OPAALS PROJECT

Contract n° IST-034824

WP3: Autopoietic P2P Networks

Del 3.12 – Infrastructure integration, Distributed identity, Trust, accountability and RESTful Transaction model

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

Contract Number: IST-034824

Project Acronym: OPAALS

Deliverable N°: D3.12

Due date: January 2010

Delivery Date: September 2010

Short Description:

This document is the last in a series of deliverables (D3.1, D3.2, D3.3, D3.6, D3.10) that aim to frame the infrastructural specifications in Digital Ecosystems research. As it has been shown (D3.6) in OPAALS the technological and social concerns in providing the necessary digital infrastructure are not treated as distinct but as part of the same continuum. This deliverable sets out the key concepts and characteristics of the current integration framework and clarifies the future work plans. The key aspects of the core DE architecture, in terms of P2P and transaction support, as well as formal analysis and current implementation roadmaps, are described from a computer science viewpoint.

The process of putting the design models and the different point of views are discussed briefly. In this deliverable we show how this process has been set up and demonstrate the consensus reached on key aspects of the core DE architecture.

In order to provide a complete overview, this document contains several chapters (2-5) that contain material that has already presented in earlier deliverables, although subject to a number of refinements and corrections. New material is presented in Chapters 1, 6, 7 and 8.

Author: Surrey

Partners contributed: IPTI, TI

Made available to: All

VERSIONING		
VERSION	DATE	NAME, ORGANIZATION
0.1	**7/2010**	**A. RAZAVI (SURREY)**
0.2	**8/2010**	**A. RAZAVI (SURREY)**
0.9	**9/2010**	**A. RAZAVI (SURREY), P. MALONE (WIT), P. SIQUEIRA (IPTI), F. SERRA (IPTI)**
1.0	**9/2010**	**P. KRAUSE (SURREY)**

Quality check

Internal Reviewers: Paolo Dini (LSE)

Dependencies:

Achievements	Discussion of the significance of digital ecosystem thinking to business, especially SMEs in the developed and developing countries; Bridge from theory to design of the OPAALS DE infrastructure; Development of Flypeer platform and integration of distributed identity, trust and accountability; Development of a RESTful transaction model; Overview of the vision for SBRV and REST as technologies to support agile development and deployment of web services; Definition of algorithm to generate SQL queries from SBVR rules.
Work Packages	WP12: Task 12.8 – OS principles of communication and collaboration, Task 12.9, 12.10 – Business models in DEs WP11: Task 11.1 – collaboration and innovation in the Knowledge Economy, Task 11.4 – social innovation networks WP10: Task 10.10 – visualisation of P2P infrastructure WP5: Task 5.6 – adding e-business support in current DBE, Task 5.7 – integration of P2P network services into the DE
Partners	TI, BCU, UniKassel, IPTI, NUIM, UL, IITK, TUT, UNIVDUN
Domains	Computer Science domain: P2P networks, interactions, long-running transactions, distributed systems and networks, redundancy, diversity, consistency, concurrency, design for failure, formal semantics, behaviour patterns, lock mechanisms, formal analysis, distributed identity, distributed trust. Social science domain: participation, power, control, technology infrastructure for competitive advantage, monopoly, lock-in, proprietary software / platforms, knowledge, openness, reciprocity, context-dependent trust, identity. Natural Science domain: simple reference to key analogies with ecosystems in nature and their key features found mostly in studies of biodiversity.

Targets	Computer, social and natural science researchers, SMEs, business analysts. Computer science communities: database, transactions, P2P networking and applications, formal methods. Social science: language, socio-economics, governance, power and control, identity and trust.
Publications	Chapters 1,2,4,5,6,7 are based on publications that are referenced at the beginning of each chapter.
PhD Students	Work of Chapters 1 & 2 were developed during Amir Razavi's PhD. Work of Chapters 6 & 7 were developed during Alexandros Marinos' PhD.
Outstanding features	Integrated Flypeer platform with sound theoretical foundation. Strong advances in the declarative use of SBVR
Disciplinary domains of authors	All authors are from the Computing domain, but with strong engagement with Social Scientists and Biologists through the ICT.

TABLE OF CONTENTS

EXECUTIVE SUMMARY

This document is the last in a series of deliverables (D3.1, D3.2, D3.3, D3.6, D3.10) that aim to frame the infrastructural specifications in Digital Ecosystems research. As it has been shown (D3.6) in OPAALS the technological and social concerns in providing the necessary digital infrastructure are not treated as distinct but as part of the same continuum. This deliverable sets out the key concepts and characteristics of the current integration framework and clarifies the future work plans. The key aspects of the core DE architecture, in terms of P2P and transaction support, as well as formal analysis and current implementation roadmaps, are described from a computer science viewpoint.

In order to provide a complete overview, this document contains several chapters (2-5) that contain material that has already presented in earlier deliverables, although subject to a number of refinements and corrections. New material is presented in Chapters 1, 6, 7 and 8.

AIMS & OBJECTIVES OF THE REPORT

This report collects together material that was part of the journey that led to the final results of D3.15. Most of the content of the report has been published in peer-reviewed conferences. It's value is primarily in documenting the work that was performed. D3.15 will provide a more definitive reference, however.

Chapter 1 (presented at IEEE DEST 2010) discusses the case for Digital Ecosystems and captures the rationale behind the OPAALS framework. Chapter 2 (Presented at ETAPS 2009, and published in Electronic Notes in Theoretical Computer Science, vol. 238) then moves on to discuss the details of the design itself, especially with regard to the support for long-term business transactions.

Chapter 3 moves to documentation of the Fypeer P2P code-base. Here we specifically focus on the integration of authentication and WIT's Identity Model into Flypeer. The complete integration of Distributed Identity, Trust and Accountability is then covered in Chapter 4. This work is documented in full in D5.5, and was a major achievement of OPAALS.

Chapter 5 sees the beginning of our move from WSDL/SOAP based architecture to a RESTful architecture. Specifically, we develop a RESTful transaction model. This chapter is based on a report that was not formally published, but which was made available on the Web, and which has received recognition from a number of leading industrial players in the RESTful web-services arena.

The report then (Chapter 6, which was presented at RuleML 2010) moves into an overview of our vision for a rule-driven (SBVR) architecture for distributed systems. This is more of a position paper. However, a concrete step in this direction has been the development of an algorithm for generating SQL queries from SBVR rules. This provides a firm foundation for the automated generation of information systems, and is covered in Chapter 8 (presented at RuleML 2010). Substantiation of this whole REST/SBVR agenda has been a major achievement of OPAALS and is covered in much greater detail in D3.15.

1 INTRODUCTION (FROM BUSINESS ECOSYSTEMS TOWARDS DIGITAL BUSINESS ECOSYSTEMS) SURREY

(This chapter has been published in IEEE-DEST 2010).

Small and Medium Enterprises (SMEs) constitute one of the most important parts of any developed and developing economy. They are the largest sources of employment and represent a major part of any nations GDP. In many cases, they are also the source of innovation.

Supporting Small and Medium Enterprises are therefore one of the major economical policies for most international players. The innovation, growth and sustainability of the economy have a direct relationship with the vitality and success of these small businesses.

"*Micro, small and medium-sized enterprises (SMEs) play a central role in the European economy. They are a major source of entrepreneurial skills, innovation and employment. In the enlarged European Union of 25 countries, some 23 million SMEs provide around 75 million jobs and represent 99% of all enterprises.*" [1]

The innovations, affiliations, competition and collaborations of these enterprises relies on several factors from a healthy economy and this is one of the reasons, despite their crucial roles in other aspects of society, for support and considerable investment, provided in a large scale;

"*...support for SMEs is one of the European Commission's priorities for economic growth, job creation and economic and social cohesion.*" [1]

Maintaining an optimised model for supporting these enterprise relationships needs a compound representation. One of the well-known metaphoric attempts for grasping this complexity is '*business ecosystem*' that describes the business environment as an economic community which "*is supported by a foundation of interacting organizations and individuals--the organisms of the business world.*" [2].

With the introduction of the Internet and increased connectivity, "*Digital Business Ecosystems*" (or Digital Ecosystems in general) have been introduced as an evolution of this model [3]. In this context, Business has measured as "*An economic community supported by a foundation of interaction organisation and individuals – the organisms of business world*". [4].

Inspired by the natural Ecosystems, Digital Ecosystems are seen as having four properties: '***Interaction and engagement***', '***Balance***', '***Domain clustered and loosely coupled***' and '***Self-organisation***' [5]. As a result, a Digital Ecosystem needs a self-organising digital infrastructure aimed at creating a digital environment for networked organisations that supports a loosely coupled business interaction between them when stability and sustainability of this dynamic environment is maintained. In contrast with conventional models (client-server, Peer-to-Peer or centralised service oriented architecture), a Digital Ecosystem is an open community without any centralised control [5], [6]. The loose coupling and local autonomy of participants, with the lack of centralised control, will result in distributed coordination which creates a fully distributed environment [7].

The Digital Business Ecosystems is concerned with building an open environment, through which businesses, in particular small to medium enterprises (SMEs), can interact within a reliable environment. The aim is to provide access to arbitrary services that help compose together to meet particular needs of the various partners. This collaborative software environment is being targeted primarily towards SMEs, who will be able to concatenate their offered services within service chains formulated on a digital ecosystem [8], [9].

1.1 BUSINESS ECOSYSTEMS

In 1993 James F. Moore by using the metaphoric concept of "*business ecosystem*" described the business environment as an economic community which "is supported by a foundation of interacting organizations and individuals--the organisms of the business world." [2].

To the extent that the business population of each business ecosystem is concerned, the majority is composed of Small and Medium size businesses (SMEs) along with a few large ones, the so called *Keystones*. Marco Iansiti and Roy Levin compare the role of these keystone companies to those of keystone species in nature [10]. They argue that we live in an interconnected world, the landscape of which is made of a network of networks, with keystones at the hubs and niche players surrounding the hubs.

Before industrial revolution (1760-1840) [11], the speed of circulation and revision of new technologies was quite slow and the actual metaphorical comparison has not been highly relevant. From improvements in steam engines to invention of telegraph in 1830s and later the telephone in the 1860s, reduction in distance also opened new markets, allowing manufacturers and producers to increase production capacity, which in turn led to increasing size of these companies. Undeniably the origins of the modern diversified corporations can be traced back to the creation of large corporations at the beginning of the last century, when mass-production (brought about by innovations in work methods and mechanical automation) allowed many companies to grow rapidly and prosper at a rate that had never been seen before in history. These large organizations played the role of '*Keystones*' of the time.

By deepening the competition, marketing and customer demand became more important. Consequently these companies began to change their focus to economies of scope [12]; that is, to producing "different products" faster, better and cheaper. The improvement in internal efficiency was not limited to production technologies. The management practices, routines and business processes were also examined and improved. As businesses, especially manufacturers, sought ways to reduce their costs and improve their responsiveness, they adopted new concepts such as Just-In-Time (JIT) systems, which demanded a closer cooperation (i.e., timely access to information and products) between producer, suppliers and customers.

Intranet and extranets were the answer to many of these problems. Intranets and extranets allowed companies to connect their offices, plants, suppliers and customers into closed networks. This allowed them a better overview of their operations while strengthening the coupling between themselves and their customers and suppliers. This made it a perfect tool for erecting entry barriers around industries. In addition these new

technologies gave large corporations geographical independence.

1.2 Internet; New era and New type of business

By developing internet, as any other new technology, a group of start-ups and SMEs move in to take advantage of the new opportunities. Technological evolutions usually give birth to new industries where start-ups and existing SMEs, because of their size (agility) enter first, becoming the first movers. First movers generally have the advantage of registering patents, establishing brand names, changing the economics of the market making it difficult for others to enter and compete and so on. In young industries, first movers have the ability to, in a very short time, become industry leaders or keystones.

The Internet is a network of computers. This network has no determined structure and expands in a random fashion. New nodes (computers) constantly connect and disconnect themselves to the network via links (vertices) to other computers. The general network topology, and its growth in term of distribution of links on one hand and the coordination of business interaction on the other, play major roles for the new business communities.

1.2.1 Networking and Connections

Studies [13] have shown that the topology of this network is governed by a power-law distribution (Fig. 1). This means that often a few nodes evolve in such a way as to attract a large number of links while many nodes continue to exist with only a few links. This gives those nodes (with large number of links) and companies that own them a disproportionate power in the network. First movers have managed early on to become large hubs. SMEs in this sector have very little chance of becoming hubs. The entry barriers in this sector are getting higher and higher.

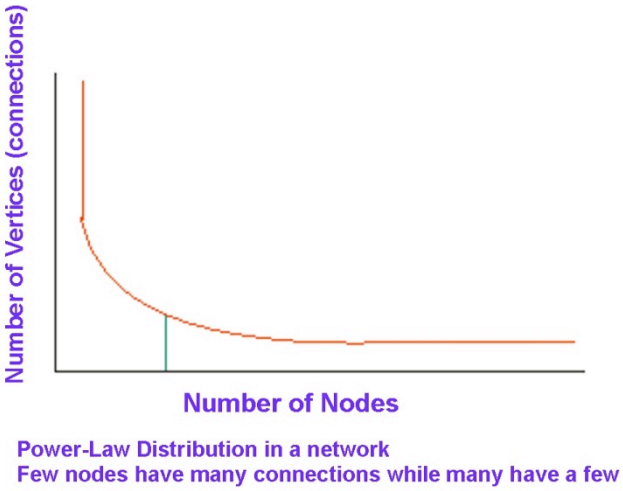


FIG. 1 DISTRIBUTION DEGREE IN SCALE-FREE NETWORK

There are several problems with having these hubs. The first is that they are costly to maintain. The second is that of increasing traffic and traffic congestion during peak time. To answer the increasing traffic, one has to keep increasing the capacity of the hub, leaving substantial free capacity in the off-peak times, which results in an increase in the overhead

costs. In addition these hubs present major targets for intentional or unintentional attacks. Any disruption at these hubs will result in major fragmentation of the network. Meanwhile there is another relevant discussion about the digital age and its interaction model, which comes back to coordination. These two cause an unbalanced environment which despite discussions about fairness will not provide the necessary sustainability in its environment and reliability for its interaction model. In the next section, we review the role of coordination in an interaction model and the state of the art in that respect.

1.2.2 BUSINESS INTERACTIONS AND COORDINATION

The purpose of any business network is to enable networked organisations to engage in business interactions (transactions) that realise their core business activities. The current business interactions on the Internet predominately works on SOC (Service-Oriented Computing) [14], [15], which aims to enable applications from different providers to be offered as services that can be used, composed, and coordinated in a *loosely coupled* manner. In this paradigm, services are fundamental elements for developing solutions. They are platform-agnostic computational elements that support rapid, low-cost composition of distributed applications. Services perform functions, which can be anything from simple requests to complicated business processes.

The actual architectural approach of SOC is called SOA (Service-Oriented Architecture) and is particularly applicable when multiple applications running on varied technologies and platforms need to communicate with each other. In this way, enterprises can mix and match services to perform business transactions with minimal programming effort.

SOA is a way of reorganizing software applications and support infrastructure into an interconnected set of services, each accessible through standard interfaces and messaging protocols. In this way, an application is the orchestration of many services that have different dependencies on each other. This is often referred to as a work flow or business process, which usually means a collection of transactions. Transactions require data consistency and recoverability. An example of data consistency would be that regardless of a transaction deployment with any rate of concurrency, the results should be consistent. By recoverability, we mean that regardless of any failure, one can roll-back the transaction to its initial state.[15].

These two are the main responsibilities of the coordinator [16]. In theory, either party to the transaction can assume the role of coordinator. However the current systems place a heavy demand on computational complexity [8], making it very difficult for the majority of SMEs to assume this role. Therefore, the current system requires that a third party with sufficient resources assumes this responsibly. Here we have to mention that the coordinator enjoys many privileges [8] without the associated responsibilities (at least, the full responsibility of recovering a failure without involving participants).

The role and function of the coordinator is determined by the WS-coordination protocol as defined in Windows Communication Foundation (WCF) [17]. This protocol is used by the two existing and competing industry frameworks: Business Transaction Protocol (BTP : supported by Oracle, Sun Microsystems, Choreology Ltd, Hewlett-Packard Co., IPNet, SeeBeyond Inc. Sybase, Interwoven Inc., Systinet and BEA System in term of OASIS Business

Transaction Protocol [18]) and Web Services transaction (WS-AtomicTransactions [19] and WS-BusinessActivities [20]: supported by Microsoft, Hitachi, IBM, IONA, Arjuna Technologies and BEA Systems).

Despite all claims and advertisements, both coordination protocols for business transactions violate loose coupling on one side and offer just one pattern of behaviour (clarifying the completion protocol in a transaction and determining the recovery method in respect to that protocol [21]) for participants of transactions on the other [22]. Violating loose coupling, is not only contrary to SOA, but also gives the coordinator the opportunity to estimate the local state of SMEs (in realistic terms, this tight coupling between coordinator and participants means the coordinator is aware of the local state of participants at any given time during or after the transaction).

This tight coupling results in the participants (e.g. SMEs) losing their local autonomy (their local states of businesses will be visible to the coordinator). At the same time the pattern of behaviour supported by the coordinator framework (do-compensate [23]), forces participants to apply specific methods of fault recovery during a transaction failure [23], [24], [25]. This not only enforces a specific problem solving method in event of a failure (which is known by the companies supporting the protocols) but also imposes the responsibility of sorting the failure out to participants. By having an overview of the participants' recovery pattern behaviour and their local state, the coordinator can construct or simulate the participants' business models [24], [25].

