OPAALS PROJECT

Contract n° IST-034824

WP3: Autopoietic P2P Networks

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

Part I

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

Contract Number: IST-034824

Project Acronym: OPAALS

Deliverable N°: D3.15 (Part I)

Due date: May 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.

Author: Surrey

Partners contributed: Surrey, IPTI, NAICA, WIT, IITK, SUAS

Made available to: All

VERSIONING		
VERSION	DATE	NAME, ORGANIZATION
0.1	**7/2010**	**A. RAZAVI (SURREY)**
0.2	**8/2010**	**A. RAZAVI (SURREY)**
0.5	**9/2010**	**A. RAZAVI, S. MOSCHOYIANNIS (SURREY), P. MALONE (WIT), J. FINNEGAN (WIT), A. AGRAWA (IITK), A. MARGARITO (NAICA), D. WILSON SOUZA ROSA (IPTI), T. J. HEISTRACHER (SUAS)**
1.0	**9/2010**	**P. KRAUSE, A. MARINOS (SURREY)**
2.0	**10/2010**	**A. MARINOS, P.KRAUSE (SURREY)**
2.1	**10/2010**	**P.KRAUSE, S. MOSCHOYIANNIS**

Quality check
Internal Reviewers: P. Dini (LSE); G. Briscoe (LSE)

Dependencies:

Achievements	Formalisation of recoverable interaction model for Digital Ecosystems (DEs). Integration of interaction model with the *Flypeer* P2P network infrastructure. Integration into *Flypeer* of support for the use of SBVR as an interface language for business analysts. Extension of SBVR to support processes Integration of support for execution of simulations into *Flypeer*. Extension of the P2P network infrastructure with distributed storage. Foundational support for use of a pure declarative interpretation of SBVR to generate information systems.
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*	S. Moschoyiannis, A. Razavi, and P. Krause, (2010), "Transaction Scripts: Making Implicit Scenarios Explicit," Electronic Notes in Theoretical Computer Science, vol. 238, Jun. 2010, pp. 63-79. *S. Moschoyiannis, A. Marinos and P. Krause* Generating SQL Queries from SBVR Rules, Proc. RuleML 2010. *A. Marinos and P. Krause,* Towards the Web of Models: A Rule-driven RESTful Architecture for Distributed Systems, Proc. RuleML 2010. *A. Marinos, S. Moschoyiannis and P. Krause*, An SBVR to SQL Compiler, Proc. RuleML 2010 (A "RuleML Challenge" demonstrator). *A. Marinos, E. Wilde, J. Lu*: HTTP database connector (HDBC): RESTful access to relational databases. WWW 2010: 1157-1158. *C. Pautasso, E. Wilde, A. Marinos*: Proceedings of the First International Workshop on RESTful Design, WS-REST 2010, Raleigh, North Carolina, USA, April 26, 2010 ACM 2010 *A. Marinos, P. Krause (2010)* **Optimisation by Proxy**. In *Web Science '10*. *A. Razavi, P. Krause, S. Moschoyiannis.* Digital Ecosystems: challenges and proposed solutions. In *Handbook of Research on P2P and Grid Systems for Service-Oriented Computing: Models, Methodologies and Applications*, IGI Global Publishers, 2010. *A. Razavi, S. Moschoyiannis, P. Krause* (2009) **An open digital environment to support business ecosystems**, 367-397. In *Peer-to-Peer Networking and Applications* 2 (4). *Razavi A, Marinos A, Moschoyiannis S, Krause P,* **Recovery management in RESTful Interactions,** 3rd IEEE International Conference on Digital Ecosystems and Technologies, Istanbul, TURKEY, 01 Jun 2009 - 03 Jun 2009. 2009 3RD IEEE INTERNATIONAL CONFERENCE ON DIGITAL ECOSYSTEMS AND TECHNOLOGIES. IEEE. 436-441. 2009 *Marinos A, Krause P*, **What, not How: A generative approach to**

	service composition, 3rd IEEE International Conference on Digital Ecosystems and Technologies, Istanbul, TURKEY, 01 Jun 2009 - 03 Jun 2009. 2009 3RD IEEE INTERNATIONAL CONFERENCE ON DIGITAL ECOSYSTEMS AND TECHNOLOGIES. IEEE. 430-435. 2009 *Marinos A, Krause P,* **Using SBVR, REST and Relational Databases to develop Information Systems native to the Digital Ecosystem,** 3rd IEEE International Conference on Digital Ecosystems and Technologies, Istanbul, TURKEY, 01 Jun 2009 - 03 Jun 2009. 2009 3RD IEEE INTERNATIONAL CONFERENCE ON DIGITAL ECOSYSTEMS AND TECHNOLOGIES. IEEE. 424-429. 2009 *Marinos A, Krause P,* **An SBVR Framework for RESTful Web Applications,** International Symposium on Rule Interchange and Applications, Las Vegas, NV, 05 Nov 2009 - 07 Nov 2009. Editors: Governatori G, Hall J, Paschke A. RULE INTERCHANGE AND APPLICATIONS, PROCEEDINGS. SPRINGER-VERLAG BERLIN. 5858: 144-158. 2009 *Razavi A, Marinos A, Moschoyiannis S, Krause P,* **RESTful Transactions Supported by the Isolation Theorems**, 9th International Conference on Web Engineering, San Sebastian, SPAIN, 24 Jun 2009 - 26 Jun 2009. Editors: Gaedke M, Grossniklaus M, Diaz O. WEB ENGINEERING, PROCEEDINGS. SPRINGER-VERLAG BERLIN. 5648: 394-409. 2009 *Marinos A, Razavi A, Moschoyiannis S, Krause P,* **RETRO: A Consistent and Recoverable RESTful Transaction Model,** IEEE International Conference on Web Services (ICWS 2009), Los Angeles, CA, 06 Jul 2009 - 10 Jul 2009. Editors: Damiani E, Zhang J, Chang R. 2009 IEEE INTERNATIONAL CONFERENCE ON WEB SERVICES, VOLS 1 AND 2. IEEE. 181-188. 2009
PhD Students*	Alexandros Marinos – technical development of REST and declarative SBVR aspects of the work reported here.
Outstanding features*	We believe this is the most significant and extensive exploration of the use of SBVR to date. It includes practical realisations of the use of SBVR, with: 1. The use of SBVR to semantically annotate service descriptions in Flypeer; 2. The extension of SBVR to include process concepts; 3. The extensive exploration of a “generative” approach to

	information systems development that remains strictly within the declarative intention of SBVR This work has been presented at WWW 2010, and also featured in an associated workshop on RESTful Design (with A. Marinos as co-chair). It is also the subject of two presentations at RuleML 2010, as well as in a successful "RuleML Challenge" submission (a tough call, with seven referees to convince).
Disciplinary domains of authors*	A. RAZAVI (SURREY), P. MALONE (WIT), J. FINNEGAN (WIT), A. AGRAWA (IITK), A. MARGARITO (NAICA), D. WILSON SOUZA ROSA (IPTI), T. J. HEISTRACHER (SUAS), P. KRAUSE, A. MARINOS (SURREY) ARE ALL FROM THE COMPUTING DOMAIN.

TABLE OF CONTENTS

EXECUTIVE SUMMARY

This report collects together a series of publications and reports that capture the OPAALS contributions to the support for the specification and execution of long-term transactions over the web, that require a collection of published services or resources. Three themes are covered:

1. The development of a recoverable interaction model for long-term transactions;
2. The development and evaluation of a distributed framework for supporting P2P business transactions (*Flypeer*);
3. The exploration of the use of SBVR for the specification of processes and transactions.

The report opens with our latest work on the development of a formal foundation for analysing the behaviour of the OPAALS model of long-running transactions and the distributed orchestration of the underlying service compositions. The formal semantics of behaviour of long-running transactions is aimed at describing the behavioural patterns services should follow in order to guarantee successful commitment or compensation within the transaction flow manager.

This is then followed by a section on the integration of support for long-term transactions into the Flypeer P2P network. We describe how services are cached for efficiency, and how transactions are coordinated using the aforementioned transaction model. This implementation has been evaluated in the Aragon Case Study. However, this part of the report covers low-level technical details that businesses and knowledge workers will not normally want to be concerned with. In order to take advantage of all the P2P benefits of Flypeer, it must be easy for businesses to create, deploy, execute services and must be simple to create and execute transactions. Consequently, as reported in Chapter 3, we provide support for the specification of a business transaction that involves several services provided from different small and medium enterprises (SMEs) in a digital ecosystem, that is realized by a p2p network based upon Flypeer, by means of a natural language based description (*Semantic for Business Vocabulary and Rule* - SBVR - has been used) in order to minimize the programming effort. The discovery of services involved in these transactions has been done via SBVR vocabularies, which are semantically annotated to the WSDL description of these services.

The *Semantics of Business Vocabulary and Business Rules (SBVR)*, an OMG standard, provides a meta-model for semantic and declarative models of business vocabulary and business rules. The logical formulation of SBVR facilitates IT people to interpret these models generated by business people. However, an important aspect of a business is the process models, and this is outside the scope of SBVR. The knowledge intensive and dynamic nature of business processes require such a declarative meta-model for process modeling in order to provide flexibility and adaptability. In Chapter 4, an approach for process modeling using SBVR's methodology is proposed. We have made an initial attempt to define a declarative meta-model, *Semantics of Business Process Vocabulary and Process Rules (SBPVR)*. This work is not currently fully integrated into the OPAALs demonstrator. Nevertheless, it contains valuable ideas that could be integrated at a future date.

The EvESimulator, or short EvESim, is then introduced in Chapter 5 as an inter-stakeholder simulation, emulation and testing framework. For this paper only a short summary is given, as an extensive description of the EvESim is documented in *D10.20 Emulation and testing of OPAALS P2P infrastructure by utilisation of EvESim*. D10.20 also includes a comprehensive description of the visualisation capabilities of EvESim including Google Earth, NASA World Wind and Jung. The code and interfaces of the visualisation components are reported in *D10.19 Extension of Google maps visualisation for large-scale social networks*. EvESim acted as one of the testing frameworks for the OPAALS architecture. Additionally, the EvESim framework runs partly on Flypeer, which makes it one of the first software products using Flypeer.

