer granularity, all the way down to the primitive data types, inspired by the property of self-similarity found in natural systems.

Self similarity, in the context of an information system refers to recursively modelling an information system as being composed of other information systems down to a base case elementary unit. This concept is found in Alan Kay's work, "Steps toward the Reinvention of Programming: a Compact and Practical Model of Personal Computing as a Self-Exploratorium" [49]. In self-similar GWISs, an information system is modelled as a digital ecosystem of interacting nodes. Which themselves can be digital ecosystems. Following self-similarity, each enterprise can model its various departments as separate information systems composed by employee role information systems etc. This level of modularity in combination with declarative modelling unlocks value in enterprises by reconstituting value chains into more dynamic value networks [68]. Self-similarity enables seamless interconnection with resources outside the local infrastructure at multiple levels.

This property will also simplifies system design by using the capabilities usually designed for interaction with external systems for internal purposes. At the same time this extends the capabilities available internally by adding discovery, composition, transactions and collaboration tools to the repertoire of internal interactions. Further, it will also allow mixed transactions which deal with both local and remote resources concurrently.

Finally, this approach will offer a novel solution to the problem of distributed programming. Jim Waldo's work [99] warned against trying to shoehorn distributed computing capabilities into traditional programming paradigms. With the recent popularisation of REST over RPC-based approaches we finally have heeded this advice. However, by modelling an information system as a distributed system at all levels, we resolve the dilemma differently. Rather than two differing paradigms for local and distributed programming, we use the distributed constructs to describe the entire system. In a world of outsourced computing resources, treating every component of the system as being in a different location is not nearly as radical as it may have seemed in the past.

Self similarity is a recursive property. The recursion manifests in information systems of the following layers, starting from the base case:

- Primitive
- Type
- Object
- Department
- Organization
- Local
- National
- Supranational
- Global

This multilayered approach to modelling allows expressing different requirements at the appropriate level of granularity. Each layer can be composed of information systems of a lower order and additional logic which governs their interaction. As the layers get higher, more and more features are made available to the information systems, corresponding to their nature. The levels of information systems/digital ecosystems are delineated by the existence of different rules and regulations (business logic) that apply at each level and which necessitates its existence. For example a supranational body such as the European Union is governed by laws that apply to its members only and which must be modelled at the supranational level, as a composition of national ecosystems and the additional logic as dictated by the EU-wide laws.

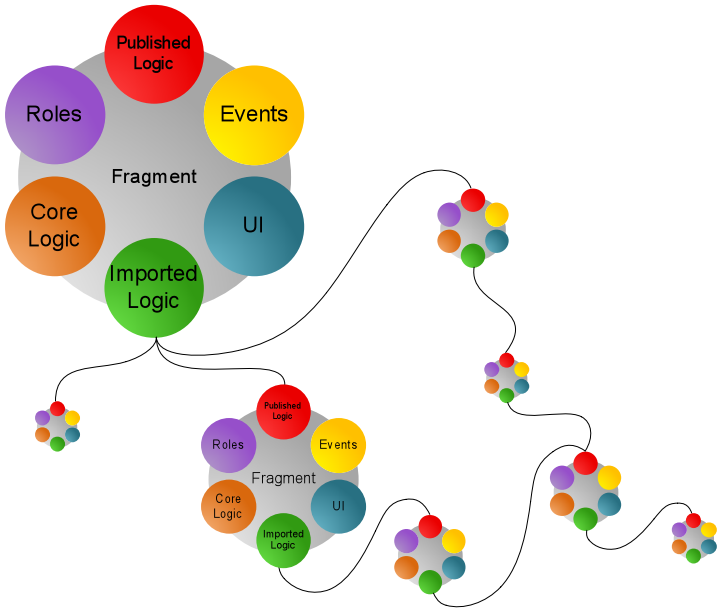

Figure 35. Model Fragments

In order to implement this self-similar architecture, we introduce the concept of a model fragment, essentially a self-contained executable information system model that can compose other fragments. A fragment is a unit of functionality and can be seen as the declarative equivalent of an object. The concept of self-similarity is directly apparent in the fragments. In fact, a fragment can represent an entity in any of the levels of self-similarity discussed in the previous section. A fragment is the sum of the Core Logic, Roles, Events, UI Specifications and Imported Logic contained in it. Each

one may consist of its own SBVR vocabulary and rule set and reference specific concepts found in others. A subset of their contents can become the published logic which can serve as the service description, to be imported into other fragments, continuing the recursive pattern.

In more detail, model fragments may contain:

Imported Logic – The imported logic of a fragment is the sum of the published logic of other fragments that the fragment incorporates. When this logic is referenced, the effects are communicated to the original fragment, whether that is local or remote.