1.3 Paradigm shift and the Digital Business Ecosystems concept

Since the necessity for providing a new model, concept and definition, the research community, different organisations and governments have started to propose a new conceptual framework for digital ecosystems. In 2002, the first intuition has been proposed by Nachira in the European Commission. This important discussion paper has been finalised in a new conceptual framework [3], called Digital Business Ecosystems:

"*The synthesis of the concept of Digital Business Ecosystems emerged in 2002 by adding 'digital' in front of Moore's (1996) "business ecosystem" in the Unit ICT for business of Directorate General Information Society of the European Commission.*"

In this conceptual framework, Business is considered as "*An economic community supported by a foundation of interaction organisation and individuals – the organisms of business world*". [4]. This economic community produces goods and services of value to customers, who themselves are members of the ecosystem". [2] A wealthy ecosystem sees a balance between cooperation and competition in a dynamic free market [3]. The European Commission has specified the explicit aims of such a Digital Business Ecosystem:

"*A Digital Business Ecosystem results from the structurally coupled and co-evolving digital ecosystem and business ecosystem. A network of digital ecosystems, will offer opportunities of participation in the global economy to SMEs and to less developed or remote areas. These new forms of dynamic business interactions and global co-operation among organisations and business communities, enabled by digital ecosystem technologies, are deemed to foster local economic growth. This will preserve local knowledge, culture and identity and*

contribute to overcome the digital divide." [26]

This conceptual framework tries to solve the current challenges of businesses (recall Previous section). That is why the projects funded by the EC have focused to provide a modern dynamic network to support business interactions of small and medium enterprises without relying on a large organisation of any sort.

Developing this conceptual framework has not been limited to Europe. The Digital Ecosystems and Business Intelligence Institute in Australia [27] as a leading research institute on Digital Ecosystems studies, not only has developed and constituted Digital Ecosystems. One of the brightest descriptions of ecosystem in this term can be found in Chang and West approach:

"*There are four essences of ecosystems: (1) Interaction and engagement (2) Balance (3) Domain clustered and loosely coupled (4) Self-organisation*" [5].

1.3.1 Small and Medium Enterprises, local autonomy and loose coupling

As with the DEBII (Digital Ecosystems and Business Intelligence Institute) vision [28], the European Commission's projects have focused on loose coupling between participants of a Digital Ecosystem [29]. Furthermore, the local autonomy of small and medium enterprises for deploying transactions has been persisted. In this, any collaboration (in terms of business transactions) will be based on demand of participants rather than on any external decision. In addition, the preservation of local autonomy and avoidance of any external control or pressure has been considered within the definition of requirements for any business transaction [7], [29]. We can see the similarity of these important requirements in DEBII's definition, when it focuses on a loosely coupled, demand driven environment when each participant (here digital species) has a specific objective (benefit or profit):

"*We define a Digital Ecosystem as a loosely coupled, demand driven collaborative environment where each digital species is proactive and responsive for its own benefit or profit*" [6]

Boley and Chang [30] has described loose coupling as a freely bound open relationship between participants, when the term is opposite to a tightly coupled relationship (where each party is heavily dependent on one another and the roles are predefined). In terms of '*Demand Driven*', their definition illustrates the deference between the driving force coming from outside '*push-in*' rather than '*pull-in*' [6]. And by example, they have explained the importance of the real motivation from participants rather than a force from outside.

It seems this framework potentially can avoid two critical problems with the current business ecosystems on the internet. For finding more evidence, we try to explore how Digital Business Ecosystem confront the business activities' challenge and what is the solution for the network topology.

1.3.2 Business activities of Digital Business Ecosystems

The aggregations of the business activities taking place between the different partners create several virtual business networks. When these business activities are conducted by means of long-term transactions which involve the execution of services from different service providers, these result in the creation of temporary networks interconnecting the

participating organisations [22]. These are typically separate disconnected networks resulting from transactions between participants, but overlaps may exist due to some participants being involved in more than one transaction in the same or different business domain (Fig. 2).

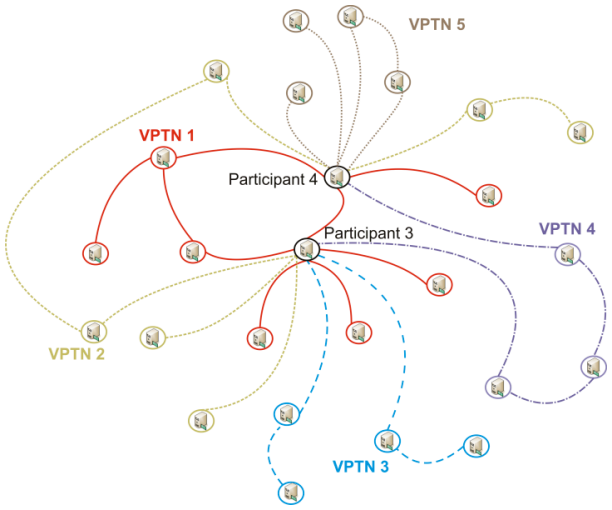


FIG. 2 VPTNS IN DIGITAL BUSINESS ECOSYSTEMS

In term of this interaction model the use of a P2P solution in the context of digital ecosystems; have been presented a fully distributed support within the digital ecosystem initiative [22]. Each participants have its own software agent which execute transactions in a local coordinator [21]. The transaction model feeds into the corresponding Virtual Private Transaction Networks (VPTNs) which are the main building block for the underlying P2P network that supports these complex interactions between participating entities. Naturally one may expect a scale-free network topology but we review the actual structure of such a network.

1.3.3 Network and connectivity in Digital Business Ecosystems

Digital Ecosystems, instead of relying on Super peers or Hubs, apply a stability function to create Virtual Super Peers (VSPs) which are effectively play role of major hubs in the traditional scale-free networks [31]. These can provide the desired stability for the network. The strong connection between the virtual super peers themselves on one hand and the connection between them and their nodes decrease the probability for fragmentation. Depending on the level of reliability required for the network, it is possible to include further redundant stable platforms from each available time zone. In this manner, the good connectivity can cause more reliable transactions at the VPTNs level [9]. As a result, better and more egalitarian system of coordinating the networks by using Virtual Super Peers (VSPs) has been proposed. The distributed local agents in digital business ecosystems organise the network regularyly where all nodes' stability is constantly monitored and graded. Based on their stability, a cluster of stable nodes are identified. These clusters will then be assigned the role of network peers or Hubs (Fig. 3).

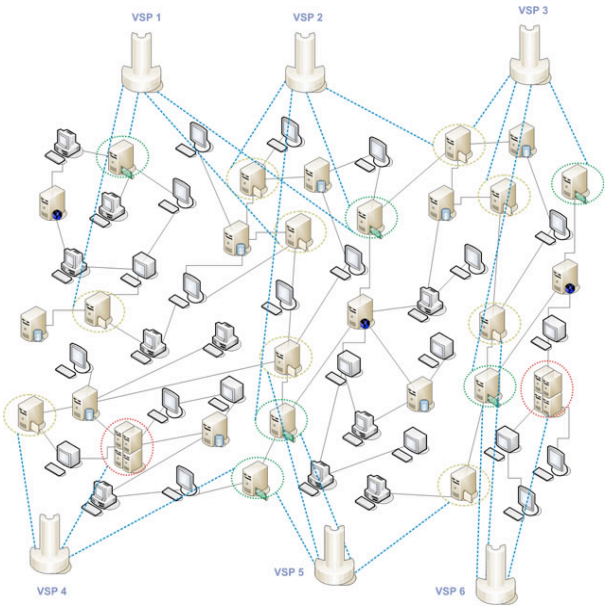


FIG 3. VSPS IN DIGITAL BUSINESS ECOSYSTEMS

This model offers two major advantages over the existing system of hubs. Having a better and more SME friendly network is the first step in addressing the problem of entry barriers in the Internet. The second problem is the problem of standards; especially the problem of influence of large enterprises or Keystones in shaping and defining standards that can be used as part of the large enterprises' competitive strategy.

1.4 Conclusion and Further works

"*Business Ecosystems*" metaphoric framework, tries to formulise the business world to a practical complex model. When crucial role of Keystones in metaphoric definition of "Business Ecosystems" can be considered as a downside for an economical ecosystem, "Digital Business Ecosystems" try to solve the problem (at least) in Digital world. The loosely-coupled interaction model, demand driven properties for business and self-organising characteristic of Digital Business Ecosystems, provide an autonomous model, which in contrast with the conventional centralised model for business activities, provides an agent-based distributed environment when the driving force for optimisation or self-organisation comes from inside of each participant (agent) rather than outside control.

The ongoing work for implementation, and open source frameworks for such a model, introduce new age for information technology which may go behind the boundary of digital world and effect different aspects of the society.

References

[1] European Commission, *Recommendation 2003/361/EC: SME Definition*, ENTERPRISE AND INDUSTRY PUBLICATIONS (European Commission), 2005.

[2] J.F. Moore, "Predators and Prey: A New Ecology of Competition," *HARVARD BUSINESS REVIEW*, vol. 71, 1993, pp. 75-83.

[3] F. Nachira, “Towards a Network Of Digital Business Ecosystems Fostering the Local Development,” *European Commission Discussion Paper. Bruxelles*, 2002.

[4] F. Nachira, P. Dini, and A. Nicolai, “A Network of Digital Business Ecosystems for Europe: Roots, Processes and Perspectives,” *Digital Business Ecosystems. European Commission, Bruxelles. www. digital-ecosystems. org/book/de-book2007. html*, 2007.

[5] E. Chang and M. West, “Digital Ecosystem-A next generation of the collaborative environment,” *The Eight International Conference on Information Integration and Web-Based Applications & Services, books@ ocg. at*, 2006, pp. 3-23.

[6] E. Chang, M. West, and M. Hadzic, “A Digital Ecosystem for Extended Logistics Enterprises,” *Proceedings of the 11th International Workshop on Telework*, 2006.

[7] S. Moschoyiannis and M.L. Darking, “Consensus detailed architecture of the (OPAALS) Digital Ecosystems ,” 2008.

[8] A.R. Razavi, P. Krause, and S. Moschoyiannis, “Deliverable D24.5: DBE Distributed Transaction Model,” 2006.

[9] A.R. Razavi, “DIGITAL ECOSYSTEMS, A Distributed Service Oriented Approach for Business Transactions,” PhD Thesis, University of Surrey, 2009.

[10] M. Iansiti and R. Levien, *The Keystone Advantage: What the New Dynamics of Business Ecosystems Mean for Strategy, Innovation, and Sustainability*, Harvard Business School Press, 2004.

[11] M. Iansiti and R. Levien, *The Keystone Advantage: What the New Dynamics of Business Ecosystems Mean for Strategy, Innovation, and Sustainability*, Harvard Business School Press, 2004.

[12] A.D. Chandler, *Scale and Scope: The Dynamics of Industrial Capitalism*, Harvard University Press, 1990.

[13] A.L. Barabási, R. Albert, and H. Jeong, “Scale-free characteristics of random networks: the topology of the world-wide web,” *Physica A: Statistical Mechanics and its Applications*, vol. 281, 2000, pp. 69-77.

[14] M.P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, and B.J. Kramer, “Service-oriented computing: A research roadmap,” *Service Oriented Computing (SOC)*, 2006.

[15] M.P. Singh and M.N. Huhns, *Service-Oriented Computing: Semantics, Processes, Agents*, Wiley, 2005.

[16] L. Cabrera, G. Copeland, M. Feingold, R. Freund, T. Freund, J. Johnson, S. Joyce, C. Kaler, J. Klein, D. Langworthy, M. Little, A. Nadalin, E. Newcomer, D. Orchard, I. Robinson, J. Shewchuk, and T. Storey, “Web Services Coordination (WS-Coordination),” Aug. 2005.

[17] L. Cabrera, G. Copeland, J. Johnson, and D. Langworthy, “Coordinating Web Services Activities with WS-Coordination, WS-AtomicTransaction, and WS-BusinessActivity,” Jan. 2004.

[18] P. Furnis, S. Dalal, T. Fletcher, A. Green, A. Ceponkus, and B. Pope, “Business Transaction Protocol, version 1.1. 0 (November 2004),” *URL http://docs. oasis-open. org/business-transaction/business_transaction-btp-1.1-spec-cd-01. pdf.–(20.07. 2006)*,

2004.

[19] L. Cabrera, G. Copeland, M. Feingold, R. Freund, T. Freund, J. Johnson, S. Joyce, C. Kaler, J. Klein, and D. Langworthy, *Web Services Atomic Transaction (WS-AtomicTransaction)*, IBM, US: IBM , 2005.

[20] L. Cabrera, G. Copeland, M. Feingold, R. Freund, T. Freund, S. Joyce, J. Klein, D. Lang-WORTHY, M. Little, and F. Leymann, *Web Services Business Activity Framework (WS-BusinessActivity). 2005*, IBM DeveloperWorks, 2005.

[21] A.R. Razavi, S. Moschoyiannis, and P. Krause, "A Coordination Model for Distributed Transactions in Digital Business EcoSystems," *Digital Ecosystems and Technologies (DEST 2007), IEEE Computer Society Press, Los Alamitos*, 2007.

[22] A. Razavi, S. Moschoyiannis, and P. Krause, "An open digital environment to support business ecosystems," *Peer-to-Peer Networking and Applications, Springer New York*.

[23] P. Furnis and A. Green, "Choreology Ltd. Contribution to the OASIS WS-Tx Technical Committee relating to WS-Coordination, WS-AtomicTransaction, and WS-BusinessActivity (November 2005), Ihttp," *www. oasis-open. org/committees/download. php/15808*, 2005.

[24] F.H. Vogt, S. Zambrovski, B. Gruschko, P. Furniss, and A. Green, "Implementing Web Service Protocols in SOA: WS-Coordination and WS-BusinessActivity," *CECW*, vol. 5, 2005, pp. 21–28.

[25] A.R. Razavi, S. Moschoyiannis, and P. Krause, "Preliminary Architecture for Autopoietic P2P Network focusing on Hierarchical Super-Peers, Birth and Growth Models," 2007.

[26] European Commission, "Technologies for Digital Ecosystems - Innovation Ecosystems Initiative - Specific Aims of the Digital Business Ecosystem," 2008.

[27] DEBII, "DEBI Institute - Digital Ecosystems and Business Intelligence Institute," 2009.

[28] E. Chang, M. Quaddus, and R. Ramaseshan, *The vision of DEBI Institute: digital ecosystems and business intelligence*, DEBII, 2006.

[29] P. Dini, G. Lombardo, R. Mansell, A.R. Razavi, S. Moschoyiannis, P. Krause, A. Nicolai, and L.R. Len, "Beyond interoperability to digital ecosystems: regional innovation and socio-economic development led by SMEs," *International Journal of Technological Learning, Innovation and Development*, vol. 1, 2008, pp. 410-426.

[30] H. Boley and E. Chang, "Digital Ecosystems: Principles and Semantics," 2007, pp. 398-403.

[31] A. Razavi, S. Moschoyiannis, and P. Krause, "A scale-free business network for digital ecosystems," *Proceedings IEEE Digital Ecosystems and Technologies (IEEE-DEST 2008)*, 2008.

2 FROM THEORETICAL APPROACH TO THE IMPLEMENTATION (SURREY)

This chapter has been published at ETAPS

2.1 TRANSACTION CONTEXT

In previous work [1] we have described the use of a tree structure to represent transactions that involve the execution of services. This allows us to capture nested sub-transactions - the internal actions that need to take place in the course of execution of the transaction. To respect the loose-coupling of the underlying services, which is a basic premise of Service-Oriented Computing (SOC) [2], [3], each participant provides its services and requests services of others through a coordinator component. Its purpose is to manage the communication between the different participants' platforms and the deployment of the corresponding services.

Drawing upon the latest work on the SOC computing paradigm [4], we have considered different composition types which allow for various modes of service interaction in our model. Fig. 3 shows a transaction tree with five basic services - a1, a2 and a3 of a local platform with coordinator component CC1, b1 of CC2, and c1 of CC3 - whose order of execution is determined by the corresponding composition types [transaction context symbols are based on the notation first appeared on [5]].[6]

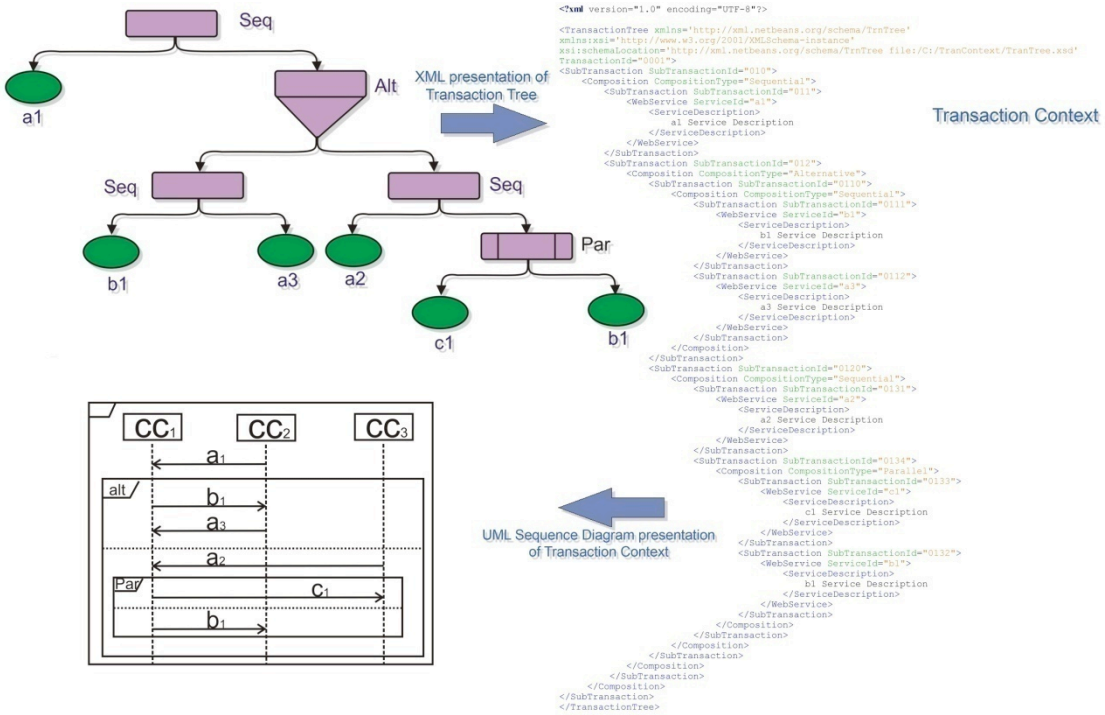


FIG. 1 TRANSACTION CONTEXT

We note that the example transaction given in Fig. 3 is an adaptation of that found in [7] and will be extended in this section to illustrate the key ideas as well as to show how these build on previous work in [8], [7].