As well as support for transactions, our concept of a Digital Ecosystem requires Flypeer to be extended with a distributed file system. This is described in Chapter 6. It was not our intention to develop a novel solution to distributed storage in the OPAALS project as there is significant ongoing work in this area already. Rather, we summarise how distributed storage is integrated into the Flypeer P2P technology and other components of the OPAALs DE infrastructure.

A real instance of an OPAALS Digital Ecosystem will require the use of search services to find candidate resources. However, the problem that faces algorithm designers is that they inevitably need to use some proxy property (or combination of properties) as an indicator for some harder to explicate quality (such as “page quality” or “relevance”). However, when intelligent actors with different motivations are involved, the existence of the algorithm reifies the proxy into a separate attribute to be manipulated with the goal of altering the algorithm's results. What follows is an arms race of continuous updates by the algorithm designers and continuous interrogation and manipulation of the proxies by the optimized entities. In Chapter 7, we examine the types of distortion that can be introduced, focusing on two web-relevant examples of optimization by proxy, and discuss potential alternatives.

We return to our core work on support for web services and distributed transactions in Chapter 8. Competing visions have been jostling to define the long-term future of the Web: WS-* Web Services, the Semantic Web, and RESTful Web Services. Chapter 8 presents the initial steps towards a Rule-driven, REST-based architecture for the Web that can enable use cases that the Semantic Web and WS-* communities require. The key enabling ingredient is the use of SBVR models as a media type for resource description that allows models to be exposed and consumed. With formal description of data, advanced scenarios such as inference, service composition, and transactions are feasible within an architecture that is backwards-compatible with today’s Web.

This chapter should be seen as an introduction to a major research activity. Our first objective is to realize a new type of information system, more dynamic and less opaque to its owners, modelled with structured natural language rules and queried through hypermedia. To accomplish this, it will focus on SBVR as a modelling language, REST as an interface paradigm and Relational Databases as the persistence mechanism. All three of these technologies have declarative underpinnings, focusing on the ‘what’ rather than the ‘how’, which is why their combination is feasible and effective. By creating mappings to align these technologies, and completing gaps where necessary with new technologies, we

create the foundations for a purely declarative information system. The resulting system can exhibit process-like behaviour *without* having processes explicitly defined and can therefore gracefully handle unforeseen requests by its users.

Realising this has been a major activity in its own right, and we would like to document it in full as it contains much material of interest. So instead of summarizing the results in this volume, we present the work in full in a separate appendix of D3.15.

AIMS & OBJECTIVES OF THE REPORT

The overall aims and objectives of this report are to record the:

- Formalisation of recoverable interaction model for Digital Ecosystems (DEs);
- Integration of this interaction model with the *Flypeer* P2P network infrastructure;
- Integration into *Flypeer* of support for the use of SBVR as an interface language for business analysts;
- Extension of SBVR to support processes;
- Integration of support for execution of simulations into *Flypeer*;
- Extension of the P2P network infrastructure with distributed storage;
- Foundational support for use of a pure declarative interpretation of SBVR to generate information systems.

1 A RECOVERABLE INTERACTION MODEL FOR DIGITAL ECOSYSTEMS (SURREY)

As discussed in the deliverable D3.10, current protocols in transaction frameworks targeted at supporting business activities between networked organizations provide a specific pattern of behaviour, which not only violates the primary concept of SOC and Digital Ecosystems but also does not provide a truly distributed coordination model.

In this chapter, a formal foundation is provided for analysing the behaviour of the proposed model of long-running transactions and the distributed orchestration of the underlying service compositions. The formal semantics of behaviour of long-running transactions is aimed at describing the behavioural patterns services should follow in order to guarantee successful commitment or compensation within the transaction flow manager.

The proposed formal model for transactions, is based on Moschoyiannis', work and has been published at [1], and [2]. It uses ideas taken from a variety of theories for describing the behaviour of communicating systems, from Shields' vector languages [3], [4], [5] to Mazurkiewicz traces [6], [7] to event structures [8] to process algebras [9], [10]. It draws upon a vector language-based description of behaviour, which allows monitoring or recording a number of communicating entities at the same time (groups of subtransactions), and has most recently been applied to modelling interactions between components of a (distributed) system in [11] and Moschoyiannis has explored the mathematic properties of the vector model [12], [13].

This theory is adopted here to underpin the local coordination required for long-running multi-service transactions in a digital ecosystem [14], [15], [1], and [2]. This chapter is focused on usage of the model for analysis of the pattern behaviour of transactional model (Appendix 1, provides the details of the formal model).

1.1 DIGITAL ECOSYSTEMS: DISTRIBUTED AGENTS

As the kernel of each platform, we have considered a software agent which is responsible for coordinating the participant's business activities (transactions). As we have seen this local agent also archives the information related to these activities (corresponding VPTNs) and improves the general connectivity of the network (its digital ecosystem), and in doing so it contributes to the so-called *network growth* [16]. This is an important aspect when it comes to sustainability, especially in a fully distributed solution. This leads up to the main definition of a digital ecosystem (recall D3.10, chapter 3) which is represented in Fig 1-1, highlighting the fact that there should be no centralized point of command and control in a digital ecosystem.

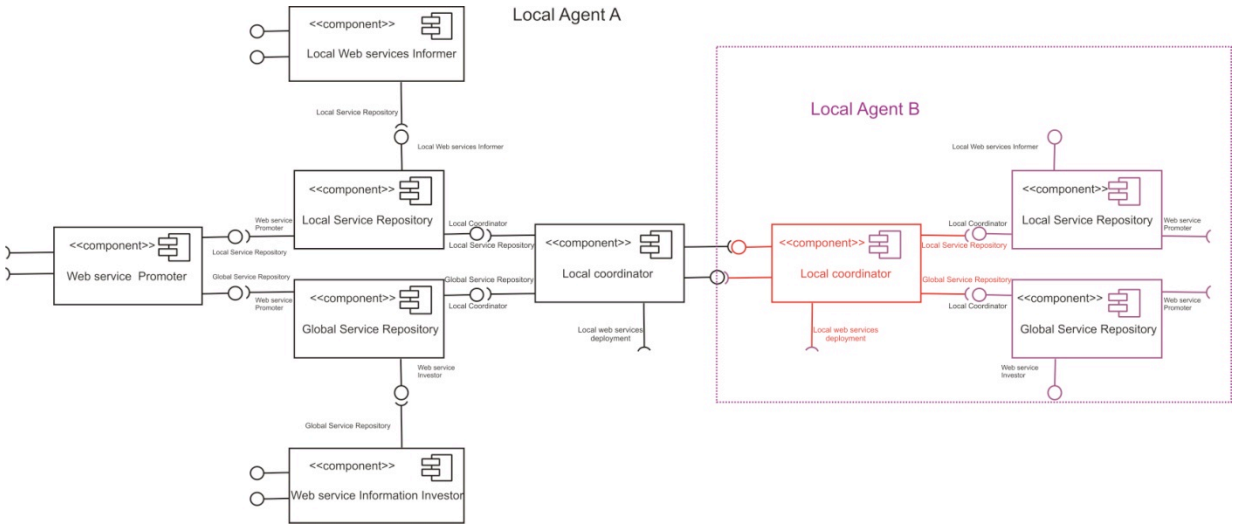


FIG. 1-1 LOCAL AGENT AND COORDINATION[1]

Fig. 1-1 shows the structure of the local software agent of each participant. The '*local coordinator*' component coordinates the service requests to and from the local platform. In other words, it deploys services of the platform, coordinates the transactions and archives their information in the '*local service repository*'. In this way, all participants of a transaction will keep the archived information of the transaction. The '*local web service informer*' component updates any changes of local services in the '*local service repository*' and relevant participants can be notified of the changes through the '*web service promoter*'. The links to other participants will be kept in the '*global service repository*'. Note that at this stage, participants of different VPTNs are connected to each other [in [16] this is called the *birth stage* of the underlying network]. For reducing the possibility of failures and increasing the network stability (as has been explained at D3.10, chapter 7), the network connectivity, i.e. the number of links to other participants, may change. These changes will be done by two components; the '*web service information investor*', for updating new links to the global repository and the '*web service promoter*', for promoting new links to other participants [in [16] this is referred to as the *growth stage*]. Fig 1-2 shows a diagram, where four agents are communicating; Agent A and B are Participating in a Transaction (recall D3.10, chapter 4) and Agent D and C are involved in the network growth for increasing the connectivity (has been explain D3.10, chapter 7).