Core Logic – This is the bare-bones business model containing terms, fact types, and rules concerning the objects being modelled by the fragment. The models discussed in this thesis are essentially core logic-only. The core logic can also build on the imported logic and include rules that affect the combination of the imported elements. The core logic is the only element that can introduce new terms and their instances are stored locally to the fragment.

Roles – In order to reveal different parts of the system to different users, we need to define the resources each user has access to. In order to achieve this, the users and/or their roles have to be modelled as well. Roles logic contains all the relevant types of actors that can interact with the data represented by the core and imported logic and the extent of that interaction.

Events – Contains all reactions to events and the actors that own these events. A typical example is if an employee needs to be updated when a specific state occurs or at predefined intervals.

User Interface – Refers to logic that has to do with user interaction with the fragment. This element may in the future be expressible through declarative means, but initially it may be described using a more traditional templating language.

Published Logic – Is the subset of the logic that is published as a description of the service offered by the fragment.

Model fragments will form a recurring pattern at all levels of our architecture. At each different level, their behaviours, features, and relationship to the underlying data change. The element, functional unit, and region fragment types are examined.

An *Element-level fragment* can represent a primitive, a composite type or an object. Their main characteristic is that identical copies can exist in different information systems and also each element can represent a range of instances that its logic governs. A *Functional Unit* fragment represents a much larger entity which may be an entire local information system or a department of a business. It combines element-level and other functional unit fragments into a higher-level construct. Their characteristic is that each fragment usually is unique to its creator and concerns only one instance of its data model. It interacts with the outside world by exposing a subset of its logic which serves as its service description and its services can be used by multiple other information systems. Finally, a Region-level fragment represents an entire region over which a unique set of regulations apply. A region can be a city, a county, a country or a supranational entity. This fragment uses its added logic in order to govern the interactions of the information systems that belong to its jurisdiction. As regions can embed other regions, it is relatively simple to infer that a transaction between two businesses that are based in different regions must adhere to the regulations of all the

relevant regions including the region supersets that are involved in connecting the businesses. For example two businesses from different countries of the EU should be able to cooperate if each one upholds the laws of its country and their interaction is lawful within the framework of the European Union.