A transaction tree determines the Participants and the respective services required for performing a business activity. In this sense, it sets the context of the conversation to follow and is issued by the Initiator of the transaction. In [7] we have provided a schema for describing transaction contexts. The derived XML description of the transaction tree of our example (given in Fig. 3) can be found following [9]. The service interactions implied by a transaction tree can be modelled using a UML interaction diagram. The sequence diagram in Fig. 3 shows the three coordinator components of the participants in the transaction and the required service invocations between them. It can be seen that the behavioural scenarios, as given by the corresponding sequence diagram, determine the order of execution of the participating components' services.

As mentioned earlier, in a transactional setting we also need to deal with faults that may arise at any stage during execution. These may be due to some service being unavailable (service failure or traffic bottleneck on the local platform) or some participant being temporarily disconnected due to network failure. We have seen that a long-running transaction should either complete successfully or not take place at all. So in the event of a failure, previous parts of the transaction that have already taken place should be 'undone' or be compensated for. We will mention this, after explaining the execution script of the transaction.

2.2 Transaction Script

In [7] we describe a formal language for long-running transactions that allows to determine the patterns of interaction the underlying service invocations should follow in order to guarantee a successful outcome.

A transaction *T* is associated with a set of coordinator components *C* and a set of actions *M*. Our interest is in the observable events on the coordinator components and thus actions can be understood as service invocations between the participating components, as shown for example in the scenario of Fig. 4. Hence, each component in *C* is associated with a set of actions which correspond to deploying (its own) or requesting (others') services. We denote this set by $\mu(i)$, for each $i \in C$, where $\mu : C \rightarrow \phi(M)$ and require that $\bigcup_{i \in C} \mu(i) \subseteq M$.

As can be seen in Fig. 3, a transaction has a number of activation or access points, namely the interfaces of the coordinator components participating in the interaction. Thus, instead of modelling the behaviour of a transaction by a sequential process, which would generate a trace of a single access point, we consider a number of such sequences, one for each component, at the same time. This draws upon Shields' vector languages [10] and leads to the definition of the so-called transaction vectors.

Transaction vectors. Let *T* be a transaction. We define V_T to be the set of all functions $\underline{v}$: $C \rightarrow M^*$ such that $\underline{v}(i) \in \mu(i)^*$.

By $\mu(i)^*$ we denote the set of finite sequences over $\mu(i)$. Mathematically, the set V_T is the Cartesian product of the sets $\mu(i)^*$, for each *i*. Effectively, transaction vectors are *n*-tuples of sequences where each coordinate corresponds to a coordinator component in the transaction (hence, *n* is the number of leaves) and contains a finite sequence of actions that

have occurred on (coordinator of) that component. When an action occurs in the transaction, that is to say when a service is called on a coordinator component, it appears on a new transaction vector and at the appropriate coordinate.

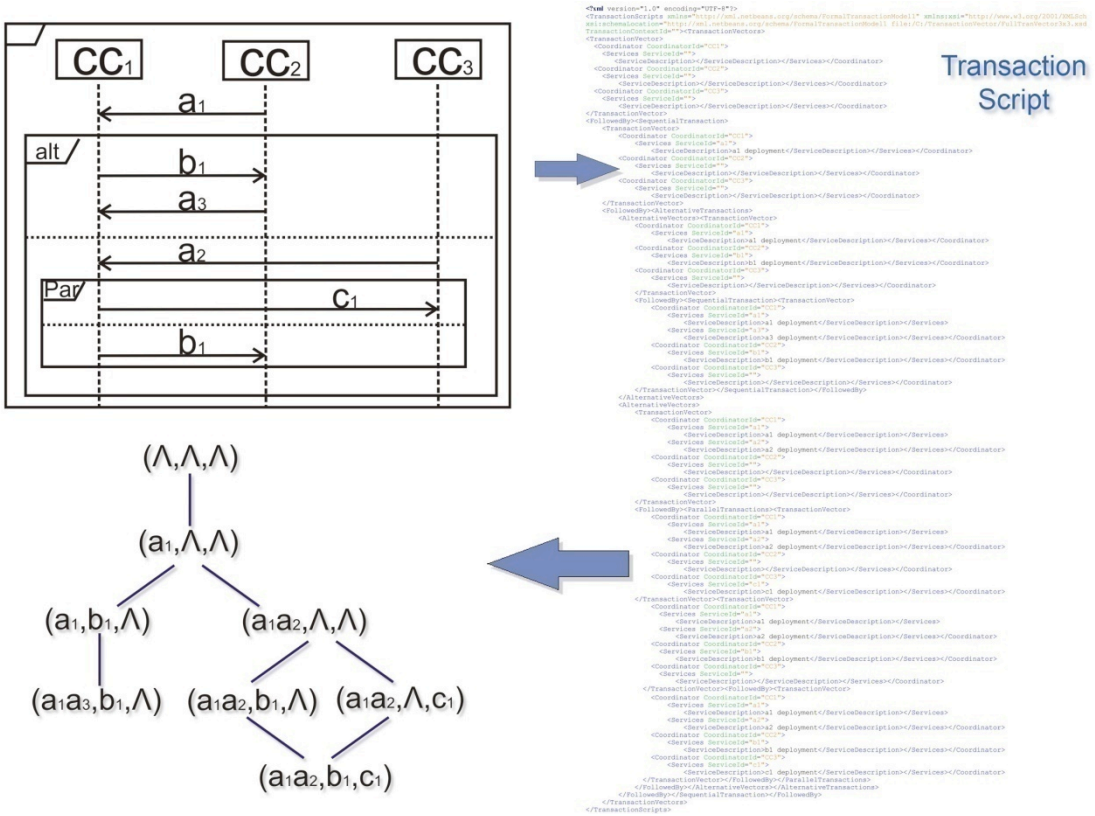


FIG. 2 TRANSACTION SCRIPT

The idea is that the particular subset of transaction vectors, for a given transaction, expresses the ordering constraints necessary in the corresponding orchestration of the underlying services. For instance, in the transaction of Fig. 4, following action a1, there is a choice between b1 on the coordinator component CC2 and a2 on CC1. Whenever b1 happens, it is followed by a3 on component CC1 whereas a2 is followed by c1 and b1 which happen concurrently on CC3 and CC2, respectively. Notice that the order structure in case of concurrent actions (as in b1 and c1 here) exhibits the characteristic structure of a finite lattice. This means that after the behaviour described by vector $\underline{u}$ = (a1a2, Λ, Λ) both independent action vectors α1 and α2 take place, resulting in the behaviour described by vector v = (a1a2, b1, c1), which is their least upper bound.

The additional structure provided by the notion of independence allows to go round the lozenge once and always end up with the behaviour in which both actions happened concurrently (this can be seen in [7]. As we will see in the next subsection, this also applies to going backwards and allows to compensate concurrently for concurrent forward actions.

The order structure of the transaction language determines the pattern the underlying service interactions between the coordinator components in the transaction should follow. Starting with the empty vector, and following the pattern of Fig. 3, each subsequent action (or concurrent actions) take place in going forward until the transaction as a whole terminates successfully. In [7], we have provided a schema for deriving the XML description of the order structure of a transaction language. These so-called transaction scripts describe the interactions between different coordinator components (recall Fig. 2) and determine the order in which the underlying services need to be deployed. Fig. 4 shows the transaction script generated from the vector-based behavioural description of the transaction in our approach

Transaction scripts reflect the corresponding transaction languages and hence describe the dependencies between services of different participants' coordinator components, in terms of the orderings of the underlying service invocations. This means that when the scripts are parsed they provide the full transaction history, resulting from the actual deployment of the transaction, but based on the associated formal semantics of the transaction. The XML schemas and further details on transaction scripts can be found (downloaded) following [9].

2.3 Execution History

So far we have described the use of transaction languages in expressing the forward behaviour of a transaction in terms of the orderings of the underlying service executions, and these can be analysed prior to deployment in determining the set of allowed sequences of actions. In our approach, the transaction history is captured in the order structure of the corresponding transaction language which is used to express forward behaviour and is reflected in the derived transaction scripts. As shown in Fig. 4 a transaction language which includes alternative actions will have different allowed *execution paths*. These start from the empty vector and lead to (one of) the largest vectors in the language. The largest vectors describe maximal behaviour of the transaction, in the sense that they do not describe an earlier part of behaviour than any other vector does. In the transaction language of Fig. 4 there are two maximal vectors, namely $\underline{v_1}$ = (a1a3, b1, Λ) and $\underline{v_2}$ = (a1a2, b1, c1). This means that one allowed sequence of execution is a1 → b1 → a3 and the other is a1 → a2 → (b1 concurrently with c1). Note that both allowed sequences end up in a maximal vector which corresponds to the behaviour exhibited when the transaction executes successfully until it terminates. Thus, in both cases the transaction produces a consistent state. In other words, transaction consistency is attained by reaching a maximal vector in the transaction language.

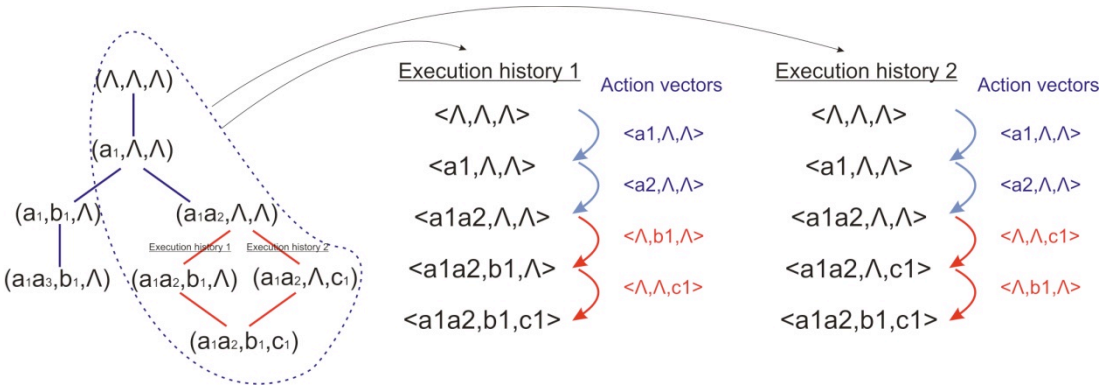


FIG. 3 EXECUTION HISTORY

When the transaction is actually deployed, then only one of the allowed sequences of actions can occur. In our example, this would depend on how the choice between a2 and b1, after a1 has occurred, is resolved. Fig. 5 shows the case where a2 occurred after a1, and hence the right branch of the Hasse diagram with the transaction language was actually deployed. It can be seen that each vector in turn is obtained by coordinate-wise concatenation with the appropriate action vector, according to the allowed sequence of actions along the particular execution path.

This distinction between the set of all allowed sequences of actions and the allowed sequence of actions that actually takes place when the transaction is deployed is important when considering compensations. Naturally, when a failure makes further progress impossible, we would only want to compensate for actions that actually happened up to that point and not for every possible action in the transaction. We will refer to the actual path of execution during a run of the transaction as the *execution history* of the particular deployment of the transaction. In Fig. 5 it appears there are two execution histories but these are the result of concurrency (between b1 and c1) and therefore, they are equivalent. This is because the series of concatenations with action vectors differ only in the order of the independent action vectors α1 = (Λ, b1, Λ) and α2 = (Λ, Λ, c1), and consequently they correspond to the same allowed sequence of actions.

Hence, the notion of independence is what allows us to identify equivalent behaviours in the presence of concurrency, something that is not possible when adopting an interleaving model, as done in [11]. The benefit of being able to identify equivalent execution histories can perhaps be seen most clearly when it comes to compensation in transaction recovery, where as we will see it is only required to compensate once for all equivalent execution histories.

2.4 Coordination and Transaction Failure

In this part we extend our approach to coordinating long-running transactions to handle compensations. This is to address occasions where some failure happens mid-way through execution in which case we need to have a way to express compensating sequences of actions that need to take place in going '*backwards*' while effectively undoing all previously successful forward actions the operation per coordinator is called *right-cancellation,* which

it is equivalent to call a cancelation for a service [formal definition of this can be found in [8] and [12]]. Fig. 6 shows the case where a failure occurs after service a2 of component CC2 has been invoked. This means that it is no longer possible for the transaction to produce a consistent state by reaching the maximal vector (a1a2, b1, c1), as dictated by the allowed sequence of actions in its execution history. Consequently, the transaction needs to be recovered and this implies returning to the (consistent) state the system was in before the transaction started.

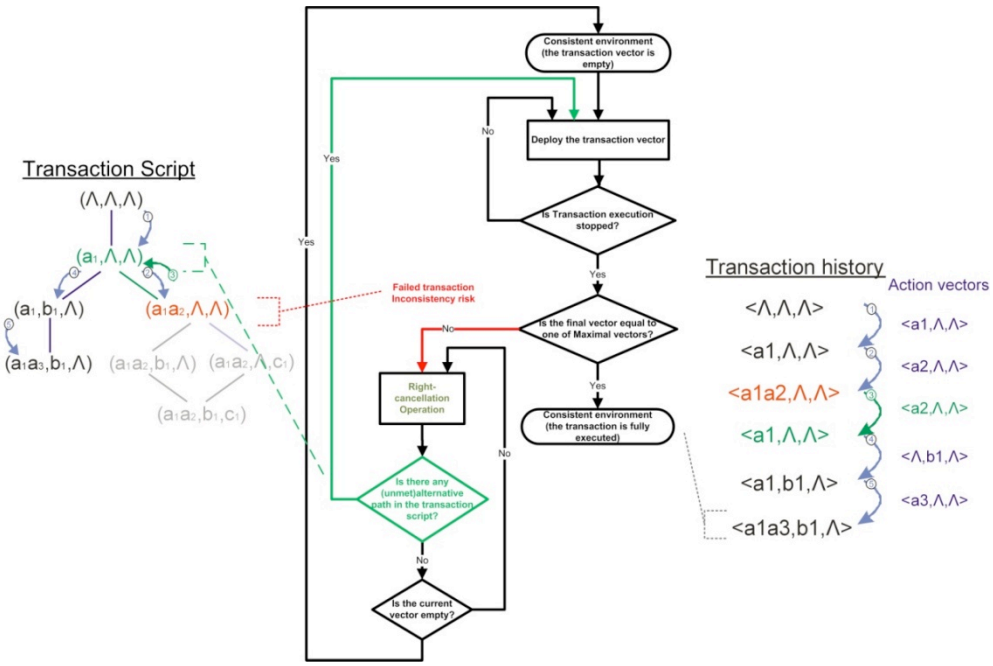


FIG. 4 TRANSACTION LIFE-CYCLE

We have seen how right-cancellation can be applied to generate compensating sequences of actions for the full recovery of a transaction. Full recovery however, can be costly in terms of resources, delays, business relations and so on. Additionally, in a highly transactional environment dependencies may also exist across transactions so the effect of a recovered transaction may be magnified. For this reason it is desirable to avoid full recovery wherever possible. One way to do this is to design transactions with a number of alternative scenarios of execution; in other words, allow for multiple execution histories in the corresponding transaction script. In such cases, our approach comes with the provision for forward recovery which is a mechanism for avoiding full recovery. The aim is during compensation to explore whether there is any possibility for successfully terminating the transaction following a different execution history to the one originally deployed, instead of compensating for the whole execution history.

This is possible in our approach because all the execution histories are captured in the

transaction language. Hence, in recovering a given execution history which failed, after the next forward action(s) is compensated for we check whether the resulting vector is part of a different execution history. If this is the case, then we attempt to go forward by performing the execution in accordance with the allowed sequence of actions in that execution history. As shown in green box, Fig. 6, in going backwards, and while cancelling out one action (or a set of concurrent actions) at a time, we look each time whether there is an alternative path from the vector we arrived on (after applying the compensating action(s)) leading to a maximal vector.

2.5 REFERENCES

[1] S. Moschoyiannis, A. Razavi, and P. Krause, "Transaction Scripts: Making Implicit Scenarios Explicit," *Electronic Notes in Theoretical Computer Science*, vol. 238, Jun. 2010, pp. 63-79.

[2] M.P. Papazoglou, "Service-Oriented Computing: Concepts, Characteristics and Directions," *Proceedings of the Fourth International Conference on Web Information Systems Engineering*, Washington: IEEE Computer Society Press, December, 2003.

[3] M.P. Papazoglou and D. Georgakopoulos, "Service-Oriented Computing," *Communications of the ACM*, vol. 46, 2003, pp. 25-28.

[4] M.P. Papazoglou, A. Delis, M. Haghjoo, and A. Bouguettaya, "Language support for long-lived concurrent activities," *Distributed Computing Systems, 1996., Proceedings of the 16th International Conference on*, 1996, pp. 698-705.