[1] The image can be found in [14] and the main agent diagram is in D3.10 Fig 4-9

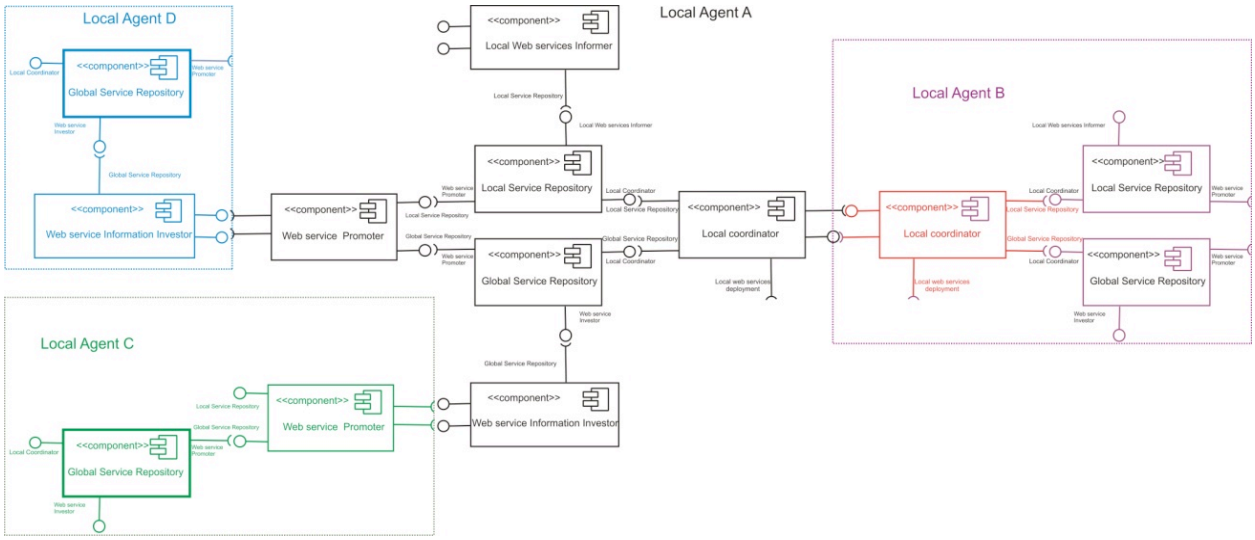


FIG. 1-2 LOCAL AGENTS COMMUNICATION DIAGRAM[2]

1.2 TRANSACTION CONTEXT

In previous chapters [14] 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, 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 [17], we have considered different composition types which allow for various modes of service interaction in our model (recall D3.10, chapter 3 and consistency model has been analysed in D3.10, chapter 4). Fig. 1-3 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 that first appeared in [18]].

[2] The image can be found in [14] and the main agent diagram is in D3.10, Fig 4-9

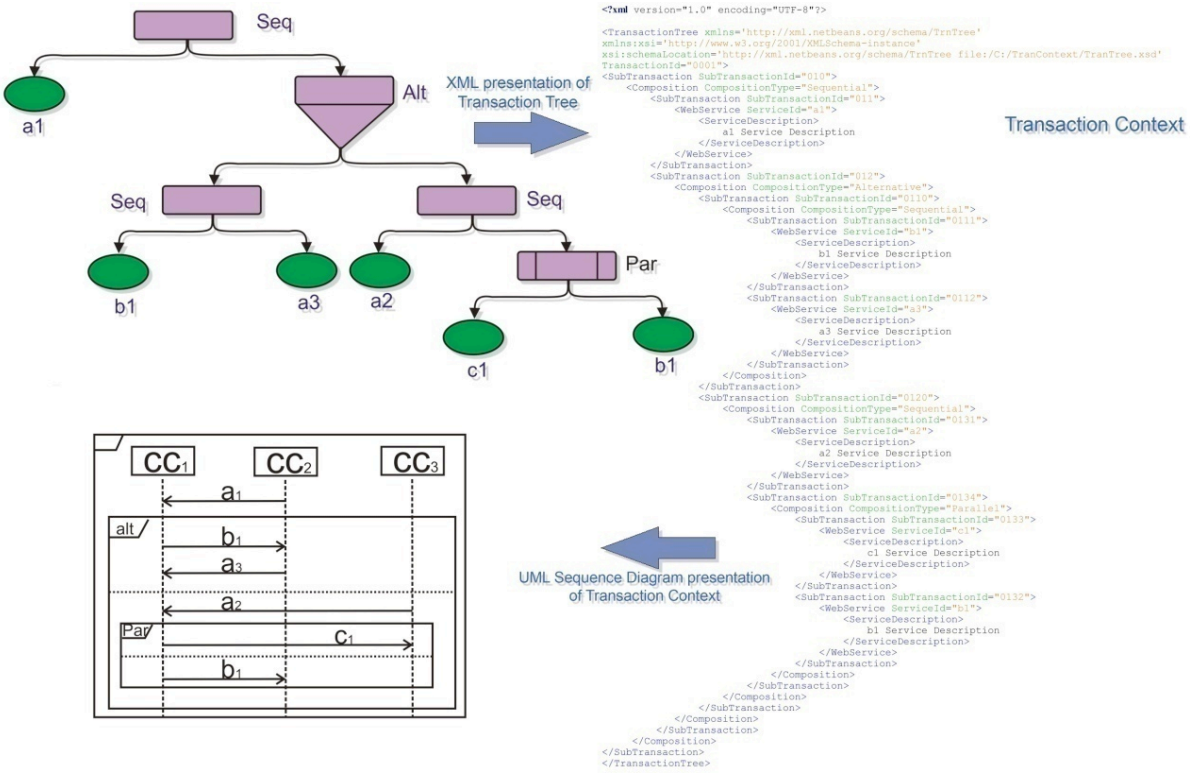


FIG. 1-3 TRANSACTION CONTEXT[3]

We note that the example transaction given in Fig. 1-3 is the same example which can be found in previous chapters [recall chapter 4 & 5, similarly this is explored in [2]]. Primarily we have simplified the communications between coordinators and the abstraction of service deployment without considering acknowledgement or logs. This will be extended in this chapter to illustrate the key ideas as well as to show how these build on previous work in [2].

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 previous chapters [and [19]] we have provided a schema for describing transaction contexts. The derived XML description of the transaction tree of our example (given in Fig. 1-3) can be found and downloaded following [20]. The service interactions implied by a transaction tree can be modelled using a UML interaction diagram. The sequence diagram in Fig. 1-3 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

[3] The XML schema is symbolic and can be checked from D3.10, chapter 4 (section 4.2) or downloaded from http://personal.cs.surrey.ac.uk/personal/pg/A.Razavi/ppna/

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 (recall D3.10, chapter 5) due to network failure. We have seen that a long-running transaction should either complete successfully or not take place at all. So as it has been shown, 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 analyse this behaviour, after explaining the execution script of the transaction.

1.3 MODELLING THE SERVICE DEPLOYMENT OF A TRANSACTION IN A DISTRIBUTED COORDINATION ORCHESTRATION

Based on definition, a distributed system is a collection of individual computing devices that can communicate with each other [21]. Conventionally in a distributed environment, each processor has its own semi-independent agenda, but for various reasons, including sharing of resources, availability, and fault tolerance, processors need to coordinate their actions. This may cause some algorithmic complexity which has been considered as the main difference with the parallel computing [22] and [21].

Digital Ecosystems can be considered as a paradigm for distributing computing, as it promises on one side loose coupling between processors/participants [23] which guarantees the local autonomy [24], and in the other side sustainability and balance [25]. [26] offers the usability of the system in the real world. Generally three challenges have been discussed for distributed systems: '*Asynchrony*', '*Limited local knowledge*' and '*Failures*' [27], [21].

'*Asynchrony*'; The absolute and even relative times at which events take place cannot always be known precisely. This is the main reason for the current complexity in our interaction model (D3.10, chapter 4). In terms of consistency of a transaction, the only measurement for success of a transaction is based on modelling the chains of events and sharing of resources based on access rights in a canonical order and avoidance of behaviours which violate this (recall 4.3 and 4.4). In contrast with conventional models we could not consider a global synchroniser (which needs centralised synchronisation).

In terms of '*Limited local knowledge*', as each computing entity can only be aware of information that it acquires, it has only a local view of the global situation. For obtaining this concept and facilitating the balance of the environment, we have considered two repositories (D3.10, chapter 3) and designed a mechanism for link replication to increase the sustainability (D3.10, chapter 8 and chapter 5). For solving unexpected '*Failures*' in a digital ecosystem, we have designed a distributed recovery model, which even tries to cover unpredicted failures (recall D3.10, 5.4) and provides a mechanism for reducing the cost of recovery (recall D3.10, 5.3).

In the rest of this chapter, we provide an analysis of the patterns of behaviour of such a model and show its behaviours in different situations; from general service deployment, to reaction to failures and forward recovery.

1.3.1 DISTRIBUTED EVENT MODELLING

As the distributed system is asynchronous, there is no fixed upper bound on how long it takes for a message to be delivered or how much time elapses between successive steps of a processor [27], [21], [22]. As a result of this assumption, the modelling of the distributed algorithm should be independent of any particular timing parameter.

Formally a system (or algorithm) consists of *n* processors p_0, ..., p_{n-1}; *i* is the *index* of processor p_i. Each *processor* p_i is modelled as a state machine with state set Q_i. The processor is equivalent to a digital ecosystem participant and identified with a particular node in the topology graph. Furthermore each processor p_i can send message or receive message by doing so it can affect or be affected by other processors. Based on this a *configuration* has been defined; a *configuration* is a vector $C = (q_0, \dots, q_{n-1})$ where q_i is a state of p_i. The states of processor can change this can affect the transmission channel (send messages to other processor) or be affected by the transmission channel (receiving messages from other processor). An *initial configuration* is a vector $(q_0, \dots, q_{n-1})$ such that where q_i is an initial state of p_i; which means each processor is in an initial state. Occurrences in a system are modelled as events, when events can be *computational event* (representing a computational step of processor p_i, when the transitional function is applied to its current accessible state), or delivery event (representing a delivery message from processor p_i to processor p_j).

A well-known concept in such modelling is that of '*execution segment*' [21]; an execution segment $\propto$ of a asynchronous message-passing system is a sequence of the following form:

$C_{0,1}, C_{1,2}, C_{2,3}, \dots$

Where each C_k is a configuration and each $_k$ is an event. If $\propto$ is finite then it must end in a configuration. An execution is an execution segment $C_{0,1}, C_{1,2}, C_{2,3}, \dots$, where C_0 is an initial configuration. A *schedule* (or *history*) has been associated to each execution which is the sequence of events in the execution, that is, $_{1,2,3,\dots}$. Conceptually the ideal situation is to have an *admissible schedule*; that is the result of an *admissible execution.* The admissible execution happens when none of the processors fail, they have an infinite number of computational events and the transmission channel does not fail to send any messages.