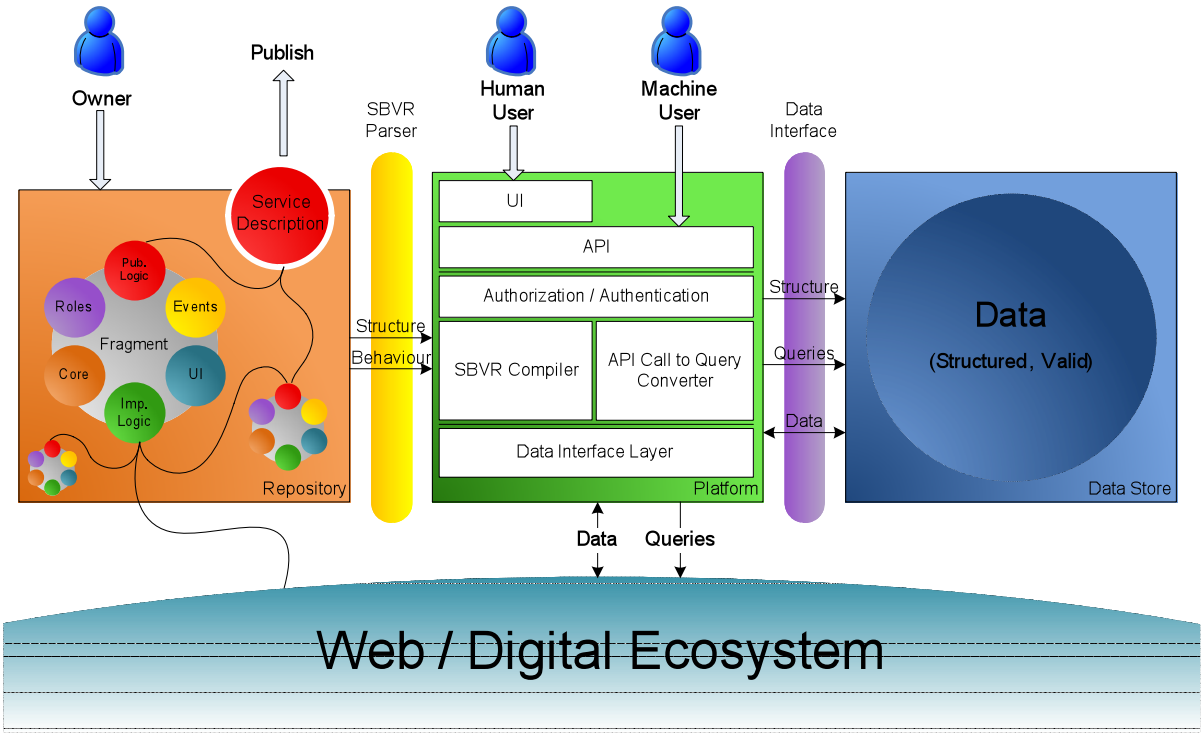

Figure 36 – Conceptual Future Architecture

The most basic fragments discussed are the primitive fragments. These are the base cases in the recursion of the fragments and represent bits of executable code. Since the generative information system is inclusive, executable code accompanied by an adequate description can be embedded at any level. So an information system that has not been built according to the generative principles should be able to participate in the digital ecosystem simply by exposing a compatible service description.

8.3.1 Aligning the Physical and Conceptual Layers

The structure of the logic layer as discussed in this section may seem disconnected from the realities of implemented software systems. This section connects the discussion of fragments with the physical architecture of the computer systems as the role of each actor. It is important to note here that while all fragments necessary for the operation of an information system may be present in a single physical computer, it is also possible that a computer system concurrently hosts more than one fragment groups, and also that a fragment set is distributed across a local or remote set of computers. If we represent each computer as a logic-platform-data set, the matching between physical computers and model fragments can be seen in Figure 36. Essentially each fragment becomes a service that can migrate to the appropriate habitat, bringing its data with it.

Also, if the model fragments are represented as a data element themselves, they can be physically stored in the database and also, by being described through other, give the architecture reflexivity by being able to provide an interface to editing its own configuration.

8.4 Community Cloud Computing

As has been hinted throughout this thesis, this project has been developed in the context of Digital Ecosystems research in general, and the OPAALS project in particular. Besides getting input that has helped shape various aspects of this work, there has also been outward influence towards the digital ecosystem conceptualisation.

These influences have been consolidated in Community Cloud Computing [121, 122], a hybridization of the digital ecosystems sensibilities about distribution and avoiding single points of control and failure, with the infrastructure-agnostic aspect of the cloud computing paradigm. The community cloud represents a version of cloud computing that is not dependent on data centres, but rather draws its resources from a peer-to-peer network of interconnected users' machines. Because the computing environment in which an application will be executed cannot be predicted in advance, the dynamic nature of GWIS is especially suited. By specifying behaviours at a declarative level, execution engines can make the appropriate trade-offs at run-time to satisfy the users' requirements without being unduly restricted.

Community cloud computing is of course at an early stage and exists purely as a concept. While there is research momentum gathering around the concept, it should be considered as a medium- to long-term aspect of future work. If however taken to its full potential, it represents the most promising avenue for the popularisation of GWIS as a methodology for specifying and implementing information systems.

9 Conclusion

This thesis has endeavoured to bring forward a new paradigm for developing information systems. To do so, we discussed definitions of terms and relevant work, examined the stakeholders involved and their requirements which were used to produce the architecture for Generative Web Information Systems. A RESTful transaction model was then presented which added the capability to perform multiple operations with the guarantee of atomicity. This enabled a service composition strategy which allows multiple Generative Web Information Systems to be composed into more complex services. The implementation work on materialising this architecture was described along with details on the next steps in its evolution.

The goal of this project as set out in chapter 1 is

"[...] to realise a new type of information system, more dynamic and less opaque to its owners, specified with structured natural language models and queried through hypermedia. To accomplish this, we focus on Semantics of Business Vocabulary and Rules (SBVR) as a modelling language, Representational State Transfer (REST) as an interface paradigm and Relational Databases as the persistence mechanism. All three of these technologies have declarative underpinnings, focusing on the 'what' rather than the 'how', which is why their combination is feasible and effective. By creating appropriate mappings to align these technologies, and completing gaps where necessary with new technologies, we create the core of a purely declarative information system. The resulting system can gracefully handle unforeseen requests by its users, exhibiting process-like behaviour without requiring processes to be explicitly defined."

Of this goal, a number of questions arise that can be used to evaluate the results produced.

Q. Does the result use strictly declarative interpretations of SBVR, REST and Relational Databases?

The paradigms are used intact, with no additional expressive capabilities being necessary. Where extensions have been proposed, such as the 'desirability' modality for SBVR, these are optional and do not infringe on the declarative nature of the underlying technology. In fact, compared to other attempts to execute SBVR-based models, SBVR with Sails is the only one thus far that executes the language without adding imperative constructs, therefore proving that this is both feasible and useful.

Q: Have all the mappings necessary to align these technologies been created?