[5] M.S. Haghjoo and M.P. Papazoglou, "TrActorS: a transactional actor system for distributed queryprocessing," *Distributed Computing Systems, 1992., Proceedings of the 12th International Conference on*, 1992, pp. 682-689.

[6] A. Razavi, A. Marinos, S. Moschoyiannis, and P. Krause, "Recovery management in RESTful interactions," *Digital Ecosystems and Technologies, 2009. DEST '09. 3rd IEEE International Conference on*, 2009, pp. 419-424.

[7] S. Moschoyiannis, A. Razavi, Yongyan Zheng, and P. Krause, "Long-running Transactions: Semantics, schemas, implementation," *2nd IEEE International Conference on Digital Ecosystems and Technologies, 2008. DEST 2008.*, IEEE Computer Society, 2008, pp. 20-27.

[8] A. Razavi, P. Krause, and S. Moschoyiannis, "Digital Ecosystems: Challenges and Proposed Solutions," *Handbook of Research on P2P and Grid Systems for Service-Oriented Computing*, IGI Global, 2010.

[9] A. Razavi, S. Moschoyiannis, and P. Krause, "An open digital environment to support business ecosystems," *Peer-to-Peer Networking and Applications, Springer New York*, vol. 2, Dec. 2009, pp. 367-397.

[10] M.W. Shields, *Semantics of parallelism*, Springer New York, 1997.

[11] M. Butler, T. Hoare, and C. Ferreira, "A trace semantics for long-running transactions," *Communicating sequential processes: the first 25 years: Symposium on the Occasion of 25 Years of CSP, London, UK, July 7-8, 2004: revised invited papers*, Springer-Verlag New York Inc, 2005, pp. 133–150.

[12] A. Razavi, S. Moschoyiannis, and P. Krause, "A Coordination Model for Distributed Transactions in Digital Business EcoSystems," *Digital EcoSystems and Technologies Conference, 2007. DEST '07. Inaugural IEEE-IES*, IEEE Computer Society, 2007, pp. 159-164.

3 INTRODUCTION TO FLYPEER INTEGRATION FRAMEWORK (IPTI)

3.1 REMARKS:

1 Identity Provider
    limited to 1 for network, for now
    really do the authentication
    permits the development of as many authentication types as need
    the peer which has the identity provider up have to be 'online'
2 Authenticator
    Interface for the authentication process
    each authentication type needs an interface implementation for it
    calls the Identity Provider
    after the authentication successful receives a valid credential
    the credential is passed through all peers participating of the transaction
    the credential passed is the initiator's credential
    identify the peer
    All services in the transaction receives a remote credential
    initiator's credential is recreated in the peers that are participating of the transaciton. It's the remote credential.
    If identity is not necessary in the service just ignore the credential
    if a credential is passed the transaction will run only if the credential is valid

3.2 IDENTITY FLOW IN FLYPEER

Identity flow, a project lead by WIT, was fully integrated to Flypeer to give it support to authentication and identification. The Identity Flow supports new identities solutions into the network with no centralized authority.

3.3 IDENTITY PROVIDER

To support authentication, the network must have at least one peer with Identity Provider set up. The Identity Provider is the component from Identity Flow which does the authentication process and generates a credential to the peer when the authentication process is successfully completed.

For now, there is a technical limitation that could not be addressed due to time constraints. We can have only one identity provider running in the Flypeer network at a time. To get a peer with the Identity Provider running, the peer must be started with the peername *'default'.* After that, any peer can be executed, be authenticated in the network and receive credentials.

3.4 AUTHENTICATING

A peer to be authenticated and receive a valid credential it must choose the authentication type. For now, Flypeer supports two types of authentication: Default and Guigoh's authentication.

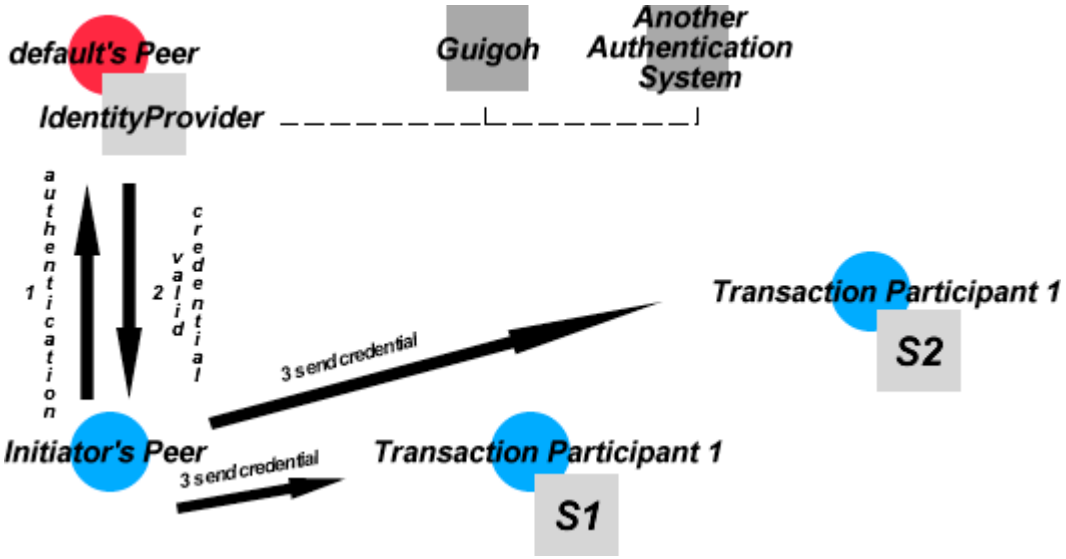
If the Default authentication is chosen, a very simple authentication will be executed and no valid credential will be obtained. Because of that, any service which calls for a valid credential will not accept anything from this peer, then any transaction will not can be executed completely.

Otherwise, if the Guigoh's authentication is chosen the peer will receive a valid credential and any type of transaction and service can be reached. But, a peer to be authenticated by Guigoh must have a Guigoh's account (http://www.opaals.org.br).

Leveraging both Flypeer and IdentityFlow, it is also possible to create new authentication methods besides the ones mentioned here, which gives some extra flexibility.

3.5 CREDENTIAL

Once a peer is successfully authenticated, it receives a credential from the Identty Provider and then when any transaction is started the credential is sent to all peers into the transaction.
The Credential can be accessed by the *getInitiatorCredential* method of the *ServiceContext*'s object received as a parameter in the *serviceEvent* method of the implemented service. In figure X, the diagram represents these classes.



In summary, what happens is that, since each participant in the transaction has access to the transaction's initiator's credential, it can then validate if the messages they are receiving for each transaction come from the correct peer, instead of a fake one, for example.

Besides transaction validation, there is at least one more use for credentials. They can be used by the services to implement user related things, like file storage with files belonging to

the correct users. It is important to note that the trust of this behaviour has not being fully investigated though.

4 INFRASTRUCTURE INTEGRATION, DISTRIBUTED IDENTITY, TRUST AND ACCOUNTABILITY (WIT IPTI)

4.1 IDENTITY MODEL AND IMPLEMENTATION

The final identity model is described in D3.9. The identity model implementation, developed as part of the IdentityFlow (reference IdentityFlow & demo) sourceforge.net project, is described in D3.11.

IdentityFlow can be found at http://sourceforge.net/projects/identityflow/, while Flypeer can be found at http://kenai.com/projects/flypeer.

4.2 INTEGRATION POINTS BETWEEN IDENTITY WORK AND FLYPEER

There are a number of points at which the infrastructure, i.e. Flypeer, requires access to identity and identity related functionality. On OPAALS, identity work has encompassed a large breadth of issues including, theoretical user identity, naming and naming schemes, authentication, (single) sign-on, authorisation, digital credentials and PKI schemes. The core of WIT's contribution is IdentityFlow, the OPAALS identity model implementation, which is a platform agnostic toolkit for identity claim and verification, facilitating sign-on and attribute claim verification. Many of the areas listed above are not addressed directly by IdentityFlow, since work on these issues does not further WIT's contribution towards advancing the state of the art, however, in order to integrate the innovative identity work into Flypeer so as to produce useful and workable identity on the platform, much additional work was required.

The following integration points were identified and worked on,

1. A username and password (and other data), captured by Flypeer, can be sent to some Identity Provider (IdP) for authentication. Authentication can be triggered from within Flypeer (usually when the node starts) and the authn data is sent to appropriate authentication modules that have been provided as an add-on to IdentityFlow.

FlypeerMain.java (snippet)

```
AuthnData authnData = new AuthnData(authnType, username, password);
new SimplePeer(authnData, new Notifier());
```

PeerStarter.java (snippet)

```
PeerGroup defaultFlypeerGroup = new PeerGroupHandler().connectTo(
          PeerGroupHandler.FLYPEER_DEFAULT_PEER_GROUP, peer, 5000);
```

PeerGroupHandler.java (snippet)

```
peer.getAuthenticator().credentialAuthenticate(
                peerGroupFound, peer.getAuthnData());
```

2. Once a user has been authenticated, a Credential is created that can be used throughout the Flypeer platform to verify an authenticated identity. This Credential object is a JXTA Credential, which is valid for JXTA-based platforms, of which Flypeer is one. The Credential is a custom implementation that contains additional encryption and IdP metadata. (See Credential XML Snippet below.)

```
(The snippet below is called by the credentialAuthenticate() method
in above snippet.)
```

Authenticator.java (snippet)

```
LocalCoordinatorCredential credential =
(LocalCoordinatorCredential) membershipService.join(authenticator);
```

3. Flypeer must present Credentials when making secure connections to other peers so that the destination peer can authenticate the incoming connection immediately.

```
Credentials are sent as payload in incoming connections that must
be authenticated. Credentials are serialized into XML for
transmission, and reconstructed remotely, using, respectively,

CredentialUtil.encodeCredential(Credential credential)

CredentialUtil.decodeCredential(String credentialEncrypted,
PeerGroup peerGroup)

(The validity of Credentials can be established as outlined in the
next section.)
```

4. An identity operation (see D3.9/3.11) must be launched by services to verify identity claims made by peers (that may or may not be authenticated at this stage). Peers must be able to determine whether the identity of other peers (using a different IdP) can be trusted. The following code snippet can be used by services to launch and identity operation.

SampleOperationLaunchTest.java (snippet)

```
/* Get the initial Connection. */

AssertionVerificationOperation operation =
AssertionVerificationOperation.assertionVerificationOperationInstan
ce();

JXTAConnection initialConnection =
(JXTAConnection)operation.getInitiatingConnection();

/* Local SAMLRequest into initialConnection dispatcher. */

JXTADispatcher dispatcher = initialConnection.getDispatcher();

dispatcher.setDestinationPeer(destinationPeer);

dispatcher.setPeerGroup(commsPeerGroup);

dispatcher.setChannel(Configuration.DEFAULT_OUTPUT_CHANNEL);
```

```
encoder.createAuthnRequest();
SAMLRequestParameterMap.LoaderMap samlRequest =
SAMLRequestParameterMap.loaderMapInstance(encoder.getAuthnRequest()
);
dispatcher.setAttributes(new SAMLRequestParameterMap(samlRequest));

/* Dispatch connection. */
initialConnection.dispatch();
```

4.3 Interactions between IdentityFlow and Flypeer

There are two main steps for verifying identity claims on the OPAALS DE. These are outlined below.

Step 1: Identifier Claims From Authenticating IdPs

JXTA provides a MembershipService framework for authenticating to PeerGroups, which IdentityFlow implements as an extension to its core functionality. This allows Flypeer to authenticate users to the main Flypeer group. The IdPMembershipService, which IdentityFlow implements, is available to each node on the JXTA network and operates as follows,

1. A set of login parameters, and the JXTA advertisement of a preferred IdP, is supplied via Flypeer as a JXTA Authenticator on the authenticating node.
2. The IdPMembershipService contacts the IdP node and supplies it with the correct authentication parameters.
3. The IdP returns an authentication result to the authenticating node, allowing the IdPMembershipService to complete it's PeerGroup 'join' attempt.
4. If authentication has been successful, the IdPMembershipService generates a credential from the result passed back from the IdP, which acts as proof that the entity's identity is indeed asserted by the IdP.
5. This credential can then be used as a means of filtering access to services in a JXTA-based environment.

The scheme for generating and using the credential assumes that all IdPs have a public-private key pair, and is as follows,

1. Following a successful authentication attempt, an IdP will sign the 'claimed identifier'6 with its private key.

2. The identifier, the IdP's public key, the encryption method and the signed identifier are packaged with other metadata into a JXTA Credential object.

3. Other entities can verify that the IdP in question authenticated the authenticating entity by obtaining the Credential ob ject and using the IdP's public key to decrypt the signed identifier and comparing it with the claimed identifier.

The Credential implementation used by IdentityFlow marshals and unmarshals to and from XML. An edited (for space) example of a marshaled Credential is given below.

```
<?xml version="1.0"?>

<!DOCTYPE jxta:Cred>

<jxta:Cred type="jxta:LocalCoordinatorCred" xml:space="..."
xmlns:jxta="http://jxta.org">

<PeerGroupID>urn:jxta:uuid-FEF..102<PeerGroupID>

<PeerID>urn:jxta:uuid-596..403<PeerID>

<SignedPeerID>MCwpf4i...K3O2oEPw=</SignedPeerID>

<SignAlgorithm>DSA</SignAlgorithm>

<GroupPublicKey>MICC...MYboJk=</GroupPublicKey>

<Username>identityflow@Guigoh</Username>

</jxta:Cred>
```

Step 2: Verifying an Identifier Claim

Identity claims are verified using identity operations, as discussed in sections 6 and 7. The identity operation specifies7

1. A set of contingent ordered connections that are executed in sequence (from one actor to the next), potentially with conditional logic affecting the protocol flow.
2. A set of actor tasks for each possible state of each actor in the protocol flow, that are executed when an actor assumes a particular state.
3. A binding, or set of bindings, that specify connection transport functionality in particular environments (e.g. JXTA binding).

A set of trust checks that must be made at appropriate points during protocol execution to ensure that there is sufficient trust between actors for the operation to succeed.

Identity operations are typically triggered by resource access, where it is necessary for entity identity claims to be accepted by the SP before the accessing entity can be granted access. Claims concerning identifiers issued by IdPs are important examples of claims that must be verified. In the particular case of the OPAALS DE, operations that verify user identifiers are simplified by the fact that each user propagates a credential which effectively encodes an identifier claim made by the user's IdP on behalf of the user. Therefore, only connections 1, 8 (service request and response) and 2, 7 (identifier claim verification request and response) in Figs. 5 and 6 need to be executed.

For all other identity claims, pertaining to any property that an entity can claim to possess and that its IdP can verify, identity operations are necessary.

Examples of other claims are "I am over the age of 18" or "I am a member of the OPAALS consortium" or "My real name is John Murphy". Whether or not these claims are accepted by other entities will depend on whether the accepting party's IdP trusts the claiming party's IdP in the context of asserting identity claims.

Since the OPAALS DE is a JXTA-based environment, a JXTA binding is used, which means

that all operation connection messages (by current convention) are passed via JXTA pipes. Using JXTA transport functionality ensures entities operate in a pure P2P overlay, where entities can be contacted by name without the need for knowing a peer's underlying transport address (i.e. IP address); and also provides transparent firewall traversal.

Identity operations will tend to be designed to address a particular need, such as identifier verification, and will tend to be triggered by an actor seeking to verify a claim during the course of some activity. Therefore, it is anticipated the development of identity operations will be in response to the needs of SPs. However, it is likely that the template of the simple operation given in Fig. 5 will suffice for most purposes. An example of a more complicated identity claim would be a situation where an SP would not accept a particular claim unless two or more IdPs asserted the claim on behalf of the user. This would necessitate a more complicated protocol flow involving 5 rather than 4 actors. In general, if any of the items in the list above change, modification will be required, which should be simplified by the modular, extensible design chosen by IdentityFlow.

4.4 Trust Model and Implementation

The final trust model is described in D3.9. The implementation, developed as part of the TrustFlow[1] sourceforge.net project, is described in D3.11.

The model is shown below in Figure 1. The approach is to use a trust overlay network for providing a community based approach to trustworthiness based on the reputation of entities. The Entity can represent a node, service, resource, a service provider or a service consumer. Each entity has a Trust Manager associated with it. The entities gain experience from interacting with other entities and publish reports of these experiences to the Trust Manager. Using a pre-defined context dependent algorithm, the Trust Manager updates the entities' local trust and based on a policy of sharing trust information provides trust updates to other Trust Managers in the overlay network.

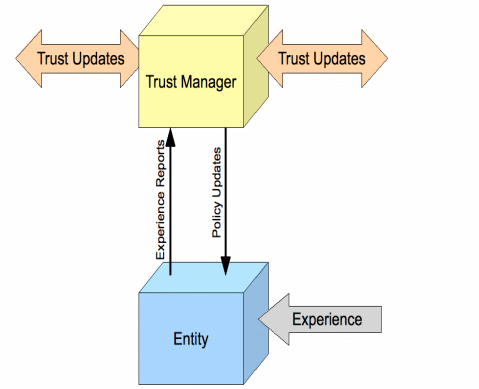


FIGURE 1: OPAALS TRUST OVERLAY MODEL

[1] TrustFlow project, a set of Java components to enable a referral based trust and

4.5 ReferralHandler

The transfer of referrals is defined through the use of a `ReferralHandler` interface (shown in Figure 2). The interface defines two methods, `getReferral()` which is used request remote referrals via a `ReferralQuery()` and `putReferral()` which is used to make available or publish a `TrustValue` as a referral. This interface is then implemented for each platform that TrustFlow supports.

```
package org.opaals.trust;
public interface ReferralHandler {
        TrustValue getReferral(ReferralQuery query);
        void putReferral(String trustor, TrustValue tv);
}
```

FIGURE 2: REFERRALHANDLER INTERFACE

4.6 JXTA Implementation

In the case of FlyPeer, which is a JXTA based platform, an implementation called `JxtaReferralHandler` using the JXTA Resolver API was developed. The `QueryHandler` interface provides a means in JXTA to query and receive messages via a query response messaging paradigm. `QueryHandler` defines two methods for this, `processQuery()` for processing received queries and `processResponse()` for processing responses to queries previously issued. The `ResolverQueryMsg` and `ResolverResponseMsg` classes allow for associating responses with queries. See Figure 3 for details.