As a Digital Ecosystem is a service-oriented environment and insist on loosely coupled binding between participants, the computational events are the execution of services and for reaching an admissible execution, we have focused on our log-based mechanism (recall D3.10, chapter 4). For detecting, avoiding and recovering failures without being involved in an infinite computational model, the lock mechanism has been combined with the consistency model to design a recovery mechanism (recall D3.10, chapter 5). For analysing such environment, we use a customised subset of the conventional distributed event modelling, introduced by Moschoyiannis [2].

1.3.2 TRANSACTION SCRIPT

In Appendix 1, we describe a formal language for pattern behaviour of our 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 of 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. 1-4. 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 $\mu : C \rightarrow \wp(M)$ and require that $\bigcup_{i \in C} \mu(i) \subseteq M$.

As can be seen in Fig. 1-3, 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 [5] 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 $\mu(i)$. Similarly, M^* is the set of all finite sequences formed over the set of actions *M*. 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.

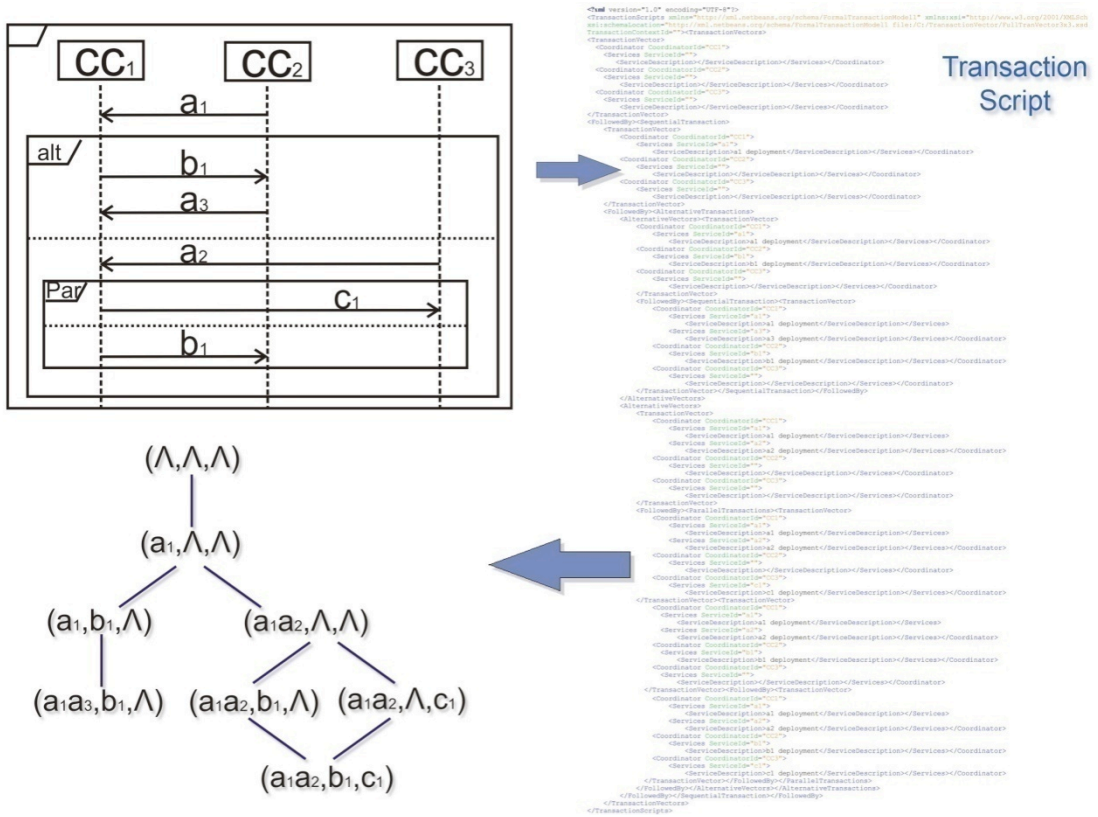


FIG. 1-4 TRANSACTION SCRIPT[4]

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. 1-4, 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.

Column vectors. Let *T* be a transaction and V_T its set of transaction vectors. We define

$$A_T = \{\underline{a} \in V_T \setminus \{\underline{\Lambda}_T\} : l \in L \Rightarrow |\underline{a}(l)| \leq 1\}$$

where | *x* | denotes the length of sequence *x*. We refer to elements of A_T as *column vectors*.

Thus, column vectors are themselves transaction vectors, but have the additional constraint that each of their coordinates is either the empty sequence or a single action. For

[4] The XML schema is symbolic and can be checked from Fig 1-5 or downloaded from http://personal.cs.surrey.ac.uk/personal/pg/A.Razavi/ppna/

example, the vector (*a1*, Λ, Λ) represents the occurrence of an action *a1* on the service associated with the first coordinate.

We have seen that transaction vectors are essentially tuples of sequences. This can be exploited in defining operations on vectors in terms of well-known operations on sequences. Let us establish some notation. If x and z are sequences, we write *x.z* for the concatenation of *x* and *z*. As is well known this operation on sequences is associative with identity Λ, where Λ denotes the empty sequence. We also have a partial order on sequences given by $x \leq z$ if and only if there exists a sequence *y* such that $x.y = z$, and this partial order has a bottom element Λ.

We may now lift these well-known operations on sequences onto transaction vectors. This is done formally in the following definition.

Operations on vectors. Let $\underline{u}, \underline{v} \in V_T$ be transaction vectors, we define

$\underline{u}.\underline{v}$ to be the unique vector $\underline{w}$ such that $\underline{w}(i) = \underline{u}(i).\underline{v}(i)$, for each $i \in C$ (*concatenation*)

$\underline{u} \leq \underline{v}$ iff $\underline{u}(i) \leq \underline{v}(i)$, for each $i \in C$ (*prefix ordering*)

glb($\underline{u},\underline{v}$) to be the vector $\underline{w}$ such that $\underline{w}(i)=\min(\underline{u}(i),\underline{v}(i))$, for each $i \in C$

lub($\underline{u},\underline{v}$) (if it exists) to be the vector $\underline{w}$ such that $\underline{w}(i)=\max(\underline{u}(i),\underline{v}(i))$, for each $i \in C$

if $\underline{u} \leq \underline{v}$, then we define $\underline{v} / \underline{u}$ to be the unique element $\underline{z} \in V_T$ such that $\underline{u}.\underline{z} = \underline{v}$ (*right-cancellation*)

Thus, the operation of concatenation on vectors is defined in terms of the concatenation of sequences appearing on their respective coordinates. For example,

$$(a_1, b_1, \Lambda).(a_3, \Lambda, \Lambda) = (a_1 a_3, b_1, \Lambda)$$

The ordering amongst vectors is defined in terms of the usual prefix ordering operation on sequences appearing on their coordinates. For example, (*a1*, *b2*, Λ) ≤ (*a1a3*, *b2*, Λ) since $a1 \leq a1a3$ and $b2 \leq b2$ and $\Lambda \leq \Lambda$. In other words, the second vector 'wins' on the first coordinate (since it has a sequence of greater length in this coordinate) while the two vectors draw on all other coordinates. It is not hard to see that some vectors will be incomparable. It turns out that such vectors describe either parallel or alternative behaviours of the transaction in question, and this will be further discussed in the following sections.

The operations *glb()* and *lub()* give the greatest lower bound and the least upper bound, respectively of $\underline{u}, \underline{v} \in V_T$, in the usual sense of lattices and domain theory [28]. As we will see, these operations are central to the treatment of concurrency in our approach.

The treatment of concurrency within our formal model of transactions thus takes up on non-interleaving models of concurrency, which introduce additional structure into formal languages in order to describe non-sequential behaviour. The additional structure is given in

terms of an independence relation over action symbols, which describes potential concurrency.

In terms of our notation it is appropriate to say that the independence relation on the set of actions *M* of a transaction equates all, and only those, sequences over $\mu(i)$, for each $i \in C$, which differ in the order of adjacent and independent actions. Note that when the independence relation is empty in the sets $\mu(i)$, for each $i \in C$, no actions can be concurrent in the corresponding sequences $\mu(i)^*$, for each $i \in C$, which amounts to our understanding of sequential transaction processing systems.

Drawing upon the extension of the independence relation ι to *behaviour vectors* in [5], the notion of independence between actions in Mazurkiewicz traces can be readily interpreted into transaction vectors in our approach.

Independence. For $\underline{u}, \underline{v} \in V \subseteq V_T$, we define

$$\underline{u} \; ind \; \underline{v} \quad \Leftrightarrow \forall\, l \in L : \; \underline{u}(i) > \Lambda \Rightarrow \underline{v}(i) = \Lambda$$

This definition says that two transaction vectors are independent if the behaviours they describe concern distinct services (correspond to activation on different leaves of the corresponding transaction tree). This means that the behaviours described by u and v may occur independently.

The additional structure provided by the notion of independence allows to go round the 'diamond' once and end up with the behaviour in which both actions happened concurrently (this can be seen in [2]). The diamond formed in the Hasse diagram depicting the order structure, see for example Figure 1-4, is characteristic of distributed finite lattices, and denotes that the actions involved (b1 on CC2 and c1 on CC3 in our running example) happen concurrently. As we will see in the next section, this also applies to going backwards and allows compensating 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. 1-4, each subsequent action (or concurrent actions) take place in going forward until the transaction as a whole terminates successfully. In [2], we have provided a schema for deriving the XML description of the order structure of a transaction language (Fig 1-5). These so-called transaction scripts describe the interactions between different coordinator components (recall Fig. 1-2) and determine the order in which the underlying services need to be deployed. Fig. 1-4 shows the transaction script generated from the vector-based behavioural description of the transaction in our approach and its XML schema has been shown in Fig 1-5.