The SBVR model is transformed to a schema for the relational database and a URI-space for the RESTful interface, covering two of the axes in the architectural diagram. The reverse mappings are not necessary as the SBVR model is dependent on the owner of the information system and not on the URI-space or relational schema. Additionally, the requests received by the RESTful API are converted to SQL queries, with the results serialised to JSON to be served as a response to the client. As such, the REST to Relational Database mapping is in fact two-way, mediated by the SBVR model. It should be noted that the SBVR language is not fully covered by these mappings, but only to the extent needed for this proof of concept. Future extensions to the mappings will provide new functionality.

Q: Has the development of additional technologies been necessary?

The mapping of paradigms produces a useful core but does not cover all the relevant requirements. The implementation of the meta-process required a RESTful transaction model, which was not readily available. For this reason, RETRO was developed from scratch. Also, since there existed no usable parser for SBVR Structured English, one had to be created. Another necessary innovation has been the use of SBVR as a schema language and the introduction of the model propagation workflow to support digital ecosystems that can sustain movement not only of data but also of models. Finally, the SBVR with Sails platform brings all the mappings and additional technologies together and adds a novel user interaction paradigm that follows from the architecture of the Generative Web Information System.

Q: Has procedural modelling of use-case/processes been avoided?

Throughout all steps of development the introduction of procedural constructs and the modelling of processes have been carefully avoided. In fact, the most major contribution of this PhD project is to examine the limits of what is possible without explicit modelling of processes. The outcome has been that an information system can indeed be modelled, and that there is significant headroom still for increasing ease of use without compromising the declarative nature of the system.

As indicated by the above answers, the implementation of SBVR with Sails provides a proof by construction for the research statement. The peripheral contributions of this work include the RESTful transaction model which was the first of its kind, the SBVR parser and the compiler to SQL, the concept of the Meta-Process and the model propagation workflow. Many of these contributions are useful outside the context of a Generative Web Information System. However the effort to realise a Generative Web Information System is what raised the right questions. With these questions framed in the appropriate context, the answers themselves were produced with significantly less friction, and when put together produce a coherent whole.

While SBVR, REST and relational databases were shown to share significant common ground, the communities that practice each are quite disparate. Many practitioners of SBVR have not heard of REST and nearly every REST practitioner is unaware of SBVR. While both communities are aware of relational databases, they are perhaps not aware of the connection that can be weaved with them. The true metric of success for this work will be the engagement of the various communities with the concepts and the code produced, and the degree to which they will be exposed to each other's existence and experience.