---

reputation system, http://sourceforge.net/projects/trustflow/

```
package org.opaals.trust.jxta.impl;
public class JxtaReferralHandler implements ReferralHandler, QueryHandler {
        public TrustValue getReferral(ReferralQuery referralQuery) {
        // implement creating a query, send it and return response if available
        }
        public void putReferral(String trustor, TrustValue tv) {
        // implement making a TrustValue available for transferral
        }
        public int processQuery(ResolverQueryMsg query) {
        // receive a query and process it
        }
        public void processResponse(ResolverResponseMsg msg) {
        // receive a response to a query and transfer to TrustValue
        }
}
```

FIGURE 3: JXTAREFERRALHANDLER CLASS

In addition to implementing the methods of the `ReferralHandler` and `QueryHandler` the `JxtaReferralHandler` class provides a constructor which takes a JXTA `PeerGroup` object as a parameter in order to associate the queries with a `ResolverService` for that `PeerGroup`.

4.7 INTEGRATING TRUSTFLOW WITH FLYPEER

As described in D3.11, the `TrustManager` is the core of TrustFlow and has the following responsibilities:

1. Maintaining a set of algorithms for connected `Trustor` entities for specific contexts. Each context can have distinct algorithms for determining trust based on direct experience, referrals and reputation.
2. Receiving `ExperienceReports` from the `Trustor` entities and generating updated trust values according to the appropriate algorithms.
3. Providing updated `TrustValues` to other `TrustManagers` which can be used as inputs to referral or reputation algorithms. It is only appropriate that these published `TrustValues` have been derived from direct experience.
4. Providing `PolicyUpdates` to the `Trustor` entities based on current `TrustValues` derived from direct experience and referrals and reputation.

In order to integrate the facilities of TrustFlow into FlyPeer it is required that FlyPeer services and service consumers can access a `TrustManager` object and query it for trust values as well as publish trust values for referrals. This is initially modelled by adding two methods to the base `Peer` interface in FlyPeer to get and set the `TrustManager` object. This is shown in Figure 4.

```
package br.org.ipti.flypeer.model;
public interface Peer {
        ...
        TrustManager getTrustManager();
        void setTrustManager(TrustManager trustManager);
}
```

FIGURE 4: TRUSTFLOW ADDITIONS TO THE PEER INTERFACE

These two methods are then implemented in in the abstract class `AbstractPeer` which maintains a private member object instance of `TrustManager`.

Each FlyPeer node is instantiated in a Thread through a class called `PeerStarter`. In this method the `TrustManager` is initialised using the `JxtaReferralHandler` and the `setTrustManager()` method is called on the Peer object. The initialisation of the `TrustManager` is shown below in Figure 5.

```
package br.org.ipti.flypeer.main;
public class PeerStarter implements Runnable{
        ...
        private TrustManager initTrustManager(PeerGroup peerGroup){
                JxtaReferralHandler handler = new JxtaReferralHandler(peerGroup);
                TrustManager trustManager = new SimpleTrustManager();
                trustManager.setReferralHandler(handler);
                AlgorithmConfigurator algoConfig = new
                        AlgorithmConfigurator(peer.getUsername(),
                        trustManager,
                        "algoConfigurationFile.xml");
                return trustManager;
    }
}
```

FIGURE 5: SNIPPET FROM PEERSTARTER SHOWING TRUSTMANAGER INITIALISATION

Once this initialisation is complete a service or service consumer can retrieve the TrustManager and assess trust ratings as described in deliverable D3.11.

4.8 ACCOUNTABILITY

The Accountability model is described in deliverable D3.8 and the interfaces for implementation is described in D3.11. The model was published in [Mal09], where it won "Best Paper Award".

While a simulation of the model using the interfaces defined in D3.11 has been performed successfully in-house, a full blown implementation integrated with FlyPeer was not performed due to a lack of resources available within the lifetime of the project.

REFERENCE:

[Mal09] Malone, P., Jennings, B., *Distributed support for public and private accountability in digital ecosystems,* in proceedings of MEDES '09: Proceedings of the International Conference on Management of Emergent Digital EcoSystems, pp 391-398, 2009, ACM, available at http://doi.acm.org/10.1145/1643823.1643895

5 RESTful approach; conventional transaction and recoverability (Surrey)

Within the EU-FP6 OPAALS project [6] and Digital Ecosystems [7, 13] research in general, there is a need for a transaction model to help disparate service providers in a distributed system coordinate and compose into unforeseen, higher value services, forming an evolving digital ecosystem for business and community activities. In this, we have found REST an ideal complement, as the following quote indicates:

"A lot of people think of systems as static things. Dead things. REST is not going to appeal to those people. All of its constraints are designed to keep systems living longer than we are willing or able to anticipate." – Roy Fielding [5]

We therefore have started research efforts in the direction of enabling RESTful transactions to take place over the web. However 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.

5.1 Why do we need RESTful Transactions?

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 the moment. 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 end up having adverse effects contrary to user expectations in some cases.

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 upcoming Sun Cloud API [1, 2]. 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 [3, 4].

5.2 Transaction Basics

The general term 'Transaction' has been introduced by Gray [8] 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 [12]. 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.

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.

Table 1 - Available Operations for Tc

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 2 as well as the operations that can be applied to them.

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 2 - Elements of 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

Table 3 - Available Operations for T

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 [9] 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 [2, 10, 11]. 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.

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

Table 4 - Available Operations for T-Lc

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 1 – (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 5.

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.

Table 5 - Elements of T-L

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 6 shows the lock compatibility rules.

	Mode of Preceding Lock		
Mode Of New Lock		*Share*	*Exclusive*
	Share	*Yes*	*No*
	Exclusive	*No*	*No*

Table 6 – Legal lock sequences

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 [8] protocol.

GET	Returns the state and links of the lock.

Table 7 - Available Operations for T-L

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 6, 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.

GET	Returns the resource's collection of locks.

Table 8 - Available Operations for R-Lc

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.

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

Table 9 - Available Operations for L-IR

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.

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.

Table 10 - Available Operations for L-CR

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 2 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.

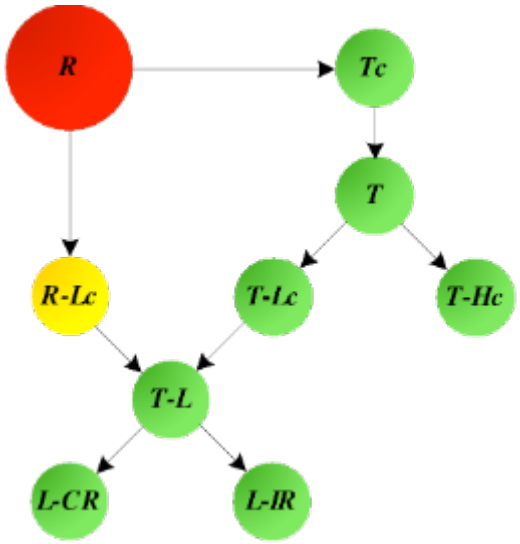


Figure 2 – Resource Hypermedia connections

Tables 11 and 12 summarize all the relevant resource types that comprise our model together with a short description and a list of the allowed operations.

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	The potential representation of a locked resource,

Representation (L-CR)	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 11 – Transaction model resource types

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]

Table 12 - Allowed Operations on the Resources

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 3 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.

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

Figure 3 – Creating a transaction

5.6.2 Example 1.1 Aborting the Transaction

Figure 4 shows how the example in figure 3 could be continued if the client decides to abort the transaction.

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

Figure 4 – Aborting a transaction

5.6.3 Example 1.2 Committing the Transaction

Figure 5 shows how the example in figure 3 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.

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

Figure 5 – Committing a transaction

5.6.4 EXAMPLE 2: CONCURRENT TRANSACTIONS

The example in figure 6 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.

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

Figure 6 – Concurrent transactions

5.6.5 EXAMPLE 3: MULTI-SERVICE TRANSACTION

The example in figure 7 shows a transaction over multiple services. In fact, it is an application of the 2 Phase Commit protocol [8]. 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.

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

Figure 7 – Multi-service transactions

5.7 YEAH, BUT IS IT RESTFUL?

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 2 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.

In case you missed the link at the top, You can give us your feedback about RETRO here.

5.9 REFERENCES

1. Google, "Protocol Reference - Google Data APIs - Google Code." URL: http://code.google.com/apis/gdata/docs/2.0/reference.html Accessed: 31/3/2009
2. Sun Microsystems, "The APIs for the Sun Cloud — Project Kenai." URL: http://kenai.com/projects/suncloudapis Accessed: 31/3/2009

3. L.F. Cabrera, G. Copeland, M. Feingold, R.W. Freund, T. Freund, J. Johnson, S. Joyce, C. Kaler, J. Klein, D. Langworthy, M. Little, A. Nadalin, E. Newcomer, D. Orchard, I. Robinson, J. Shewchuk, and T. Storey, "Web Services Coordination (WS-Coordination)," Aug. 2005
4. P. Furnis and A. Green. Choreology Ltd. Contribution to the OASIS WS-TX Technical Committee relating to WS-Coordination, WSAtomicTransaction and WS-BusinessActivity. November 2005
5. Rest-Discuss http://tech.groups.yahoo.com/group/rest-discuss/message/12170?var=1
6. "OPAALS Project: Open Philosophies of Associative Autopoietic Digital Ecosystems" URL: http://www.OPAALS.org - Accessed: 31/3/2009
7. F. Nachira, A. Nicolai, P. Dini, M. Le Louarn, and L. Rivera Leon, Eds., Digital Business Ecosystems. European Commission, 2007.
8. Gray, J. & Reuter, A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1993
9. D. Recordon and D. Reed, "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.
10. T. Bray, "ongoing · The Sun Cloud.", URL: http://www.tbray.org/ongoing/When/200x/2009/03/16/Sun-Cloud Accessed: 31/3/2009
11. R. Fielding, "It is okay to use POST » Untangled." URL: http://roy.gbiv.com/untangled/2009/it-is-okay-to-use-post Accessed: 31/3/2009
12. Bernstein, P.A., Hadzilacos, V., and Goodman, N. Concurrency control and recovery in database systems. Addison-Wesley, Boston, MA, USA, 1987
13. "IEEE Conference on Digital Ecosystems and Technologies." URL: http://www.ieee-dest.curtin.edu.au/ Accessed: 31/3/2009

6 Towards the Web of Models: A Rule-driven RESTful Architecture for Distributed Systems (Surrey)

This chapter was presented at RuleML 2010.

6.1 Introduction

Since its inception, the Web has radically expanded its limits to include ever more people, and cover an ever 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 in their approach, intending to enable new uses and standardize widespread practices, without causing deep 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. In this paper we examine the competing visions and propose criteria for a way forward. Based on these criteria, we propose an architecture built on RESTful Web Services, Semantics of Business Vocabulary and Rules (SBVR) [1] as a modelling language, and proceed to show how a number of advanced use cases can be built upon this minimal foundation.

6.2 Visions for the Future Web

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

The WS-* web services efforts [2], also known as Big Web Services [3] 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 [4] 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. 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[6] and more recently the Linked Data[7] 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 is an approach that focuses on the principles behind the architecture of the Web. As such, its ambition may appear limited at first, compared to the other two visionary approaches. Its focus is on scalability, loose coupling, linkability and ease of maintenance [3]. It is a direct result of the work of Roy Fielding in developing HTTP, as described in his doctoral dissertation[5]. What sets REST apart is that its focus is not on novel technologies as much as on a principled use of those already available,

mainly HTTP, URI, and the various hypermedia formats. The principles are well-defined as a set of architectural constraints that are reflected in the operation of these standards. These are: Naming of resources, the uniform interface, statelessness and self-descriptive messages, manipulation of resources through representations, and use of hypermedia as the engine of application state. Each of these can and has been used on today's Web with great effect. The result is a large recent shift away from WS-* standards and towards RESTful Web Services on the part of web practitioners. Perhaps the most well known usage of REST as a machine-to-machine interaction paradigm is the increasing use of feeds (RSS, ATOM) not only for their native blog subscription scenarios, but as a generic mechanism to communicate updates between services. Recent advances include podcasting and real-time feeds.

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 addressing individual triples. As a result, HTTP is used to read collections of triples, and SPARQL [8] and the newer SPARQL Update [9] 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 even be described with WSDL as a web service. This brings us back to the service level of granularity in terms of linkable entities.

Table 1. 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,	Predicate (RDF Triple)	Data analysis, Automated	RDF, RDFS, OWL,

		Semantic annotation		reasoning	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

To judge any competing proposals for the future of the Web, a consistent set of criteria must be applied. The criteria that we have judged crucial for the viability o any approach are the following:

- *Compatibility*: It must be backwards compatible with today's Web to the greatest extent possible.
- *Simplicity*: It must place the least amount of overhead to adoption for developers and users alike.
- *Completeness*: It must allow the different communities to implement as many of their critical use cases as possible.

While any set of criteria contains an element of subjectivity, stating them explicitly allows the reader to decide for themselves whether they agree with our priorities. Judging from the state of the mainstream web, and the uptake of the alternatives, any solution for the future of the Web has to take these two primitive concepts into account. This does not mean that we should abandon the vision of a machine-to-machine Web, or that we should stop working for service composition and other 'enterprise-grade' requirements to be taken seriously. This paper presents the initial steps towards a REST-

based architecture for the web that can enable use cases that the Semantic Web and WS-* communities require. In many cases, concepts and even technologies from all three approaches can be used together under the architectural principles of REST to reach these goals.

6.3 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."* [5]

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 solution we propose 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 if 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 designers of the client. 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, which they are not. A core difference of the two is in the expressiveness of the meta-model, and thus XML is an option where JSON is not suitable due to a more expressive meta-model, despite its higher bandwidth overhead.

The ATOM and ATOM Publishing Protocol [10] specifications define three new media types, even though they are all serialised as XML. This implies 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 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. We have identified four types of information that the media type can convey: These are syntax, meta-model, schema, and semantics.

In a previous publication [11] the potential of an SBVR-described media type was first discussed with the example of a student record resource given. Here, we revisit this example in the light of the four types of information that a media type. 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. 6. 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 2. Instance of an SBVR model subset describing a single resource.

Terms	Fact Types	Rules
Student	**Student** *is under probation*	It is necessary that each **student** *is registered for* at most five **courses**.
Module	**Student** *is registered for* **course**	It is necessary that each **module** that a **student** *is registered for is available for* a **course** that the **student** *is enrolled in*.
Course	**Student** *is enrolled in* **module**	It is necessary that each **student** that *is*

Grade A or B or C or D or F First name Last name	Student *has* first name Student *has* last name Student *is marked with* grade *for* course Module *is available for* course	*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 [12] 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, or fragments of an overall model, 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 natural language form. This is a use case that derives from SBVR's unique advantages and is therefore hard for other architectures to match. 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 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, 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.

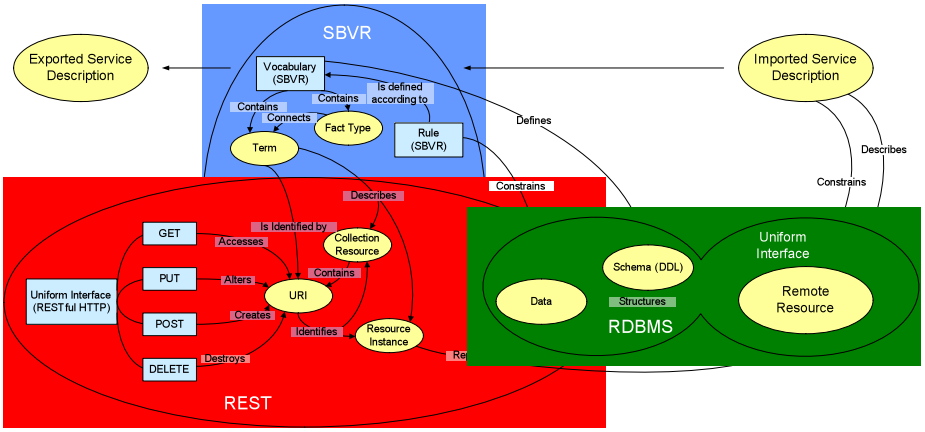


Fig. 7.

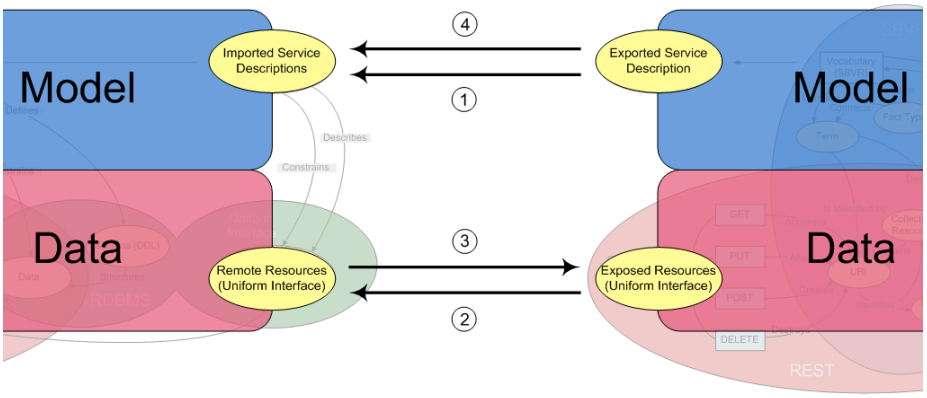
6.4 Model Propagation

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 Information Systems (GIS), which use SBVR as their internal modelling language as well as external resource description language. It is important to note that while GIS are the canonical example, in the spirit of backward compatibility, anything that can be done in a GIS can be done in a conventional information system. The internal architecture of a GIS simply allows it to do so more efficiently, utilising the full strength of the new paradigm. Figure 2 portrays the architecture of a GIS, 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.