Source Schema Design

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Document   : Transaction Script xml Schema 3 coordinators vector
    Created on : 06 March 2009, 16:07
    Author     : A. Razavi
    Description: Prepared for PhD Thesis
-->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://xml.netbeans.org/schema/FormalTransactionModel1"
            xmlns:tns="http://xml.netbeans.org/schema/FormalTransactionModel1"
            elementFormDefault="qualified">
    <xsd:complexType name="Vectors">
        <xsd:sequence>
            <xsd:element name="TransactionVector" type="tns:TransactionVectorType"/>
            <xsd:element name="FollowedBy" minOccurs="0" type="tns:CompositionType"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="CompositionType">
        <xsd:choice>
            <xsd:element name="SequentialTransaction" type="tns:SequentialComposition"/>
            <xsd:element name="AlternativeTransactions" type="tns:AlternativeComposition"/>
            <xsd:element name="ParallelTransactions" type="tns:ParallelComposition"/>
        </xsd:choice>
    </xsd:complexType>
    <xsd:complexType name="SequentialComposition">
        <xsd:sequence>
            <xsd:element name="TransactionVector" type="tns:TransactionVectorType"/>
            <xsd:element name="FollowedBy" minOccurs="0" type="tns:CompositionType"/>
        </xsd:sequence>
    </xsd:complexType>