References

[1] J. Johnson, “CHAOS: the dollar drain of IT project failures, Application Development Trends”, January 1995, pp. 41–47
[2] J. Johnson, “CHAOS 2006 Research Project,” CHAOS Activity News, vol. 2, 2007
[3] F. Nachira et al., “Digital Business Ecosystems”, Office for Official Publications of the European Communities, 2007
[4] Zittrain J., “The Future of the Internet--And How to Stop It”, Yale University Press, 2008.
[5] Dictionary of Business Terms. Copyright © 2000 by Barron's Educational Series, Inc.
[6] University of New South Wales, “Policy: UNSW Electronic Recordkeeping Policy.”, viewed on April 30, 2009, http://www.policy.unsw.edu.au/policy/electronic_recordkeeping.htm,
[7] UN/UNECE, Terminology on Statistical Metadata, United Nations, Geneva, Switzerland, 2000, www.unece.org/stats/publications /53metadataterminology.pdf.
[8] C.J. Date, “What Not How: The Business Rules Approach to Application Development,” Addison-Wesley Professional, 2000.
[9] D.D. Chamberlin et al., “A history and evaluation of System R,” Communications of the ACM, vol. 24, 1981, pp. 632-646.
[10] Information and Communications Technology Strategic Plan, 2005-06 to 2009-10, http://www.ict.ox.ac.uk/strategy/plan/plan.xml.ID=appF)
[11] Bussler, C. (2006). Semantic web and web services. Information Systems, 31, 229–231.
[12] Sassen, A. M., & Macmillan, C. (2005). The service engineering area: An overview of its current state and a vision of its future. Retrieved April 7th 2007 from ftp://ftp.cordis.europa.eu/pub/ist/docs/directorate_d/st-ds/service-area_en.pdf.
[13] IBM, “Services Sciences, Management and Engineering: Services definition”, viewed on April 30, 2009, http://www.research.ibm.com/ssme/services.shtml
[14] J. O'Sullivan, D. Edmond, and A. Hofstede, “What's in a Service?,” Distributed and Parallel Databases, vol. 12, 2002, pp. 117-133.
[15] U. Kuster, M. Stern, and B. Konig-Ries, "A Classification of Issues and Approaches in Automatic Service Composition," Intl. Workshop WESC, vol. 5, 2005.
[16] Directorate General Information Society and Media of the European Commission, “Technologies for Digital Ecosystems - Innovation Ecosystems Initiative”; http://www.digital-ecosystems.org/.
[17] Digital Business Ecosystems, Accessed on 30 April 2009, http://www.digital-ecosystem.org
[18] Open Philosophies for Associative Autopoietic digitaL ecosystemS, Accessed on 30 April 2009, http://www.digital-ecosystem.org
[19] DBE Project, “Deliverable 15.1: Business Modelling Language”, Available at: http://www.digital-ecosystem.org/Members/aenglishx/linkstofiles/deliverables/Del_15.1_DBE_Business%20Modelling%20Language%201.0.pdf/download
[20] DBE Project, “Deliverable 15.2: BML Editor, Second Release”, Available at: http://www.digital-ecosystem.org/Members/aenglishx/linkstofiles/deliverables2/Del_15.2_DBE_BML_Editor_2nd_Release.pdf/download
[21] Isakowitz, T., Bieber, M., Vitali, F., “Web information systems”, Communications of the ACM 41 (7), 1998, pp. 78–80
[22] Zachman, J., "A Framework for Information Systems Architecture", IBM Systems Journal, Vol. 26, No. 3, pp 276-292, 1987.
[23] The Business Rules Group, 2000, “Defining Business Rules ~ What Are They Really?” 4th ed. (July 2000), viewed on July 21, 2006 http://www.BusinessRulesGroup.org/first_paper/br01c0.htm
[24] The Business Rules Group, 2005, The Business Motivation Model: Business Governance in a Volatile World (September 2005), viewed on July 21, 2006 http://www.BusinessRulesGroup.org/second_paper/BRG-BMM.pdf
[25] Zachman Institute for Framework Architecture, “The Zachman Framework”, viewed on April 30, 2009, http://www.zifa.com/framework.pdf
[26] Schmidt, D. C., 2006, ‘Model-Driven Engineering’, IEEE Computer, 39, (2), pp.25-31.
[27] Tilkov, S., 2003, “MDA from a Developer’s Perspective”, Random Stuff Stefan Tilkov’s Weblog, viewed on July 21, 2006, http://www.innoq.com/blog/st/2003/07/mda_from_a_developers_perspective.html
[28] Lindsay A., Downs D., Lunn K., 2003, "Business processes – attempts to find a definition", Information and Software Technology, (45), pp. 1015-1019.
[29] Davenport, Thomas (1993), Process Innovation: Reengineering work through information technology, Harvard Business School Press, Boston
[30] Hammer, Michael and Champy, James (1993), Reengineering the Corporation: A Manifesto for Business Revolution, Harper Business
[31] John Wiley & Sons Inc., “Glossary – Chapter 11: Artificial Intelligence”, Viewed on April 30, 2009, http://www.wiley.com/college/busin/icmis/oakman/outline/chap11/misc/glossary.htm
[32] W3 Consortium, “OMG Production Rule Representation - Context and Current Status”, Viewed on April 30, 2009, http://www.w3.org/2004/12/rules-ws/paper/53/
[33] Red Hat Inc., "JBoss.com - JBoss Rules", Viewed on April 30, http://www.jboss.com/products/rules