It is important to note that in figure 2 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).

Fig. 8.

The exact mechanism by which inconsistencies are noted in the behaviour of the resource is shown in figure 4. 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 on 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.

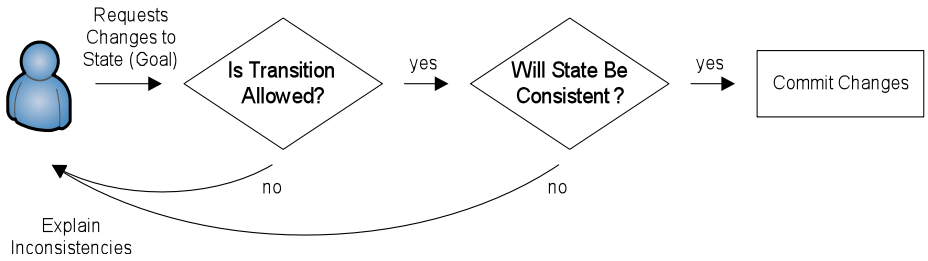


Fig. 9. The meta-process of interaction between a client and the internal logic of a service.

This of course only covers changes to the model that make it more restrictive, not more permissive. For instance, if a sixth course was suddenly allowed, the consuming service would not notice this with this mechanism. What this mechanism is intended to do is avoid 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.

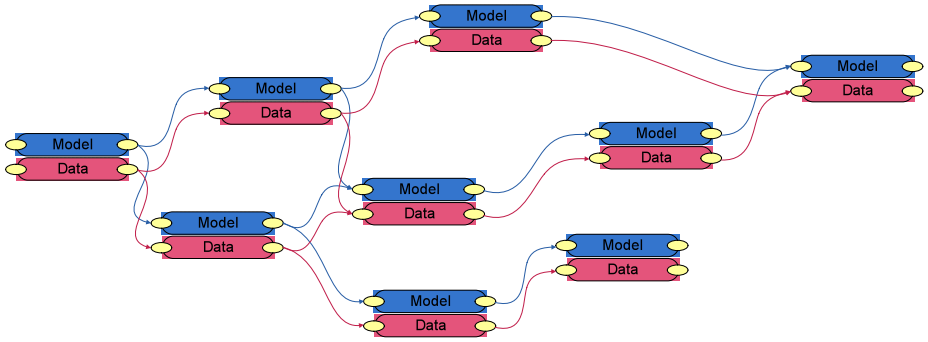


Fig. 10. The Web of Models overlaid on today's Web of Data. As data updates flow from clients to servers, updates in the models propagate in the reverse direction.

6.5 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 Information Systems operate in determining whether an update is allowed and whether it will leave the node in a consistent state. Inference however, can be thought of as a special case of service composition.

According to [13], service composition can 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. The

second is ‘Generating Multiple Effects’ in which a number of services should be executed to produce separate but interrelated outcomes. 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. 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 expose them through SBVR descriptions. 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 services themselves (such as credit card processing) the nesting can expand to a fourth level.

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 (figure 4) 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. In every case, the updates can then be propagated further until all relevant nodes in this 'web of models' (figure 5) 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, a transaction model is required, and since our architecture is RESTful the transaction model needs to operate within the Web Architecture's constraints also. To this end, the authors of this paper have developed RETRO [14], A RESTful Transaction Model, capable of atomic transactions across distributed HTTP systems.

This description of a service composition scenario is a smaller but updated version of previous work done by the authors on declarative service composition with SBVR [15].

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.

6.6 Conclusions & Future Work

In this paper we have brought together concepts and use cases from the three leading schools of thought regarding the future of the Web and constructed an outline of a novel step forward for Web Architecture paradigm. 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 implementing many of the aspects described in this paper. Also, having exposed the ontology problem at its core, there are approaches that can be applied to it, inspired by emergent folksonomies as seen in the area of social networks.

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 paradigm that can act as a unifying force and expose new avenues for exploration.

References

1. Object Management Group, "Semantics of Business Vocabulary and Rules Formal Specification v1.0", OMG document formal/08-01-02, January 2008. URL: http://www.omg.org/spec/SBVR/1.0/. Accessed: 11/6/2010.
2. World Wide Web Consortium, "Web Services Architecture", W3C Working Group Note 11 February 2004. URL: http://www.w3.org/TR/ws-arch/. Accessed: 11/6/2010.
3. Richardson, L. & Ruby, S. RESTful Web Services. O'Reilly Media, Inc. (2007)

4. 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.
5. R.T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," University of California – Irvine (2000.)
6. World Wide Web Consortium, "W3C Semantic Web Activity", URL: http://www.w3.org/2001/sw/, Accessed: 11/6/2010.
7. Berners-Lee, T., "Linked Data", (2006) URL: http://www.w3.org/DesignIssues/LinkedData.html, Accessed: 11/6/2010.
8. World Wide Web Consortium, "SPARQL Query Language for RDF", URL: http://www.w3.org/TR/rdf-sparql-query/, Accessed: 11/6/2010.
9. World Wide Web Consortium, "SPARQL Update: A language for updating RDF graphs" (2008), URL: http://www.w3.org/Submission/SPARQL-Update/, Accessed: 11/6/2010.
10. Gregorio, J. and De Hora, B "The Atom Publishing Protocol", Internet RFC 5023, October 2007. URL: http://www.ietf.org/rfc/rfc5023.txt, Accessed: 11/6/2010.
11. Marinos, A., Krause, P., "An SBVR Framework for RESTful Web Applications", in G. Governatori, J. Hall, and A. Paschke (Eds.): RuleML 2009, LNCS 5858, pp. 144–158, 2009.
12. Lee, D., Chu W.W., "Comparative analysis of six XML schema languages", ACM SIGMOD Record, v.29 n.3, p.76-87, September 2000.
13. U. Kuster, M. Stern, and B. Konig-Ries, "A Classification of Issues and Approaches in Automatic Service Composition", Intl. Workshop WESC, vol. 5, 2005.
14. Marinos, A., Razavi, A., Moschoyiannis. S., Krause, P., "RETRO: A (hopefully) RESTful Transaction Model", Technical Report CS-09-01, University of Surrey, Guildford, Surrey, August 2009.
15. Marinos, A., Krause, P., "Using SBVR, REST and Relational Databases to develop Information Systems native to the Digital Ecosystem", IEEE Conference on Digital Ecosystems Technologies 2009 (DEST 2009)

7 GENERATING SQL QUERIES FROM SBVR RULES (SURREY)

This paper was presented at RuleML 2010

The Business Rules Approach [1] has made significant strides in bridging the spheres of everyday human interactions and information technology. An outgrowth of that movement was the OMG standard Semantics of Business Vocabulary and Rules (SBVR) [2], which brought together research from linguistics, formal logics, as well as practical expertise. SBVR Models are considered constructs that are supposed to help businesses communicate with each other and also business people to communicate with implementers of information technology. Direct transformation of SBVR models into executable code is generally not encouraged and has often resulted in rather harsh compromises of SBVR's meta-model and intended use when attempted [3]. The reason for this mismatch is the chasm between the declarative paradigm implemented by SBVR and the imperative procedural paradigm that is at the heart of most modern programming and business process languages. So thus far, human programmers are needed to interpret and convert the SBVR models into real-world applications. An alternative approach, called Generative Information Systems [4], was presented by the authors of this paper that allowed for real-world systems to be produced by inferring the appropriate reaction directly from a model, without the need for an intermediate code generation step, and without the need for explicitly defined business processes. A significant aspect of that model was the method of generating schemas for a relational database from the SBVR Vocabulary, and converting rules into SQL queries to verify the consistency of the data set. This last step, had been only sketched out in the original paper, as the theoretical framework required for this undertaking is significant. This paper addresses precisely these issues and examines in detail the conversion of SBVR rules into SQL queries for the purpose of validating the consistency of a given data set with the SBVR model.

7.1 GENERATIVE INFORMATION SYSTEMS

This section summarizes the architecture of Generative Information Systems (GIS) [7] to provide appropriate context for the rest of the paper. A GIS is based around the concept that the logic of the system is accessible to the owner of the system, and that any change in the logic is immediately reflected in the operation of the system. The architecture as can be seen in Figure 1, specifies that both the RESTful API and the relational database schema (in SQL-DDL) are to be generated from the model.

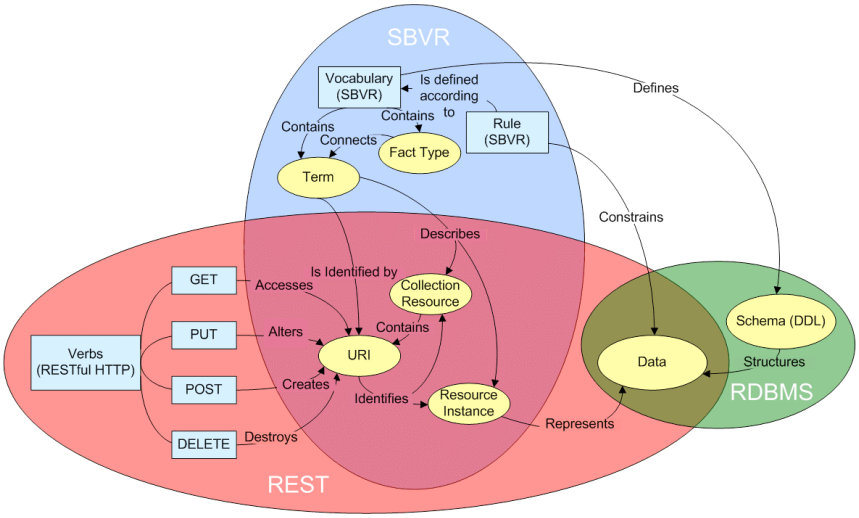


Fig. 11. Connections between REST, SBVR and Relational Databases

The end user can place requests on the system through the API. These requests get evaluated through the ruleset, and if they represent a legal transition and their result is a system in a consistent state, they are applied to the dataset. If not, the inconsistency is presented to the user, who can amend the request to take account of the new information. Through this back and forth negotiation, the user either concludes that the request is fundamentally incompatible with the system, or reaches a formulation of the request that satisfies both the original goal and the system's consistency requirements. This is, in an abstract sense, what many processes achieve. By guiding the user through a sequence of steps, they determine what change needs to be made to the system state to satisfy the user's request while maintaining consistency. We call thus abstract process the 'meta-process'. Its advantage over the traditional process-driven message is that it can respond to unforeseen requests by the user, in contrast to the hardcoded process model which is constrained to the design-time foresight of the developers. More detail on this process can be seen in Figure 2.

User:

POST <en101>

http://domain.org/students/John/courses/

System:

403 Forbidden

It is necessary that each **student** *is registered for* at most five **courses**

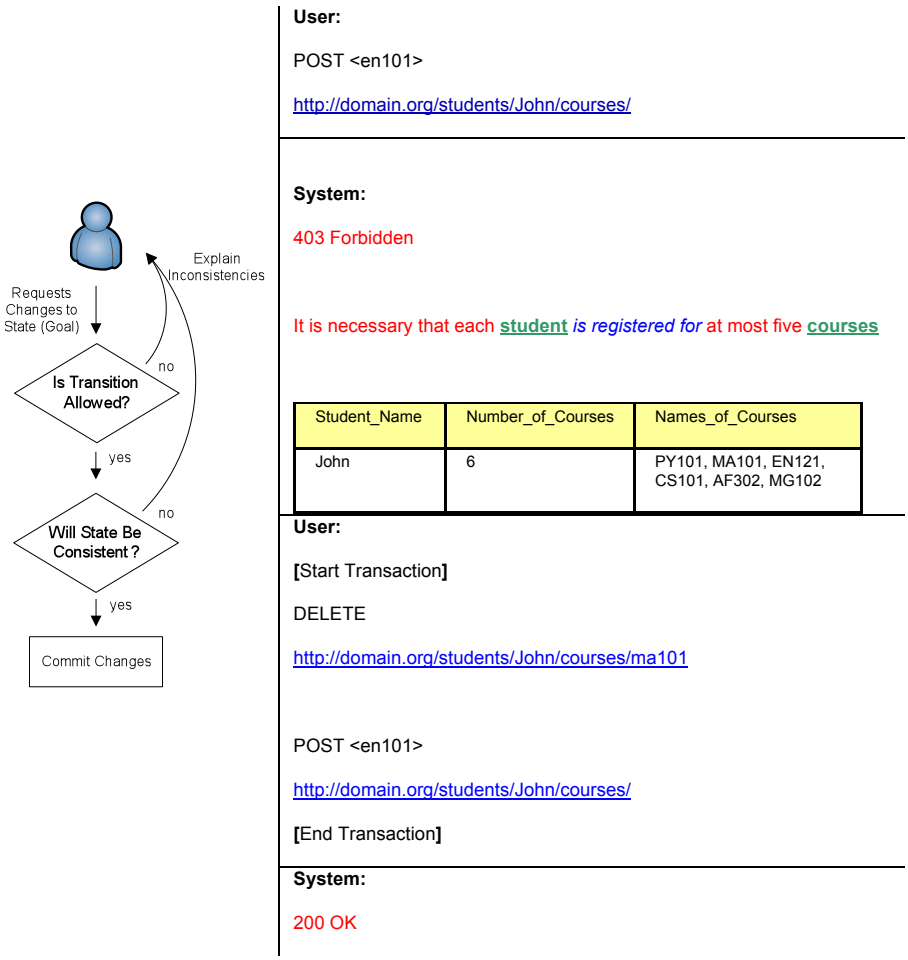


Fig. 12. The meta-process control structure

The way that consistency is currently checked is by performing a sequence of actions on the database as a transactional unit. First, a transaction is initiated. Secondly, the updates are applied to the dataset. If the database schema makes this impossible, the updates are rejected. If it is allowed, the relevant rules are checked against the dataset to make sure they are not violated. If they are violated, updates are rolled back and the details of the violation returned to the user. If the rules are not violated, the transaction proceeds. This mechanism is suitable for a proof of concept, but may have scalability limitations for concurrent systems. Optimizations can be explored that avoid the round-trip to the database. It is however interesting to note that this rough process of adding tentative information to the knowledge base, then checking for consistency, and deleting in case of violation, counterintuitively seems to be the way that the human brain deals with new knowledge. [5] This does not mean that the method is ideal, but it is an interesting parallel that we noted after setting the foundations for Generative Information Systems architecture.

The step in the above process that was left least defined is the one where the updated dataset is checked against relevant rules for consistency. This is done by transforming

each rule to an SQL query that requests violations to the rule to be returned (Figure 3). The precise mechanism by which this is carried out is the focus of this paper.

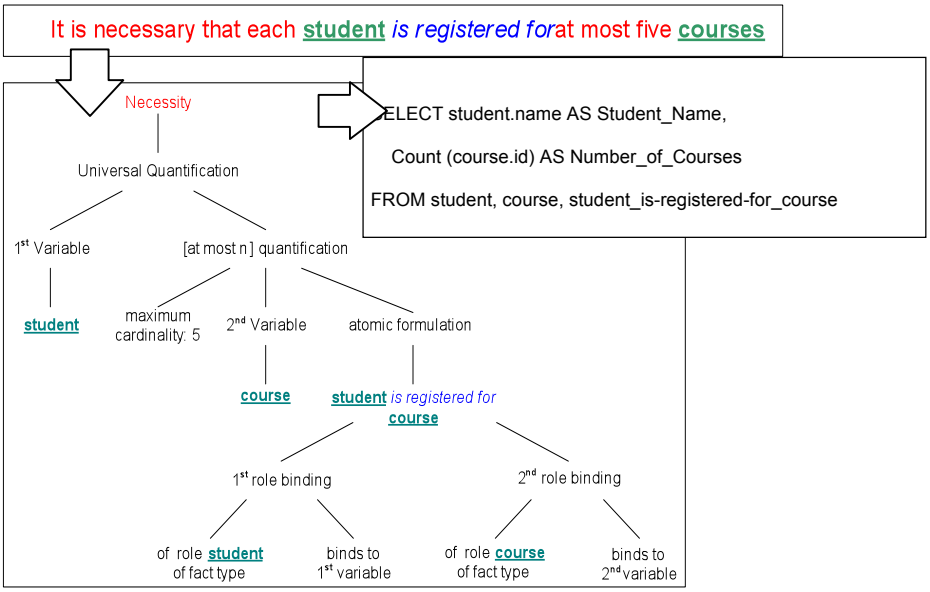


Fig. 13. From SBVR Structured English, to SBVR Logical Formulation, to an SQL query.

7.2 Vocabularies to SQL Schemas

For rules to be validated over a dataset however, first there must be a schema for that dataset. As our starting point is an SBVR model, it is the vocabulary that is the obvious candidate for becoming the scaffold for our schema. The detailed process has been described in our previous work so here we will instead go through an example scenario which we will use throughout this paper. The model for our example can be seen in Table 1. One aspect not covered in previous work is that of primitive data types. We can see that the term Name has a concept type of Varchar(255). This can be read as a reference to a vocabulary of primitive data types that a generative information system is built on. These terms are essentially terminal symbols that get mapped directly onto programming language data types. We use the data types that are fundamental to SQL as this is our target data store. Another novel convention is that since the Name is had by Student , this relation is constrained to a one-to-one cardinality, and Name has no other attributes than its value, we render it an attribute of the table with which it is associated rather than

representing it in a separate table, similarly for Code and Title.