     <xsd:complexType name="AlternativeComposition">
        <xsd:sequence>
            <xsd:element name="TransactionVector" type="tns:Vectors" minOccurs="2" maxOccurs="3"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="ParallelComposition">
        <xsd:sequence>
            <xsd:element name="TransactionVector" type="tns:TransactionVectorType" minOccurs="2" maxOccurs="3"/>
            <xsd:element name="FollowedBy" type="tns:Vectors"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="TransactionVectorType">
        <xsd:sequence>
            <xsd:element name="Coordinator" type="tns:AccessPoint" maxOccurs="3" minOccurs="3"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="AccessPoint">
        <xsd:sequence>
            <xsd:element name="DeServices" type="tns:Actions" maxOccurs="unbounded" minOccurs="0"/>
            <xsd:element name="Id" type="tns:AcessPointId"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="Actions">
        <xsd:sequence>
            <xsd:element name="ServiceDescription" type="xsd:string"/>
        </xsd:sequence>
        <xsd:attribute name="Id" type="xsd:string" use="required"/>
    </xsd:complexType>
    <xsd:complexType name="TransactionsVectorId">
        <xsd:sequence>
            <xsd:element name="InitiatorId" type="tns:AcessPointId"/>
            <xsd:element name="LocalId" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="TransactionScripts">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="TransactionVectors" type="tns:Vectors"/>
                <xsd:element name="TransactionId" type="tns:TransactionsVectorId"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:simpleType name="AcessPointId">
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
</xsd:schema>
```

FIG. 1-5 TRANSACTION SCRIPT SCHEMA

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 [20].

1.4 FAILURES, RECOVERY AND PATTERN BEHAVIOURS

So far we have described the use of vector languages (full details in appendix 1) in determining the patterns the underlying service compositions should follow in the course of a long-running transaction. In particular, by exploiting the order-theoretic structure of transaction vectors we have shown how all possible interaction scenarios can be captured, and analysed prior to deployment, in determining the set of allowed sequences of actions. In transactional terms, this allows to determine the history of the transaction based on the transaction semantics [29]. In practical scenarios, applications in a transactional environment should offer recoverability and consistency. These two central aspects are addressed in the following sections.

1.4.1 EXECUTION HISTORY

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. 1-6 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. 1-4 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.

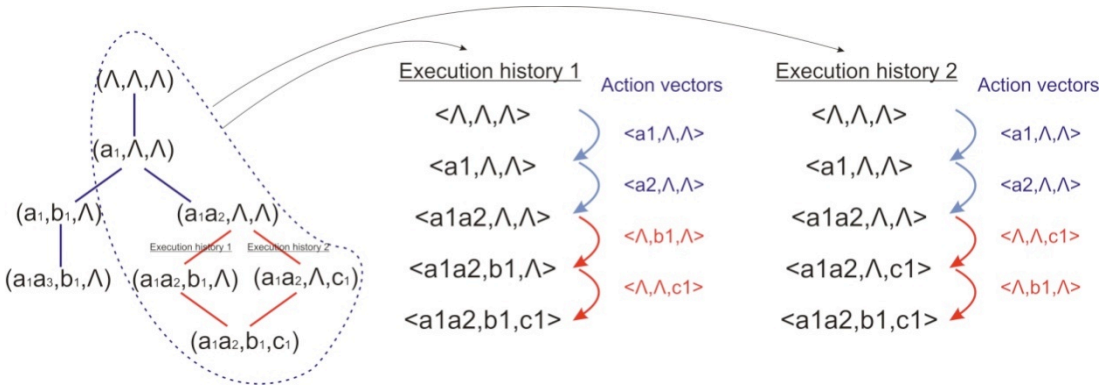


FIG. 1-6 EXECUTION HISTORY

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. 1-6 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. 1-6 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 [30]. 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.

1.4.2 RECOVERY AND ROLLBACK

In this section we examine the formal 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.

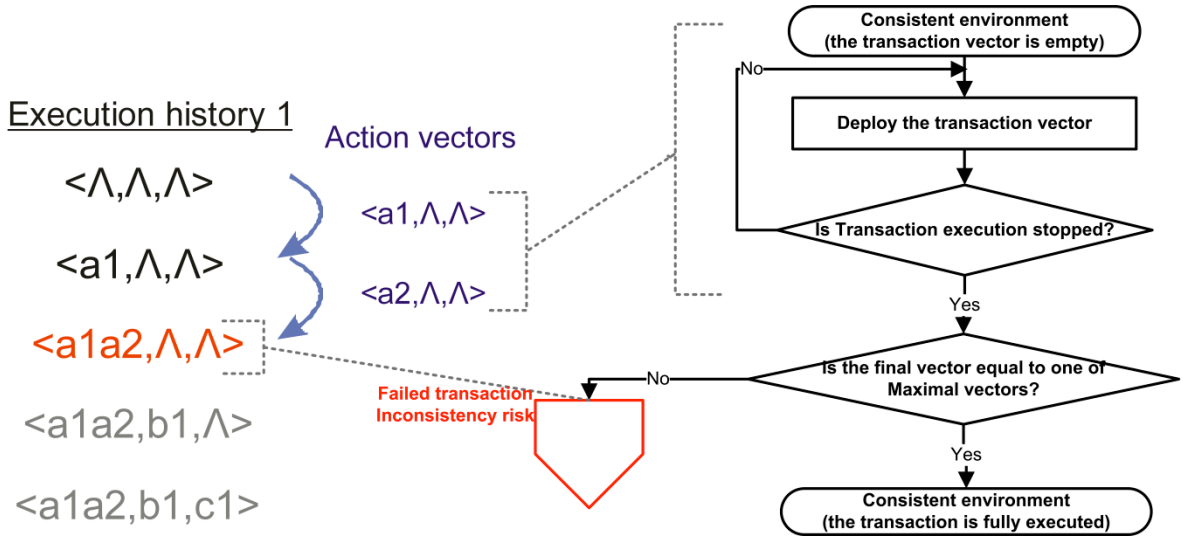


FIG. 1-7 FAILURE IN A TRANSACTION

When the deployment of a transaction stops before it reaches the maximal vector, there is a failure (transaction can not reach the commit and deploys all of its services), this has been shown in Fig 1-7. Based on atomicity, the transaction should be aborted. A cancellation operation on vectors is central to the handling of abortion (compensations) in the vector model. This is done by considering the usual operation of right-cancellation on sequences and then lifting it onto vectors (by applying it coordinate-wise) as follows.

Right-cancellation. Let $\underline{u}$, $\underline{v} \in V$. If $\underline{u} \leq \underline{v}$, then we define $\underline{v}/\underline{u}$ to be the unique element z such than $\underline{u}.\underline{z}=\underline{v}$.

The right-cancellation operator '/' says that if $\underline{u}$ is a transaction vector describing an initial part of the behaviour described by $\underline{v}$ so that $\underline{u} \leq \underline{v}$, then $\underline{v} / \underline{u}$ is the 'continuation' of $\underline{u}$ that extends it to $\underline{v}$. This dictates that what takes a transaction vector and extends it to its successors is an action vector (e.g. service invocation between different participants) of the transaction. It means that vectors are built by a series of concatenations with action vectors, and this is used to model forward actions, and it also means that successive applications of right-cancellations can be used to express the series of compensating actions required whenever some failure occurs during execution.

In other words, our approach uses (coordinate-wise) concatenation of vectors to model the occurrence of an action by

$u.\alpha=v$

and, instead of introducing a separate notation and associated semantics for cancelling forward actions, we use right-cancellation to express the compensating action needed at each point in returning to the initial configuration.

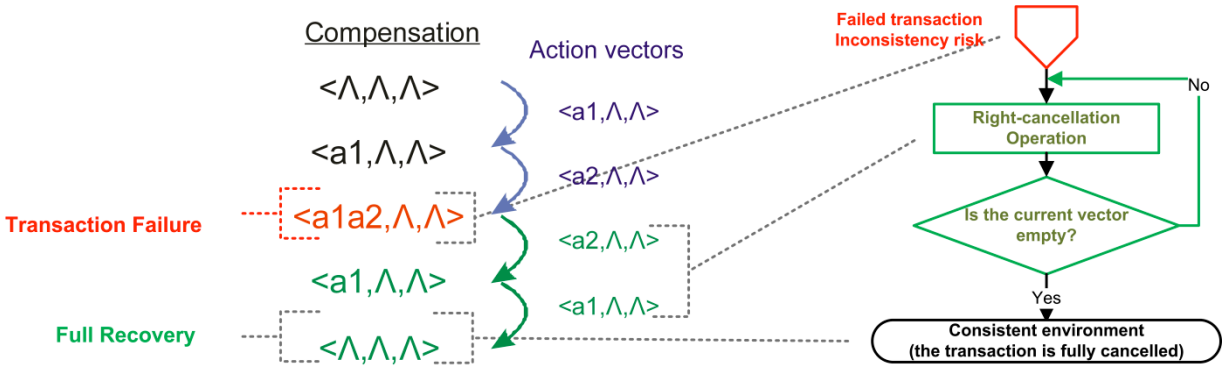


FIG. 1-8 TRANSACTION RECOVERY

Fig. 1-8 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. This is represented by the empty vector. In returning to empty vector, all previously successful forward actions need to be compensated for. This is done by successive applications of the '/' operation given earlier.

The first application of right-cancellation moves the transaction back to (a1,Λ, Λ) and the next to (Λ, Λ, Λ). Hence, the system is returned to the empty vector which is now the (only) consistent state. In this case we talk about full recovery of the transaction. It might be worth noting that in the case where failure happened after (a1a2, b1, c1) our approach would allow for the concurrent forward actions b1 and c1 to be compensated concurrently. This is because the vector (a1a2, b1, c1) has two immediate predecessors, and hence the independent action vectors α1 = (Λ, b1, Λ) and α2 = (Λ, Λ, c1) that extended them to (a1a2, b1, c1) would be called to be compensated for concurrently. In other words,

$(\underline{v}/\ \underline{\alpha_1})/\ \underline{\alpha_2} = (a1a2, \Lambda, c1)/\ \underline{\alpha_2} = (a1a2, \Lambda, \Lambda)$

And also

$(\underline{v}/\ \underline{\alpha_2})/\ \underline{\alpha_1} = (a1a2, b1, \Lambda)/\ \underline{\alpha_1} = (a1a2, \Lambda, \Lambda)$

1.4.3 Forward Recovery

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 concatenations in accordance with the allowed sequence of actions in that execution history, as described in Section 4.2.

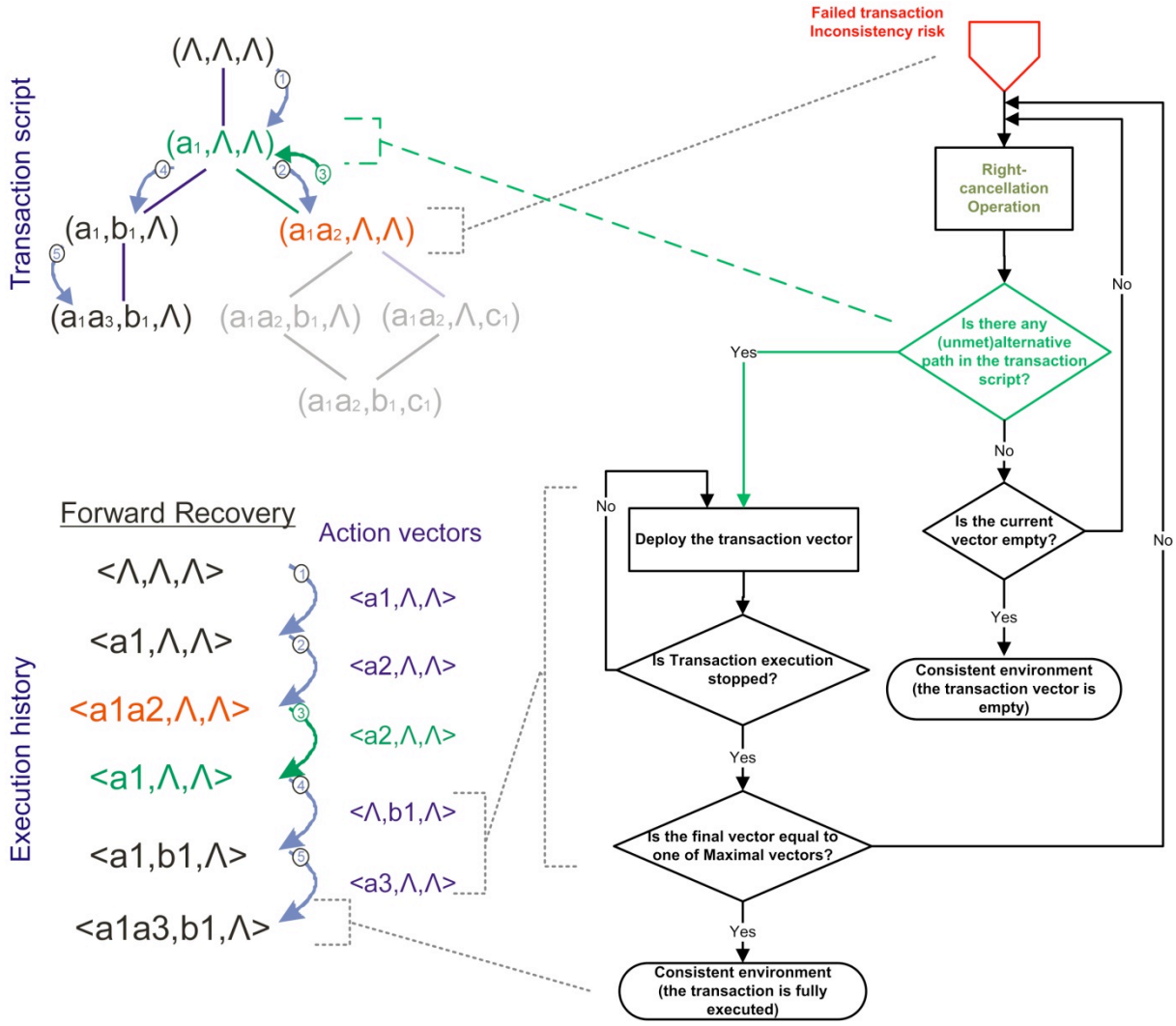


FIG. 1-9 FORWARD RECOVERY

As shown in Fig. 1-9, 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.

1.5 RESULTS AND COMPARISON

In this chapter, we have described a formal model for analysing patterns of behaviour of Digital Ecosystem long-running transactions. The tuples-based behavioural description allows to capture the sequences of forward and compensating actions of each participant and at every stage in the course of execution of a transaction. Our formal approach to transaction semantics drives the recovery mechanism necessary to deal with failures during transaction deployment in a way that asserts transaction consistency.

On one hand, in contrast with conventional pattern behaviour analysis such as BAPC or BACC (D3.10, chapter 3), this formal model for analysing patterns of behaviour of Digital Ecosystem long-running transactions, shows the behaviour of distributed coordinators of transaction model, rather than a linear analysis based on 2 phase commit protocol. However, it proves the Transaction model proposed in D3.10, chapter 4 and 5 models a loosely coupled interaction of participants (because there is not any scenario for relying on local states of any services of participants) and as a result does not suffer violating the local autonomy of participants.

On the other hand, unlike other approaches to transaction semantics, such as [30], [31], concurrent actions are handled via a notion of independence and as a result concurrent forward actions are compensated for concurrently in our approach. We have shown how transaction languages capture the history of a transaction and this not only allows for the full recovery of a transaction in case of a failure, but also to attempt forward recovery wherever possible and avoid the sometimes costly abortion of the whole transaction. We have also highlighted the implementation aspects of the theory presented and further details can be downloaded following [20].

We have been concerned with transactions that involve the execution of our transaction model (D3.10, chapter 4) and have focused on the orderings of the underlying service interactions. The specifications of this ordering, can be found in the variety of service compositions in a Digital Ecosystem (recall D3.10, chapter 3 and 4). Another aspect concerns the translation of business requirements into the transaction tree and the UML model, which are currently our starting point. Preliminary analysis shows that the use of SBVR could be harnessed in expressing business logic, specifically in terms of (orderings of) the deployment of services and their resources. This is another possible direction for future work as it would make the transactional framework accessible to a wider audience and particularly smaller businesses.

Reference:

[1] 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.

[2] 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.
[3] M.W. Shields, *Semantics of parallelism*, Springer New York, 1997.
[4] M.W. Shields, “Concurrent machines,” *The Computer Journal*, vol. 28, 1985, pp. 449-465.
[5] M.W. Shields and U.O.N.U.T.C. Laboratory, *Adequate path expressions*, Springer, 1979.
[6] A. Mazurkiewicz, “Basic notions of trace theory, in de Bakker, de Roever and Rozenberg (eds.), Linear Time, Branching Time and Partial Orders in Logics and Models for Concurrency,” *Springer Lecture Notes in CS*, vol. 354, 1988, pp. 285-363.
[7] A. Mazurkiewicz, “Concurrent Program Schemes and their Interpretation. Arhus University,” *Computer Science Department, DAIMI PB-78 (July 1977)*, 1977.
[8] M. Nielsen, G.D. Plotkin, and G. Winskel, “Petri nets, event structures and domains,” *Theoretical Computer Science*, vol. 13, 1981, pp. 85–108.
[9] R. Milner, *A calculus of communicating systems, volume 92 of Lecture Notes in Computer Science*, Springer-Verlag, 1980.
[10] C.A.R. Hoare, *Communicating Sequential Processes.*, Prentice Hall, 1985.
[11] S. Moschoyiannis, M.W. Shields, and P.J. Krause, “Modelling Component Behaviour with Concurrent Automata,” *Electronic Notes in Theoretical Computer Science*, vol. 141, 2005, pp. 199-220.
[12] S. Moschoyiannis, “Specification and Analysis of Component-Based Software in a True-Concurrent Setting,” University of Surrey, 2005.
[13] S. Moschoyiannis, “A set-theoretic framework for component composition,” *Fundamenta Informaticae*, vol. 59, 2004, pp. 373-396.
[14] A. Razavi, S. Moschoyiannis, and P. Krause, “Concurrency Control and Recovery Management for Open e-Business Transactions,” *Proc. WoTUG Communicating Process Architectures (CPA 2007)*, 2007, pp. 267–285.
[15] A. Razavi, P. Malone, S. Moschoyiannis, B. Jennings, and P. Krause, “A Distributed Transaction and Accounting Model for Digital Ecosystem Composed Services,” *Digital EcoSystems and Technologies Conference, 2007. DEST '07. Inaugural IEEE-IES*, IEEE Computer Society, 2007, pp. 63-66.
[16] A.R. Razavi, S. Moschoyiannis, and P. Krause, “Report on formal analysis of autopoietic P2P network, together with predictions of performance,” 2007.
[17] 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.
[18] 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.
[19] A.R. Razavi, “DIGITAL ECOSYSTEMS, A Distributed Service Oriented Approach for Business Transactions,” PhD Thesis, University of Surrey, 2009.
[20] A.R. Razavi and S. Moschoyiannis, “FESCA WORKSHOP 2009 xml Sources - http://www.computing.surrey.ac.uk/personal/st/S.Moschoyiannis/trnscripts/,” 2008.
[21] H. Attiya and J. Welch, *Distributed Computing: Fundamentals, Simulations and Advanced Topics*, WileyBlackwell, 2004.
[22] A.D. Kshemkalyani and M. Singhal, *Distributed Computing: Principles, Algorithms, and Systems*, Cambridge University Press, 2008.

[23] E. Chang and M. West, “Digital Ecosystems and comparison to existing collaboration environment,” *WSEAS Transactions on Environment and development*, vol. 2, 2006, pp. 1396-1404.
[24] 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.
[25] 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.
[26] H. Boley and E. Chang, “Digital Ecosystems: Principles and Semantics,” 2007, pp. 398-403.
[27] A.S. Tanenbaum and M.V. Steen, *Distributed Systems: Principles and Paradigms*, Pearson Education, 2008.
[28] B.A. Davey and H.A. Priestley, *Introduction to Lattices and Order*, Cambridge University Press, 1990.
[29] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, 1993.
[30] 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.
[31] J. Gray, *Benchmark handbook: for database and transaction processing systems*, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1992.

2 INTEGRATION OF FLYPEER (IPTI)

This chapter reports the integration of the transaction coordination into the Flypeer P2P network. It is a somewhat technical documentation of the code base, and is mostly of value from this perspective. It can be skipped for those not wishing to explore the code base.

2.1 THE SERVICE REPOSITORY

Publishing of services in JXTA can be made through the publication of its advertisement. The advertisement is a XML file that holds data required by JXTA and added by Flypeer like: service name, group, peer name and its WSDL.

In this way, to invoke a service you must have its advertisement. Therefore the advertisement has a transient state according to the network topology and moment when the peer is connected, consequently the advertisement has a limited time of life.

However, data added by Flypeer do not change often, allowing the caching of data into a Service Repository.

So, in the second time that a service is called by the same peer, there is no need to look for its advertisement until the moment that it will be really called. With cached data, each new invocation of a different service becomes the “wiser” peer, and consequently able to start transactions faster than before.

Making an analogy, cached data is almost like data that you have stored about a friend. You probably know its name, physical characteristics and how to find him, but don’t know where he is now.

2.2 HOW THE MECHANISM OF PARAMETERS WAS IMPLEMENTED

Each service published in Flypeer must have a WSDL File (Web Service Description Language). Into this file are described with parameters the service request and what it returns.