[34] C.L. Forgy, “Rete: a fast algorithm for the many pattern/many object pattern match problem,” IEEE Computer Society Reprint Collection, 1991, pp. 324-341.
[35] Halpin, T., “ORM2 Perspectives, Objectification, Modality, and Tooling”, Neumont University, 2006, viewed on April 30, 2009, http://www.starlab.vub.ac.be/events/TerryHalpinVUB2006.ppt
[36] Osterwalder, A., Pigneur, Y., Tucci, C., 2005, “Clarifying Business Models: Origins, Present and Future of the Concept”, CAIS, Vol. 15, 2005, pp. 751-775
[37] DBE Project, “Deliverable 15.3: DBE BML Framework, Second Release”, Available at: http://www.digital-ecosystem.org/Members/aenglishx/linkstofiles/deliverables/servicefactory/Del_15.3_DBE_BML_Framework_2nd_Release.pdf/download
[38] http://www.omg.org/mda/mda_files/09-03-WP_Mapping_MDA_to_Zachman_Framework1.pdf
[39] http://omg.org/docs/br/03-09-03.pdf
[40] Jan Vanthienen and Stijn Goedertier, "How Business Rules Define Business Processes," Business Rules Journal, Vol. 8, No. 3 (March 2007), Available at: http://www.BRCommunity.com/a2007/b336.html
[41] Goedertier, S., Haesen, R., Vanthienen, J., "EM-BrA2CE v0.1: A Vocabulary and Execution Model for Declarative Business Process Modelling" (2007). Available at SSRN: http://ssrn.com/abstract=1086027
[42] S. Goedertier and J. Vanthienen, “Declarative Process Modelling with Business Vocabulary and Business Rules,” LECTURE NOTES IN COMPUTER SCIENCE, vol. 4805, 2007, p. 603.
[43] S. Goedertier and J. Vanthienen, “Designing Compliant Business Processes with Obligations and Permissions,” LECTURE NOTES IN COMPUTER SCIENCE, vol. 4103, 2006, p. 5.
[44] S. Goedertier and J. Vanthienen, “Rule-based business process modelling and execution,” Proceedings of the IEEE EDOC Workshop on Vocabularies Ontologies and Rules for The Enterprise (VORTE 2005). CTIT Workshop Proceeding Series (ISSN 0929-0672), Enschede, 2005.
[45] S. Goedertier, “Using Logic for automating business decisions: a critical reflection from a Philosophy of Science”, Katholieke Universiteit Leuven, FETEW, 2005
[46] C.J. Date, A guide to the SQL standard, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1986.
[47] R. G. Ross, “The Business Rules Manifesto,” Business Rules Group. Version 2 (2003).
[48] Object Management Group, “Semantics of Business Vocabulary and Rules Interim Specification,” 2006, Online at: www.omg.org/cgi-bin/doc?dtc/06-03-02. Accessed: 25/10/2007
[49] A. Kay et al., Steps Toward The Reinvention of Programming a Compact and Practical Model of Personal Computing as a Self-Exploratorium, VPRI Research Note RN-2006-002. Available at: http://www.viewpointsresearch. org/pdf/NSFproposal.pdf, 2006.
[50] R. Pawson, “Naked Objects”, PhD Thesis, Department of Computer Science, Trinity College, Dublin, http://www.nakedobjects.org/downloads/thesis.pdf.
[51] “Webyog.com”, viewed on April 30, 2009, http://webyog.com/en/.
[52] A.B. MySQL, MySQL 5.0 Reference Manual, Bestandteil des Programnmpaketes, 2006.
[53] P.S.C. Ltd, “Navicat Oracle, MySQL Admin Tool”, Accessed on April 30 2009, http://www.navicat.com
[54] Delisle, M., “phpMyAdmin”, Paris, CampusPress, 2005.
[55] DabbleDB, “Create an Online Database - Collect, report, and share your data”, accessed on April 30, 2009, http://dabbledb.com/.
[56] Dabble DB Blog, “White- (or green, or blue, or yellow) label Dabble”, Accessed on April 30, 2009, http://blog.dabbledb.com/2007/04/white--or-green.html.
[57] Blist Inc., “share your weblists”, Accessed on April 30, 2009, http://www.blist.com/.
[58] Iceberg, “Build Workflow Powered Applications Without Coding”, Accessed on April 30, 2009, http://www.geticeberg.com/.
[59] Zoho Inc, “Zoho Creator”, Accessed on April 30, 2009, http://creator.zoho.com/.
[60] Wikipedia, “AppML”, Accessed on April 30, 2009, http://en.wikipedia.org/wiki/AppML.
[61] D. Thomas and D.H. Hansson, “Agile web development with rails,” The facets of Ruby series.
[62] S.J. Mellor and M.J. Balcer, Executable UML: A Foundation for Model-Driven Architecture, Addison-Wesley Professional, 2002.
[63] B. Lucas, C. F. Wiecha, “Collage: A Declarative Programming Model for Compositional Development and Evolution of Cross-Organizational Applications,” W3C Workshop on Declarative Models of Distributed Web Applications, 2007.
[64] A. van Deursen and J. Visser, “Domain-specific languages: an annotated bibliography,” ACM SIGPLAN Notices, vol. 35, 2000, pp. 26-36.
[65] P. Norvig, D. Cohn, “Adaptive software,” PC AI, vol. 11, 1997, pp. 27-30.
[66] C. Shapiro and H.R. Varian, Information rules, Harvard Business School Press Boston, Mass, 1999.
[67] R. Kowalski, “Algorithm= logic+ control,” Communications of the ACM, vol. 22, 1979, pp. 424-436.
[68] V. Allee, “A Value Network Approach for Modeling and Measuring Intangibles,” Transparent Enterprise, Madrid, November, 2002.
[69] Hendryx, Stan “Model-Driven Architecture and the Semantics of Business Vocabulary and Business Rules” , Hendryx & Associates, 2005
[70] Google, "Google Data APIs (Beta) Developer's Guide," Online at: http://code.google.com/apis/gdata/index.html, Accessed: 25/10/2007.
[71] Amazon, “Amazon Simple Storage Service (Amazon S3),” 2007, Online at: http://www.amazon.com/gp/browse.html?node=16427261, Accessed: 25/10/2007