Table 3. Example SBVR Model

Terms	Fact Types	Rules
Student Module Course Name Concept-type: Varchar(255) Code Concept-type: Varchar(255) Title Concept-type: Varchar(255)	Student *is registered for* course Student *is enrolled in* module Module *is available for* course Student *is under probation* Student *has* name Course *has* title Module *has* code	It is necessary that each student *is registered for* at most five courses. It is necessary that each student that *is under probation is registered for* at most three courses. It is obligatory that each student *has* exactly one name. It is obligatory that each course *has* exactly one title It is obligatory that each module *has* exactly one code.

The result of converting the vocabulary (and some of the more basic rules) into a schema can be seen in Figure 4.

```
CREATE TABLE student (id INT NOT NULL AUTO_INCREMENT,  name VARCHAR(255),
           is-under-probation BOOL, level INT, primary Key (id));


CREATE TABLE course (id INT NOT NULL AUTO_INCREMENT,
           code VARCHAR(255), primary Key (id));


CREATE TABLE module (id INT NOT NULL AUTO_INCREMENT,
           title VARCHAR(255), primary Key (id));
```

```
CREATE TABLE student_is-enrolled-in_module (studentID INT, moduleID INT,
           primary Key (studentID, moduleID),
           foreign Key (studentID) references student(id),
           foreign Key (moduleID) references module(id));


CREATE TABLE student_is-registered-for_course (studentID INT, courseID INT,
           primary Key (studentID, courseID),
           foreign Key (studentID) references student(id),
           foreign Key (courseID) references course(id));


CREATE TABLE course_is-available-for_module (courseID INT, moduleID INT,
           primary Key (courseID, module_id),
           foreign Key (courseID) references course(id),
           foreign Key (moduleID) references module(id));
```

Fig. 14. Resulting SQL DDL Schema

7.3 A TUPLE RELATIONAL CALCULUS EXTENSION FOR EXPRESSING RULES

We have seen how the SBVR model can be used to generate a schema for a relational database. The generated schema already constrains the dataset [6]. Business rules however, go beyond the kind of constraints captured in a relational schema such as domain constraints, and integrity constraints (primary, foreign keys).

Constraints and queries can very well be formalized. A formal description is necessary to describe the constraint or the query unambiguously – allow one and only one meaning. This has led to a long strand of work on relational theory (algebra and calculus), e.g., see [10], going back to the early ’70s [8], which provides a formal semantics for querying languages such as SQL.

To describe constraints and queries in a formal way we use first-order logic, set theory, and draw upon existing constructs in tuple relational calculus. Just like existing relational theory, the result is a predicate calculus based on the use of tuple variables. A tuple variable is a variable that ranges over a named relation (table).

In what follows, we give a description of the formal notation as well as a mapping of these expressions onto standard SQL. We use the student enrollment example, which includes a relational database and an SBVR model, to illustrate our approach.

7.3.1 PRELIMINARIES

We start by describing the bare basics in standard relational theory. The general form of a query in tuple relational calculus is

$$\{\mathbf{x} \mid F(\mathbf{x})\}$$

where **x** is the set of tuples for which the expression $F(\mathbf{x})$ is true. The relation is defined somewhere inside $F(\mathbf{x})$. As we will see, in our approach we make this explicit by separating the filter from the domain.

If only some attributes of **x** are of interest, the above expression takes the form

$$\{\mathbf{x} : (x_1, x_2, .., x_m) \mid F(\mathbf{x})\}$$

where $x_1, .., x_m$ are attributes of the relation which is the result of the query (i.e. attributes of a tuple **x**). This set is created by selecting all tuples **x** for which $F(\mathbf{x})$ is true, and then projecting those tuples on attributes $x_1, .., x_m$. The result of a query on a set of tuples (relation) is either a set of tuples matching a certain condition or a value (when using aggregate functions, cf. Section 4.3). For example, the query

$$\{\mathbf{x} : (name, id) \mid student(\mathbf{x})\}$$

returns a set of tuples which contain attributes *name* and *id* from the *student* relation.

A predicate $P(x)$ is a function that maps each element x of a set S to the value 'true' or 'false', i.e., $P : S \rightarrow \{true, false\}$.

Let $x \in N$ - so x is an element of the set of natural numbers. Then the predicate $P_1(x) \equiv x \geq 0$ is true for all *x* while the predicate $P_2(x) \equiv x < 0$ is false for all *x*.

Predicates can consist of one expression (as in $x \geq 0$ above) or as a combination of expressions. These combinations arise by combining expressions using the usual first-order or predicate logic operators (e.g. see [9] given in Table 2.

Table 4. Logical connectives

∧ (conjunction)	∨ (disjunction)	¬ (negation)	⇒ (implication)

Let $x \epsilon \boldsymbol{N}$, as before. The predicate $P_3(x) \equiv x < 7 \wedge x > 10$ combines the expressions $x < 7$ and $x > 10$ (and is false for all $x \epsilon \boldsymbol{N}$). The predicate $P_4(x) \equiv x \geq 3 \wedge x < 6$ is true for $x = 3,4,5$.

Predicates can be used to define sets. For example,

$$S_1 = \{x \mid x \in N \wedge P_1(x)\}$$

denotes the set of all x such that x is natural number ($x \in N$) and satisfies the

predicate P_1 (where $P_1(x) \equiv x \geq 0$, as before). We have seen that P_1 is true for all x which means that S_1 is the set of natural numbers, and we can write

$$S_1 = \{x \mid x \in N \wedge P_1(x)\} = N$$

Similarly, $S_2 = \{x \mid x \in N \wedge P_3(x)\} = \varnothing$ where predicate P_1 is as defined before.

The fact that predicates can be used to define sets is well-known in mathematics and is central to our approach – we will be using the set membership to identify relations and the predicate as the selection condition on the tuples of these relations.

If *p* and *q* are expressions that valuate to true or false, sometimes called WFFs for Well-Formed Formulae in the literature, e.g. see [6], then the following equations hold. These are standard in first-order logic, e.g. see [9], so we list them here in Table 3 without further explanation. A thorough treatment can be found in [9].

Table 5. Equations on expressions (WWF)

$\neg(\neg p) \equiv p$	$\neg(p \wedge q) \equiv \neg p \vee \neg q$	$\neg(p \vee q) \equiv \neg p \wedge \neg q$
$p \Rightarrow q \equiv \neg p \vee q$	$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$	$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$

We now turn our attention to the basic structure of a query expressed in SQL DML in our formalization which is an extension to the tuple relational calculus while staying within the predicate calculus semantics – in particular, we will be concerned with setting up formal semantics for transforming SBVR rules to SQL queries based on a predicate calculus with identity.

7.3.2 Basic structure of an SQL query

Since our interest is in transforming SBVR rules to SQL queries on the relational schema generated by the SBVR model, we will be concerned with predicates that define sets of tuples. The general form of a query in our predicate calculus is

$$\{\mathbf{x} \mid D(\mathbf{x}) \wedge P(\mathbf{x})\}$$

where $\mathbf{x} : (x_1, \ldots, x_n)$ is the set of tuples from a domain $D(\mathbf{x})$, and $D(\mathbf{x})$ specifies the set of all possible tuples that $\mathbf{x}$ ranges over, i.e. a relation with $x_1, \ldots, x_n$ attributes, and $P(\mathbf{x})$ is predicate on the set of all tuples in $D(\mathbf{x})$ (i.e. specifies the restriction to tuples that satisfy the condition).

For example, $\{\mathbf{x}|student(\mathbf{x}) \wedge student.id = '6081958'\}$ returns the set of all tuples **x** from the relation *student* whose attribute *id* has the value 6081958. (Note that *student.id* is a primary key in our relational database schema, given in Section 3, so this expression would return a single tuple.)

The following expression

$$\{\mathbf{x}|D(\mathbf{x}) \wedge P(\mathbf{x})\}$$

in the extended predicate calculus considered here is mapped to SQL DML as:

SELECT DISTINCT **x**

FROM $D(\mathbf{x})$

WHERE $P(\mathbf{x})$;

The SQL keyword DISTINCT is used to remove duplicates.

If we are only interested in certain attributes $x_1,..,x_n$ in the result **x** and not all attributes $x_1,..,x_m$ of the relation specified in $D(\mathbf{x})$, then we write for the projection

$$\{\mathbf{x}:(x_1, x_2..,x_n)|D(\mathbf{x}) \wedge P(\mathbf{x})\}$$

which is mapped to SQL DML as:

SELECT DISTINCT $x_1,..,x_n$

FROM $D(\mathbf{x})$

WHERE $P(\mathbf{x})$;

To express the JOIN statements in SQL DML which applies to two or more relations, we need to take a closer look at $D(\mathbf{x})$. In standard tuple relational calculus semantics, it is well known that joining two relations means taking the Cartesian product ($\times$) of the two relations. In our formalization, the join of two relations (tables) is captured in $D(\mathbf{x})$ which is what is used to specify the set of all tuples from which the returned set of tuples **x** come from. The join condition, if any, is then added in the predicate $P(\mathbf{x})$ - in addition to the selection condition, if any.

Therefore, if we want to join tuples from relations $\mathbf{y}_1,\mathbf{y}_2,..,\mathbf{y}_k$ we write

$$\{\mathbf{x} : (x_1,...,x_n) \mid D(\mathbf{y}_1) \times D(\mathbf{y}_2) \times \cdots \times D(\mathbf{y}_k) \wedge P(\mathbf{x})\}$$

where $D(\mathbf{y}_1) \times D(\mathbf{y}_2) \times \cdots \times D(\mathbf{y}_k) = D(\mathbf{x})$. This predicate calculus construction is mapped to SQL DML as:

SELECT $x_1,..,x_n$

FROM $D(\mathbf{y}_1) \times D(\mathbf{y}_2) \times \cdots \times D(\mathbf{y}_k)$

WHERE $P(\mathbf{x})$;

Note that $\mathbf{y}_1, \mathbf{y}_2,.., \mathbf{y}_k$ denote relations (sets of tuples) while $x_1,.., x_n$ is the list of attributes returned after the join of the relations, and this is denoted by $\mathbf{x} : (x_1,..., x_n)$. It is also worth pointing out that selection conditions on attributes of $\mathbf{y}_1, \mathbf{y}_2,.., \mathbf{y}_k$ are included in $P(\mathbf{x})$ since they are applied after the Cartesian product on these relations has been applied.

For example,

$$\{\mathbf{x} : (student.id, student.name, COUNT(course.id)) \mid D(student) \times D(s\_irf\_c) \times \\ \times D(course) \wedge student.id = s\_irf\_c.studentID \wedge s\_irf\_c.courseID = course.id\}$$

is transformed into:

SELECT student.id, student.name, COUNT(student.id)

FROM student, s_irf_c, course

WHERE student.id = s_irf_c.studentID AND

AND s_irf_c.courseID = course.id ;

We now turn our attention to arithmetic operations and aggregate functions.

7.3.3 ARITHMETIC OPERATIONS AND AGGREGATE FUNCTIONS

In the SQL, arithmetic operations may appear in the SELECT clause, as in:

SELECT Salary*1.1, EmpID, EmpName

FROM Employee

WHERE DeptName = 'Research';

which reflects the values of a 10% increase in salaries in the Research department. So we need to apply this arithmetic operation as a function on the returned set of tuples $\mathbf{x}$. For

this reason, we write

$$\{E(\mathbf{x}) \mid D(\mathbf{x}) \wedge P(\mathbf{x})\}$$

where $E(\mathbf{x})$ is a function on $\mathbf{x}$ that includes addition (+), subtraction (-), multiplication (*), division (/), or a combination of these on one or more attributes of the tuples in $\mathbf{x}$, i.e. tuples from $D(\mathbf{x})$ which satisfy $P(\mathbf{x})$.

Often, arithmetic operations only apply to certain attributes in the set of returned tuples $\mathbf{x}$. So E should be applied to the attributes of $\mathbf{x}$ rather than across $\mathbf{x}$. Thus,

$$\{\mathbf{x}.(E(x_1), E(x_2),..., E(x_n)) \mid D(\mathbf{x}) \wedge P(\mathbf{x})\}$$

where $E(x_i)$, $i = 1..n$, is applied to some attributes, in which case it is one or more of '+', '-', '*', '/' and not applied to others, in which case we have $E(x_i) = x_i$ (identity).

In similar fashion, we can address the aggregate functions in SQL DML, i.e. SUM, AVG, MIN, MAX, COUNT. To take into account the fact that an arithmetic operation may have been already applied to a certain attribute, we define F as a composite function on E so that $(F \circ E)(x_i) = F(E(x_i))$. In other words, F is applied to the output of E, and we write

$$\{\mathbf{x}.(F(E(x_1)), F(E(x_2)),..., F(E(x_n))) \mid D(\mathbf{x}) \wedge P(\mathbf{x})\}$$

Note that if F is the aggregate function COUNT, for some attribute x_i, then $E(x_i)$ must be the identity, i.e. $E(x_i) = x_i$, so that only attribute names are allowed in this case and no arithmetic operations.

This predicate calculus construction is mapped onto SQL DML as

SELECT $F(E(x_1)), F(E(x_2)),..., F(E(x_n))$

FROM $D(\mathbf{x})$

WHERE $P(\mathbf{x})$;

For example,

$$\{\mathbf{x}.(student.id, student.name, COUNT(student.id)) \mid student(\mathbf{x}) \wedge student.level = 3\}$$

is mapped onto

```
SELECT student.id, COUNT(student.id)
```

FROM student

WHERE student.level = '3' ;

and returns the number of final year students in the dataset.

We now turn our attention to grouping and filtering on groups.

7.3.4 GROUPING AND HAVING

The set up of the predicate calculus semantics so far has been laid out in a way that will allow grouping (GROUP BY) and grouping on condition (HAVING) to be addressed in a straightforward manner.

The grouping operation on a database comes down to stating the desired grouping attribute(s) and the grouping condition, if any. The grouping condition selects those groups that satisfy the condition and discards those who do not.

In our formalisation, the grouping attributes are specified before the projected attributes (and therefore will be mapped onto the SELECT clause in SQL DML) while the grouping condition will be part of the predicate itself. We note that it cannot be included in $P(\mathbf{x})$, like we did for JOIN, because the grouping condition applies to the results of the grouping operation, i.e., once the groups have been formed by the grouping operations.

Therefore, if we want to group a relation by a set of attributes $x_l,...x_n$ (a subset of all the attributes $x_1,...x_m$ of the relation), we write

$$\{\mathbf{x}:(x_l,..,x_m),\mathbf{x}:(x_1,..,x_n)|D(\mathbf{x})\wedge P(\mathbf{x})\}$$

$(x_l,..,x_m)$ and $(x_1,..,x_n)$ need not be disjoint but both need to be subsets of the set of attributes of $D(\mathbf{x})=(x_1,..,x_m)$. Finally, we note that $D(\mathbf{x})$ may be the result of the Cartesian product of a number of relations, as before.

The above expression in our extended tuple relational calculus is mapped onto SQL DML as:

SELECT $x_1,..,x_n$

FROM $D(\mathbf{x})$

WHERE $P(\mathbf{x})$

GROUP BY $x_1,..,x_m$

For example,

$$\{\mathbf{x}: student.id, \mathbf{x}: (student.id, student.name, COUNT(course.id)) |$$
$$| D(student) \times D(s\_irf\_c) \times D(course) \wedge$$
$$\wedge student.id = s\_irf\_c.studentID \wedge s\_irf\_c.courseID = course.id\}$$

returns the number of courses a student has taken, and does this for every student.

This translates to the following SQL query:

SELECT student.id, student.name, COUNT(course.id)

FROM student, s_irf_c, course

WHEREstudent.id = s_irf_c.studentID AND

AND s_irf_c.courseID = course.id

GROUP BY student.id ;

Next we may add the grouping condition as an additional predicate $H(\mathbf{x})$ which applies to the result (set of tuples) of the grouping operation, i.e. to the set of attributes in $(x_1,..,x_m) \subseteq (x_1,..,x_n)$. Therefore, we write

$$\{\mathbf{x}:(x_1,..,x_n) | \{\mathbf{x}:(x_1,..,x_m), \mathbf{x}: (x_1,..,x_n) | D(\mathbf{x}) \wedge P(\mathbf{x})\} \wedge H(\mathbf{x})\}$$

which is mapped onto SQL DML as:

SELECT $x_1,..,x_n$

FROM $D(\mathbf{x})$

WHERE $P(\mathbf{x})$

GROUP BY $x_1,..,x_m$

HAVING $H(\mathbf{x})$

Note that this is different to a nested predicate calculus expression because a nested query would simply apply a selection condition to the result of the inner query but could project onto different attributes while a grouping condition only filters the groups

returned by the grouping operation, and thus cannot apply a further projection. For a nested query we would write

$$\{\mathbf{x}:(x_1',..,x_n') \mid \{\mathbf{x}:(x_l,..,x_m), \mathbf{x}:(x_1,..,x_n) \mid D(\mathbf{x}) \wedge P(\mathbf{x})\} \wedge P'(\mathbf{x})\}$$

which would in turn map onto the following SQL DML:

SELECT $x_1',..,x_n'$

FROM (SELECT $x_1,..,x_n$

FROM $D(\mathbf{x})$

WHERE $P(\mathbf{x})$)

GROUP BY $x_l,..,x_m$)

WHERE $P'(\mathbf{x})$;

It can be seen that $P'(\mathbf{x})$ applies to the result of the inner query, but the result of the nested query as a whole can include a projection on any attributes from $D(\mathbf{x})$.