[72] P. Castro, “Overview: Microsoft Codename Astoria,” Microsoft Corporation, 2007, Online at: http://astoria.mslivelabs.com/Overview.doc, Accessed: 25/10/2007.
[73] Microsoft Corporation, “Web Structured, Schema’d & Searchable (Web3S)”, 2007, Online at: http://dev.live.com/livedata/web3s.pdf, Accessed: 25/10/2007.
[74] IBM, “Project Zero”, Accessed on April 30, 2009, http://projectzero.org
[75] J. Gregorio, B. de Hora, “The Atom Publishing Protocol,” The Internet Engineering Task Force, 2007, Online at: http://tools.ietf.org/html/rfc5023, Accessed: 25/10/2007
[76] Y. Goland, E. Whitehead, A. Faizi, D. Jensen., “HTTP Extensions for Distributed Authoring--WEBDAV,” Microsoft, UC Irvine, Netscape, Novell. Internet Proposed Standard Request for Comments (RFC), vol. 2518, 1999.
[77] R.T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” University of California - Irvine, 2000.
[78] L. Richardson and S. Ruby, “RESTful Web Services,” O'Reilly Media, Inc., 2007.
[79] R.Fielding, J.Gettys, J.Mogul, H.Frystyk, T. Berners-Lee., “Hypertext Transfer Protocol--HTTP/1.1. RFC 2616”, The Internet Engineering Task Force, 1999.
[80] Open Philosophies for Associative Autopoietic Digital Ecosystems, 2008. “Automatic code structure and workflow generation from natural language models”. URI: http://files.opaals.eu/OPAALS/Year_2_Deliverables/WP02/D2.2.pdf
[81] Linehan, M.H., “SBVR Use Cases,” Proceedings of the International Symposium on Rule Representation, Interchange and Reasoning on the Web, Springer-Verlag Berlin, Heidelberg, 2008, pp. 182-196.
[82] Date, C. J., "All for One, One for All, Part 2: How Many Cases Are There?" Business Rules Journal, Vol. 7, No. 12 Dec. 2006, URI: http://www.BRCommunity.com/a2006/b324.html
[83] Moschoyiannis, S., Marinos, A., Krause, P., “Generating SQL Queries from SBVR Rules”, in G. Governatori, J. Hall, and A. Paschke (Eds.): RuleML 2010, To Appear.
[84] J. Gregorio, “URI Template”, Internet Draft draft-gregorio-uritemplate-04, March 2010. URI: http://tools.ietf.org/html/draft-gregorio-uritemplate-04
[85] Berners-Lee, T., Fielding, R., Masinter, L., "Uniform Resource Identifier (URI): Generic Syntax", IETF RFC 3986, URI: http://tools.ietf.org/html/rfc3986
[86] Mexis, D., "HTTP to Database Connectivity - Making Databases Available on the Web", MSc Disseertation, 2010, University of Surrey
[87] Lu, J., "RESTful Data Services and their Applications", MSc Disseertation, 2009, University of Surrey
[88] Gliozeris, G., "Taxonomic Web Database for IAH Pirbright", Undergraduate Disseertation, 2010, University of Surrey
[89] Google, “Protocol Reference - Google Data APIs - Google Code.” URI: http://code.google.com/apis/gdata/docs/2.0/reference.html Accessed: 31/3/2009
[90] Sun Microsystems, “The APIs for the Sun Cloud — Project Kenai.” URI: http://kenai.com/projects/suncloudapis Accessed: 31/3/2009
[91] Cabrera, L.F., Copeland, G., Feingold, M., Freund, R.W., Freund, T., Johnson, J., Joyce, S., Kaler, C., Klein, J., Langworthy, D., Little, M., Nadalin, A., Newcomer, E., Orchard, D., Robinson, I., Shewchuk, J., Storey, T., “Web Services Coordination (WS-Coordination),” Aug. 2005
[92] Furnis, P., and Green, A., “Choreology Ltd. Contribution to the OASIS WS-TX Technical Committee relating to WS-Coordination, WSAtomicTransaction and WS-BusinessActivity”. November 2005
[93] Gray, J. & Reuter, A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1993
[94] Recordon, D., Reed, D., “OpenID 2.0: a platform for user-centric identity management,” Proceedings of the second ACM workshop on Digital identity management, ACM New York, NY, USA, 2006, pp. 11-16.
[95] Bray, T., “ongoing · The Sun Cloud.”, URI: http://www.tbray.org/ongoing/When/200x/2009/03/16/Sun-Cloud Accessed: 31/3/2009
[96] Fielding, R., “It is okay to use POST » Untangled.” URI: http://roy.gbiv.com/untangled/2009/it-is-okay-to-use-post Accessed: 31/3/2009
[97] Bernstein, P.A., Hadzilacos, V., and Goodman, N. Concurrency control and recovery in database systems. Addison-Wesley, Boston, MA, USA, 1987
[98] World Wide Web Consortium, “Web Services Architecture”, W3C Working Group Note 11 February 2004. URI: http://www.w3.org/TR/ws-arch/. Accessed: 11/6/2010.
[99] Waldo, J., Wyant, G., Wollrath, A., and Kendall, S., “A note on distributed computing”, Technical Report SMLI TR-94-29, Sun Microsystems Laboratories Inc., November 1994.
[100] Microsoft, “UDDI Business Registry Shutdown FAQ”, URI: http://uddi.microsoft.com/about/FAQshutdown.htm, Accessed 30/9/2010
[101] Barr, J., “REST and SOAP”, 2006, URI: http://aws.typepad.com/aws/2006/12/rest_and_soap.html, Accessed: 30/9/2010
[102] Google, “Well Earned Retirement for SOAP Search”, 2009, URI: http://googlecode.blogspot.com/2009/08/well-earned-retirement-for-soap-search.html
[103] World Wide Web Consortium, “W3C Semantic Web Activity”, URI: http://www.w3.org/2001/sw/, Accessed: 11/6/2010.
[104] Berners-Lee, T., “Linked Data”, (2006) URI: http://www.w3.org/DesignIssues/LinkedData.html, Accessed: 11/6/2010.

[105] RSS Advisory Board, “RSS 2.0 specification”, URI: http://www.rssboard.org/rss-specification, Accessed: 30/9/2010
[106] Google, “PubSubHubbub”, URI: http://code.google.com/p/pubsubhubbub/, Accessed: 30/9/2010
[107] Apple, “Making a Podcast”, URI: http://www.apple.com/itunes/podcasts/specs.html, Accessed: 30/9/2010
[108] World Wide Web Consortium, “SPARQL Query Language for RDF”, URI: http://www.w3.org/TR/rdf-sparql-query/, Accessed: 11/6/2010.
[109] World Wide Web Consortium, “SPARQL Update: A language for updating RDF graphs” (2008), URI: http://www.w3.org/Submission/SPARQL-Update/, Accessed: 11/6/2010.
[110] Lee, D., Chu W.W., “Comparative analysis of six XML schema languages”, ACM SIGMOD Record, v.29 n.3, p.76-87, September 2000.
[111] Marinos, A., Krause, P., “What, not How: A generative approach to service composition”, IEEE Conference on Digital Ecosystems Technologies 2009 (DEST 2009)
[112] Moschoyiannis, S., Marinos, A., Krause, P., “Generating SQL Queries from SBVR Rules”, RuleML 2010, *to appear.*
[113] Warth, A. "Experimenting with programming languages" Technical report, Viewpoints Research Institute, 2008.
[114] Warth, A. Piumarta, I., "Ometa: an object-oriented language for pattern matching" DLS’07 DSL, 2007.
[115] World Wide Web Consortium, “Web SQL Database”, 2010, URI: http://dev.w3.org/html5/webdatabase/, Accessed: 20/8/2010.
[116] Lu, J., “HDBC: A RESTful HTTP-to-Database-Connector”, URI: http://code.google.com/p/rest-hdbc/
[117] Marinos, A., Wilde, E., Lu, J., “HDBC: RESTful Access to Relational Databases”, WWW 2010, Raleigh, NC, USA, pp. 1157-1158
[118] Hwaci, “SQLite”, URI: http://sqlite.org/, Accessed: 30/9/2010
[119] Mozilla Labs, “Bespin”, URI: https://bespin.mozillalabs.com/, Accessed: 30/9/2010
[120] ECLiPSe, “The ECLiPSe Constraint Programming System”, URI: http://eclipseclp.org/, Accessed: 30/9/2010
[121] Briscoe, G., Marinos, A., “Digital Ecosystems in the Clouds: Towards Community Cloud Computing”, IEEE Conference on Digital Ecosystems Technologies 2009 (DEST 2009)
[122] Marinos, A., Briscoe, G., “Community Cloud Computing”, CloudCom 2009, LNCS 5931, 2009