Going back to our example, if we want to check whether the business rule

It is necessary that each **student** *is registered for* at most five **courses**

expressed in the SBVR model given earlier in Figure 3 is satisfied, we need to restrict to groups (one for each student) who are associated with (registered for) more than five courses. We check for these cases since these are cases where the rule might be violated, and if this happens, the corresponding database operations will need to be executed as a transaction. Taking into account the associated database schema, this rule is expressed in terms of our extended predicate calculus as follows:

$$\{\mathbf{x}:(student.id, student.name, COUNT(student.id)) \mid$$
$$\mid \{\mathbf{x}:student.id, \mathbf{x}:(student.id, student.name, COUNT(course.id)) \mid$$
$$\mid D(student) \times D(s\_irf\_c) \times D(course) \wedge student.id = s\_irf\_c.studentID \wedge$$
$$\wedge s\_irf\_c.courseID = course.id\} \wedge (COUNT(course.id) > 5)\}$$

which is in turn mapped onto the following SQL DML statements:

SELECT student.id, student.name, COUNT(course.id)

```
FROM        student, s_irf_c, course
WHEREstudent.id = s_irf_c.studentID AND
                        AND s_irf_c.courseID = course.id
GROUP BY    student.id
HAVING      COUNT(course.id) > 5 ;
```

It is in this way that we can take rules from an SBVR model and transform them into SQL DML so that we can then check whether they are satisfied on a relational database schema by executing a standard SQL query. In the next section we attempt to generalize this by taking a closer look at both ends, our predicate calculus -based formalisation and the SBVR-LF, and do so at the semantics level.

7.4 MAPPING ONTO THE SBVR-LF SEMANTICS

In this section we turn our attention to the SBVR Logic Formulation (SBVR-LF) as defined in the SBVR specification document [2], which provides the semantic foundation for Rules in SBVR, and define a mapping onto the extension of the predicate calculus described in the previous section. The objective is to obtain a generic mapping between rules expressed in SBVR and our relational theory–based formalisation of such rules that makes them amenable to validation against a dataset.

Before we embark on the mapping, we define the quantifiers within the predicate calculus semantics in our approach. There are two quantifiers in predicate logic that can be used in an expression (WFF) to find out how many elements of the corresponding set satisfy the expression.

Let $P(\mathbf{x})$ be a predicate (as before). Let $D(\mathbf{x})$ denote the domain of $\mathbf{x}$, i.e. the set of all possible values for the tuple $\mathbf{x}$. To find out if *for at least one* tuple $\mathbf{x}$ from the domain $D(\mathbf{x})$ the predicate $P(\mathbf{x})$ is true, we write

$$(\exists \mathbf{x})(D(\mathbf{x}) \wedge P(\mathbf{x}))$$

which is read as "there is an $\mathbf{x}$ for which $D(\mathbf{x})$ holds, and $P(\mathbf{x})$ is true". The result of this expression is either true or false.

To find out if for all tuples in the domain $D(\mathbf{x})$ the predicate $P(\mathbf{x})$ is true, we write

$$(\forall \mathbf{x})(D(\mathbf{x}) \Rightarrow P(\mathbf{x}))$$

which is read as "for all $\mathbf{x}$ for which $D(\mathbf{x})$ holds, $P(\mathbf{x})$ is true". The result of this expression

is true or false.

With reference to the example predicates discussed in the start of Section 4.1, the expression $(\exists x)(x \in \boldsymbol{N} \wedge P_3(x))$ is false. The expression $(\forall x)(x \in \boldsymbol{N} \Rightarrow P_1(x))$ is true.

Note the difference between $(\exists \mathbf{x})(D(\mathbf{x}) \wedge P(\mathbf{x}))$ and $(\forall \mathbf{x})(D(\mathbf{x}) \Rightarrow P(\mathbf{x}))$ which can yield different results (true or false) for the same expression. To avoid such ambiguities the domain $D(\mathbf{x})$ of an expression with a universal quantifier is always placed to the left of the implication logical operator ($\Rightarrow$).

Again, drawing upon first-order logic we have that if $P(\mathbf{x})$ is a predicate and $D_1(\mathbf{x})$, $D_2(\mathbf{x})$ are domains (expressions that define relations from the database schema generated by the SBVR vocabulary, as discussed in Section 3, and hence restrict the set of all possible values for a tuple $\mathbf{x}$), then the following equations hold.

Table 6.

$(\forall \mathbf{x})(P(\mathbf{x})) \equiv \neg(\exists \mathbf{x})(\neg P(\mathbf{x}))$	$(\exists \mathbf{x})(P(\mathbf{x})) \equiv \neg(\forall \mathbf{x})(\neg P(\mathbf{x}))$
$(\forall \mathbf{x})(D_1(\mathbf{x}) \times D_2(\mathbf{x}) \Rightarrow P(\mathbf{x})) \equiv (\forall \mathbf{x})(D_1(\mathbf{x}) \Rightarrow (D_2(\mathbf{x}) \Rightarrow P(\mathbf{x})))$	
$(\forall \mathbf{x})(D_1(\mathbf{x}) \times D_2(\mathbf{x}) \Rightarrow P(\mathbf{x})) \equiv \neg(\exists \mathbf{x})((D_1(\mathbf{x}) \times D_2(\mathbf{x})) \wedge \neg P(\mathbf{x}))$	

We have given these standard equations in terms of predicates that define sets of tuples. In their general form, they apply to an element x rather than a tuple $\mathbf{x}$ and we would also have $\wedge$ instead of $\times$ in the last two.

The specification document of SBVR includes the definition of the Formal Logic and Mathematics Vocabulary [2, pp. 109-118] which provides the logical foundations for SBVR in terms of first-order logic. However, the SBVR specification predefines some numeric quantifiers [2, pp.97-98] in addition to the standard universal and existential quantifiers found in first-order predicate logic. These allow the user to say things like ‘exactly one car’ or ‘exactly two cars’ or ‘at most 8 and at least 3 cars’ or ‘at most two cars’ and so on.

Due to space limitations we do not reproduce the SBVR predefined quantifiers here, and rather refer the interested reader to the SBVR specification [2, pp.97-98].

The predefined quantifiers can be defined in terms of the quantifiers in our

formalisation which were defined earlier in the standard way of quantifiers in predicate logic, which is what we have used up to now in expressing SBVR rules in terms of SQL queries (previous section). Drawing upon the definition schemas in [11], also outlined in [2], we may obtain a rewriting of the SBVR predefined quantifiers in our approach.

The *exactly one quantifier* in SBVR-LF, denoted by $\exists^1\mathbf{x}$, can be rewritten as:

$$(\exists^1\mathbf{x})(D(\mathbf{x}) \wedge P(\mathbf{x})) \equiv (\exists\mathbf{x})(D(\mathbf{x}) \wedge P(\mathbf{x})) \wedge (\forall\mathbf{y})(D(\mathbf{y}) \Rightarrow P(\mathbf{y}) \wedge \mathbf{y} = \mathbf{x})$$

The *at most n quantifier* given in SBVR-LF, denoted by $\exists^{0..n}\mathbf{x}$, can be rewritten in terms of our predicate calculus as:

$$\begin{aligned}(\exists^{0..n}\mathbf{x})(D(\mathbf{x}) \wedge P(\mathbf{x})) \equiv \neg(\exists\mathbf{x})(D(\mathbf{x}) \wedge P(\mathbf{x})) \vee (\exists\mathbf{x}_1)(D(\mathbf{x}_1) \wedge P(\mathbf{x}_1)) \vee \\ \vee((\exists\mathbf{x}_1)(D(\mathbf{x}_1) \wedge P(\mathbf{x}_1) \wedge (\exists\mathbf{x}_2)(D(\mathbf{x}_2) \wedge P(\mathbf{x}_2)) \wedge \neg(\mathbf{x}_1 = \mathbf{x}_2) \vee \\ \vdots \\ \vee((\exists\mathbf{x}_1)(D(\mathbf{x}_1) \wedge P(\mathbf{x}_1) \wedge \cdots \wedge (\exists\mathbf{x}_n)(D(\mathbf{x}_n) \wedge P(\mathbf{x}_n)) \wedge \neg(\mathbf{x}_1 = \cdots = \mathbf{x}_n) \wedge \\ \wedge(\forall\mathbf{y})(D(\mathbf{y}) \Rightarrow P(\mathbf{y}) \wedge ((\mathbf{y} = \mathbf{x}_1) \vee \cdots \vee (\mathbf{y} = \mathbf{x}_n)))\end{aligned}$$

The first disjunction covers the case that there might not exist such a tuple $\mathbf{x}$ (case of 0), the second covers the case there is one such $\mathbf{x}$, the third is for two such $\mathbf{x}$, and so on. The last disjunction says that *n* such $\mathbf{x}$ may exist, but then there cannot be any more (*n+1*) tuples that satisfy the predicate.

Similarly, the *at least n quantifier*, denoted by $\exists^{n..}\mathbf{x}$, can be rewritten as:

$$\begin{aligned}(\exists^{n..}\mathbf{x})(D(\mathbf{x}) \wedge P(\mathbf{x})) \equiv (\exists\mathbf{x}_1)(D(\mathbf{x}_1) \wedge P(\mathbf{x}_1)) \wedge (\exists\mathbf{x}_2)(D(\mathbf{x}_2) \wedge P(\mathbf{x}_2) \wedge \neg(\mathbf{x}_1 = \mathbf{x}_2)) \wedge \\ \wedge \cdots \wedge (\exists\mathbf{x}_n)(D(\mathbf{x}_n) \wedge P(\mathbf{x}_n) \wedge \neg(\mathbf{x}_n = \mathbf{x}_1) \wedge \cdots \wedge \neg(\mathbf{x}_n = \mathbf{x}_{n-1})) \wedge \\ \wedge((\exists\mathbf{x}_{n+k})(D(\mathbf{x}_{n+k}) \wedge P(\mathbf{x}_{n+k}) \wedge \neg(\mathbf{x}_{n+k} = \mathbf{x}_1) \wedge \cdots \wedge \neg(\mathbf{x}_{n+k} = \mathbf{x}_n)) \vee \\ \vee(\forall\mathbf{x}_{n+1})(D(\mathbf{x}_{n+1}) \Rightarrow P(\mathbf{x}_{n+1}) \wedge ((\mathbf{x}_{n+1} = \mathbf{x}_1) \vee \cdots \vee (\mathbf{x}_{n+1} = \mathbf{x}_n))))\end{aligned}$$

The first *n-1* conjunctions refer to each of the *n* tuples $\mathbf{x}$ that must exist, must satisfy the predicate. The last conjunction captures the fact that there may be *k* additional such $\mathbf{x}$ that satisfy the predicate or no other $\mathbf{x}$ (apart from the *n* we already have) may exist that satisfy the predicate.

The *at least n and at most m quantifier* given in SBVR-LF, and denoted by $\exists^{n..m}\mathbf{x}$, can be obtained by combining the rewriting of the *at least n* and that of the *at most n quantifiers* given earlier.

The intention behind SBVR-LF is to (be able to) capture business facts and business rules formally. Formal statements of business rules may then be transformed into logical formulations that can be read in software tools, or readily adopted in approaches like the one we describe in this paper. An example given in the specification [2, pp.90-91] is the formalisation of a static constraint that says 'each person was born on some date' as the logical formulation:

$$\forall x : person, \exists y : Date, x \text{ was born on } y$$

Going back to our example, the rule in our SBVR model can be written as:

$$\forall \mathbf{x} : \underline{\textbf{student}},\ \exists^{0..5}\mathbf{y} : \underline{\textbf{course}},\ \mathbf{x}\ \textit{is registered for}\ \mathbf{y}$$

With reference to the tree representation of this rule given in Figure 3 earlier, it can be seen that the root is a universal quantification ($\forall$), the 1st variable is **student**, the 2nd variable is **course** and the max cardinality is 5 ($\exists^{0..5}$) while the atomic formulation that completes the [at most n] quantification node is **student** *is registered for* **course** and this binds the 1st variable to x and the 2nd to y.

In fact we are interested in disproving the rule, i.e. identifying students registered for 6 or more courses. This can be encoded by taking the negation of the logical formulation in which case the existential quantifier $\neg\exists^{0..5}$ gives $\exists^{6..}$. Thus, we have

$$\exists \mathbf{x} : \underline{\textbf{student}},\ \exists^{6..}\mathbf{y} : \underline{\textbf{course}},\ \mathbf{x}\ \textit{is registered for}\ \mathbf{y}$$

Now **student** and **course** are relations in our database schema (Figure 4) and so is *is registered for*, thus all three appear in the domain $D(\mathbf{x})$ (in a Cartesian product) and consequently in the FROM clause of the resulting SQL query. The primary key of **student** will have to match the foreign key of *is registered for*, similarly for **course**. These join conditions become the predicate $P(\mathbf{x})$ and hence appear in the WHERE clause. The cardinality on the existential quantifier (6 or more) is the condition applied to the resulting tuples (per student), hence becomes the predicate $H(\mathbf{x})$ and appears in the HAVING clause.

SBVR currently allows the user to stay with first-order logic, and its variations [2]. More generally, the logical foundations are based on a predicate calculus with identity, so as to

be able to talk about two (or more) different things of the same kind, e.g., there ‘at least two cars’, or ‘at least two employees’. This is the primary reason for staying with predicate calculus in our formalisation in the previous section - the other reason is that predicate calculus is at the heart of the tuple relational calculus that underpins query languages such as SQL. Modalities aside, the predicate calculus we presented in the previous sections provides a bridge between the SBVR logical foundations for expressing business rules formally and SQL DML.

7.5 Conclusions and Future Work

In this paper we have briefly described the concept of generative information systems, and how rule-based modeling is at their core. We have discussed how an SBVR model (terms, fact types) is transformed into a relational schema that can act as a data store for our information system. By showing how the user interacts with the system, we have demonstrated the need for a formal and rigorous approach to transforming SBVR rules to SQL queries. This means that a rule can be validated against the dataset in much the same way as issuing a query on a database. We have achieved this by building on a predicate calculus semantics which can be seen as an extension to the tuple relational calculus.

The proposed transformation of SBVR-LF rules to SQL queries includes provision for aggregate functions and arithmetic operations, and extends this relational theory to address more advanced constructs such as grouping (GROUP BY clause) and grouping on condition (HAVING clause). We have also shown how predefined quantifiers in SBVR-LF can be re-written in terms of the quantifiers in our predicate calculus. These extensions were given within a predicate calculus with identity so that the semantic mapping between SBVR-LF and our formalisation is preserved.

The work in [12] is also concerned with producing SQL from business rules. However, the rules are expressed in ORM-based language ConQuer and the mapping between the rule and the resulting SQL is not attempted at the semantic level (at least not through relational theory). Our approach is concerned with rules expressed in SBVR-LF, and we have laid out the transformation of SBVR rules to SQL queries by considering the semantics of each and exploiting the common ground offered by predicate calculus with identity in both.

To further the research discussed, the transformations need to be converted to algorithms and implemented such that they can be applied to real-world problems.

Another direction is to add model checking capabilities to the model execution functionality described such that models with inconsistent, redundant, or needlessly complex rules can be identified. Finally, Constraint Logic Programming solvers can be used to answer questions over the dataset of the information system, receiving SBVR model fragments as their problem definition.

7.6 REFERENCES

16. R. G. Ross, "The Business Rules Manifesto," Business Rules Group. Version 2 (2003)
17. Object Management Group, "Semantics of Business Vocabulary and Rules Formal Specification v1.0", OMG document formal/08-01-02, January 2008. URL: http://www.omg.org/spec/SBVR/1.0/. Accessed: 14/5/2010
18. Open Philosophies for Associative Autopoietic Digital Ecosystems, 2008. "Automatic code structure and workflow generation from natural language models". URL: http://files.opaals.eu/OPAALS/Year_2_Deliverables/WP02/D2.2.pdf . (14/5/2010)
19. Marinos, A., Krause, P., "An SBVR Framework for RESTful Web Applications", Proceedings of the International Symposium on Rule Representation, Interchange and Reasoning on the Web, Springer-Verlag Berlin, Heidelberg, 2009, pp. 144-158.
20. Gilbert, D., Tafarodi, R. and Malone, P. 1993. 'You can't not believe everything you read'. '*Journal of Personality and Social Psychology*, **65**(2), 221-233.
21. Date, C.J. *An Introduction to Database Systems*. Addison-Wesley, 2004.
22. Marinos, A. Krause, P. "An SBVR Framework for RESTful Web Applications". Proceedings of RuleML 2009, G. Governatori, J. Hall, A. Paschke (eds), LNCS 5858, pp144-158, Springer, 2009.
23. Codd, E.F. "Relational Completeness of Data Base Sublanguages", Proceedings of Data Base Systems, Courant Computer Science Symposia Series 6, R. J. Dustin (ed), Prentice Hall, 1972.
24. Huth, A.R.A, Ryan, M.D., *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge University Press, 2002.
25. Nakano, R. "Translation with Optimization from Relational Calculus to Relational Algebra having Aggregate Functions". *ACM Transactions on Database Systems*, 15(4):518-557, ACM, 1990.
26. Halpin, T. A. "A Logical Analysis of Information Systems: Static Aspects of the Data-Oriented Perspective". PhD Thesis, University of Queensland, 1989.
27. Bloesch, A.C., Halpin, T.A. "ConQuer: a Conceptual Query Language", Proc. 15th Int'l Conf. on Conceptual Modelling, LNCS 1157, pp. 121-133, Springer, 1996.