```
<wsdl:types>
    <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified"
            targetNamespace="http://BA-FLIGHT-BOOKING.ws.services.flypeer.ipti.org.br">

        <xs:element name="bookFlight">
            <xs:complexType>
                <xs:sequence>
                    <xs:element minOccurs="0" name="flightTarget" nillable="true" type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="bookFlightResponse">
            <xs:complexType>
                <xs:sequence>
                    <xs:element minOccurs="0" name="return" nillable="true" type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:schema>
</wsdl:types>

<wsdl:message name="flightTarget">
    <wsdl:part name="flightTarget" element="bookFlight"/>
</wsdl:message>

<wsdl:message name="bookedDate">
    <wsdl:part name="bookedDate" element="bookFlightResponse"/>
</wsdl:message>

<wsdl:portType name="BABooking">
    <wsdl:operation name="bookFlight">
        <wsdl:input message="flightTarget"/>
        <wsdl:output message="bookedDate"/>
    </wsdl:operation>
</wsdl:portType>
```

In the following excerpt taken from the WSDL of the service BA-FLIGHT-BOOKING we define this service expects a string parameter called ”flightTarget” and return also a String value.

There are no limits of parameters that one service can receive but its return must be just one single value.

All parameters and return types should be primitive types or literals (String).

This limitation exists because the support of complex types still was not implemented.

When the transaction is started, Flypeer get all WSDL files of the participant services of the transaction and verify by name and type if all parameters were provided. If there is any parameter that was not passed, the Flypeer stop the transaction execution an call the method processFail of the FlypeerResponseListener registered passing the respective exception otherwise starts the transaction.

2.3 Mapping Service Dependencies

There is just one case where not all parameters expected will be passed in the start of the transaction. It happens when the services need of data exchange to be able process it and produce a result. For instance, if a service C needs of data generated from service A and B, we say that C depends of A and B. The code bellow maps this relationship from java source code:

```
view plain  copy to clipboard  print  ?
01. Service serviceA = new Service("SERVICE_A", "myGroup");
02. Service serviceB = new Service("SERVICE_B", "myGroup");
03. Service serviceC = new Service("SERVICE_C", "myGroup");
04.
05. //Map the dependencies
06. serviceC.addDependency("firstParamName", serviceA);
07. serviceC.addDependency("secondParamName", serviceB);
```

In the case above the first three lines create the services participants of the transaction, and in the following lines add A and B as dependencies of the service C.

The service C expect two parameters, “firstParamName” set as the result of the service A and “secondParamName” set as the result of the service B.

There is another type of mapping were we set what will be de result of the transaction, in other words, which values will be returned as result of the transaction execution. This type of mapping is done by the TransactionFlow class:

```
view plain  copy to clipboard  print  ?
01. TransactionFlow txFlow = new TransactionFlow();
02. txFlow.setGenericFlow(anyFlow);
03. txFlow.addTransactionResult("result", serviceC);
```

The case above, the result of the transaction is mapped as the result of the Service C execution and stored with the ”result” name. A transaction can set zero or more parameters as result of the transaction.

When the transaction is started and the Flypeer realize that there are missing parameters, it looks for other mappings of the dependencies to check if the parameter was really not informed or will be provided by other services.

2.4 The Transaction Initiator

It is responsible by execute and manage all initialization procedures of the transaction. It is who warn the participant peers that a new transaction has been created.

The TransactionInitiator is called right after the initiator peer(who have created the transaction) call the methods startTransaction of the TransactionInitiator class.

A unique id is generated to identify the transaction and Transaction Context over the network and then both are stored in a local repository.

The first task of the TransactionInitiator is called “Service Verification” and consists in analyzing the TransactionContext and retrieving all participant services. After that, it search

those into the local service repository, in the case of any service be not found, the TransactionInitiator try to find the service advertisement in the network. If the service still cannot be found, the method proccessFail of the registered listener is called and receives the related exception as argument. On the other hand, if all services were found, the second task is started.

The second task is the parameter validation previously mentioned, to validate the parameters all service WSDL should be available. This is why it is done as the second task.

In the third phase, the list of peer names of the service owners participants of the transaction are retrieved and a TRANSACTION_CTX is sent for each peer that contain its name in the list. The TRANSACTION_CTX holds the Transaction Context and the transaction id, both should be known by all participants.

When a participant peer receives the message and stores all data in the message into a local repository, it starts the flow of Service Verification previously mentioned. Then returns a RECEIVE_ACK_TRANS_CONTEXT message to the Initiator Peer. This message means that the TransactionContext has been successfully received.

The Initiator Peer wait for all participant peers to respond with a RECEIVE_ACK_TRANS_CONTEXT message and then init the last phase.

In the last phase the transaction initiator analyzes the transaction context retrieving and storing in a local repository the list of services that should be called according to the service composition. After that, it calls all the services in the list passing the following parameters:

- TRANS_CONTEXT_ID: The transaction identification.
- CURRENT_SUBTRANSACTION_ID:
- LAST_SUBTRANSACTION_ID: Null for this case because no services has been executed before.
- PARAMETERS: The Map of parameters, this map contains all parameters defined In the start of the transaction and the result generated by execution of the participant services.

Finally, the transaction initiator concludes its execution.

2.5 The Transaction Coordinator

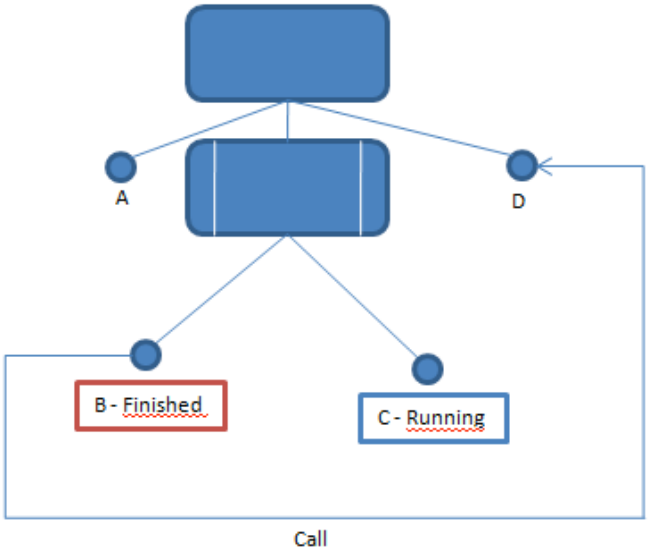
The coordinator in the peer where it is running is responsible by manage the service access and control the transaction flow. Each peer has just one Coordinator that intercepts all calls to the services provided by its peer.

When the coordinator intercepts a call of a service the following parameters are retrieved:

- TRANS_CONTEXT_ID: The transaction identification.
- CURRENT_SUBTRANSACTION_ID: The subtransaction id of the service that will be executed in this peer.
- LAST_SUBTRANSACTION_ID: The subtransaction Id of the service who called it.

- PARAMETERS: The Map of parameters.

Then the coordinator stores the LAST_SUBTRANSACTION_ID and through the TRANS_CONTEXT_ID retrieves its transaction context. With the transaction context the coordinator is able to verify if the service called can be executed. This verification is needed due to the parallel composition.



In the figure above we have a transaction context mapped graphically. Assuming that Service B will finish first then C, the coordinator of the service B will call Service D but D also depends of C, then the coordinator of the service D waits until the Service D is Called by C.

When there are no more dependencies to wait for, the coordinator starts the preparation to execute the requested service. In this preparation, all maps of parameters received are merged and the duplications removed. In this way, the result is a single Map of parameters whence the parameters that the service depends on is extracted.

The next step is getting the registered service listener and executes it with the extracted parameters, the result of this execution is stored by the coordinator into the Map of Parameters.

A new analyses on the Transaction Context is made, this time the data retrieved are a list of next services that should be executed when the current ends.

On the one hand the list of services may be null, it means that the current service was the last service of the transaction (or one of the lasts for parallel compositions) and should call the peer initiator to process the result of the transaction until this point. To do this, the coordinator sends a TRANSACTION_RESPONSE message that contains the following parameters:

- TRANS_CONTEXT_ID: The transaction identification.
- LAST_SUBTRANSACTION_ID: The subtransaction id of the service that was execute in this peer.
- PARAMETERS: The Map of parameters.

This message means for the initiator peer that one of the "possible ends" of the transaction has been reached. After send this message, the coordinator conclude its execution.

On the other hand if the list of services is not null the coordinator stores the list in a local repository and call each service passing the following parameters:

- TRANS_CONTEXT_ID: The transaction identification.
- CURRENT_SUBTRANSACTION_ID: The subtransaction id of the service that will called..
- LAST_SUBTRANSACTION_ID: The subtransaction id of the service that was execute in this peer.
- PARAMETERS: The Map of parameters.

Finally, when all messages have been sent, the coordinator concludes its execution.

2.6 THE RESPONSETRANSINFRASERVICE

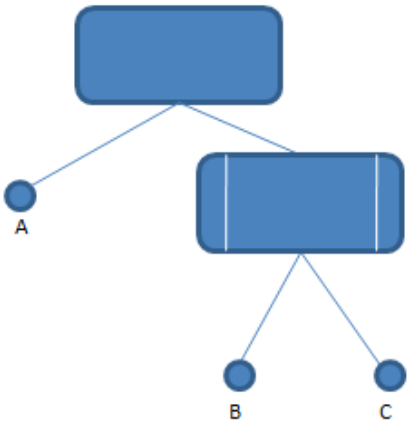
Running in the Peer Initiator the ResponseTransInfraService is who receive the message TRANSACTION_RESPONSE previously mentioned, it

is the responsible to filter the data produced by the transaction to generate the result that will be returned to the transaction listener.

When a message is received, the ResponseTransInfraService retrieves and stores the following parameters.

- TRANS_CONTEXT_ID: The transaction identification.
- PARAMETERS: The subTransaction id of the service that was execute in this peer.
- LAST_SUBTRANSACTION_ID: The Map of parameters.

From the TRANS_CONTEXT_ID the ResponseTransInfraService can get the transaction context stored by the TransactionInitiator. A new analysis over the transaction context is made to verify if there is some service that should be waiting for.



In the figure above, we have two parallel services B and C, when B finish its execution and call the Initiator Peer the service C still is running. In that case the ResponseTransInfraService wait for a call from C to finally produce a result to return and conclude the transaction.

If there is no service to wait for, the ResponseTransInfraService get all Map of parameters received and merge them removing any duplication.

Then retrieve from the transaction context all parameters expected by the responseListener, The next step is call in a new thread the registered response listener passing the retrieved parameters.

In the end, a RECEIVE_COMMIT_TRANS message is sent to all participant peers of the transaction. This message notify that the transaction has been committed and all resources allocated for its can be released.

2.7 How the mechanism of rollback was implemented

When the execution of a service throw an exception, the coordinator start the rollback flow, the stack of the detected fail is added into a map of exceptions and then the service rollback listener is called.

After that, the coordinator retrieve the stored list of last services and send a RECEIVE_FAIL_NOTIFICATION message for the coordinator of the peers that hosts the services of the list with the following parameters:

- CURRENT_SUBTRANSACTION_ID – The subTransaction id of the service to be rolled back;
- TRANS_CONTEXT_ID - The transaction identification.
- LAST_SUBTRANSACTION_ID – The subTransaction id of the last service that was rolled back;
- EXCEPTIONS – Map of exceptions that have been thrown by the services during the execution of the transaction;

This type of message notifies the peer who receives it that the execution of this ramification of the tree failed and the rollback of the referred service should be done.

When message arrives, the coordinator retrieve the transaction and subtransaction ids of the services that should be rolled back and get the rollback listener of the service, then invoke the listener passing the same parameters passed when the services was executed.

So the list of last services is restored and a RECEIVE_FAIL_NOTIFICATION message is sent for each coordinator of the services.

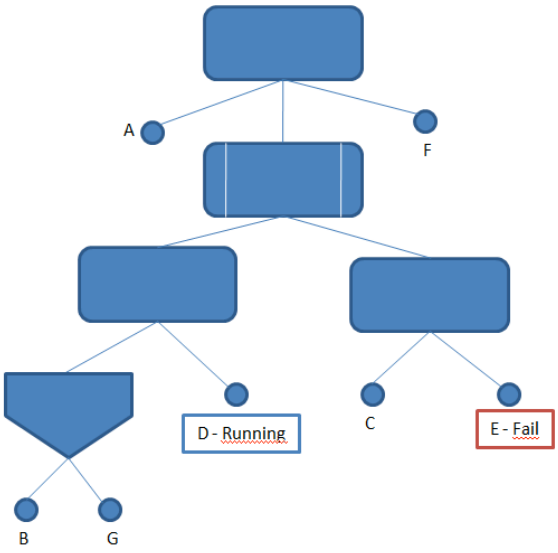
This process is repeated until the coordinator of the initiator peer receives a message of the same type.

When it happens the coordinator retrieve the transaction listener and verify if there are services that still not called its. If so, the coordinator enter in a sleep state till the missing messages arrives.

In the moment that there is no more missing messages, the coordinator assume that all services were rolled back, then merge the map of exceptions received removing any duplication, get the transaction listener registered and call in a new thread the method processFail passing the map of exceptions as parameter.

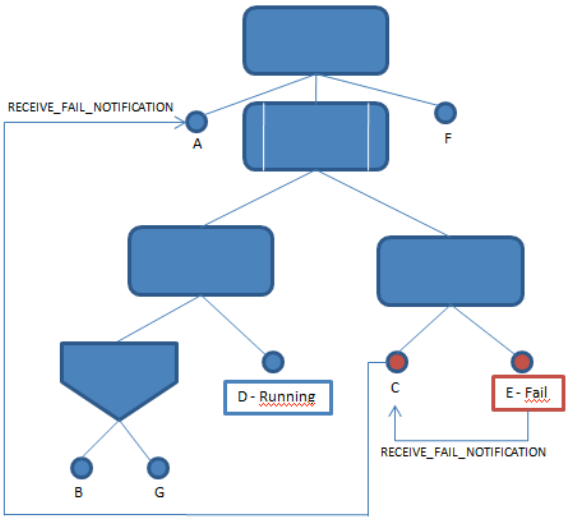
2.8 Rollback of Parallel Compositions

When there is a parallel composition in the Transaction Context, the rollback flow has a peculiarity:



In the case bellow, when the service E fails, the service B and D continue unaware of that, and in the B-D ramification the transaction continues executing normally.

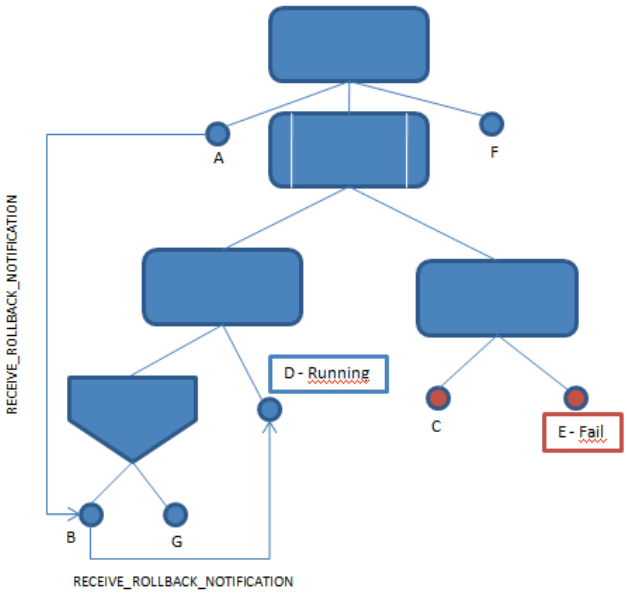
So, to stop this execution the Flypeer find the common node between them and send a notification of rollback. To do this, the coordinator of the service E execute its rollback and call the coordinator of C, the coordinator of C execute its rollback and call the service A that is the common node between the ramification B-D and C-E.



The coordinator of A restore the list of next services [B,C] and verify that should notify the other ramification[B] to stop its execution and start the rollback, then send a RECEIVE_ROLLBACK_NOTICATION message that notify the receiver to start the rollback flow.

Into this message we can find a parameter called “ROLLBACK_TARGET” , this parameter says till which subtransaction the rollback should be done even if there is an alternative service in the mid of the track.

When the coordinator of B receives the notification, check for the status of the transaction at this node and realize that the service B has already finished and all the messages to the next services has been sent. In this case the coordinator forwards the rollback notification to the next services.

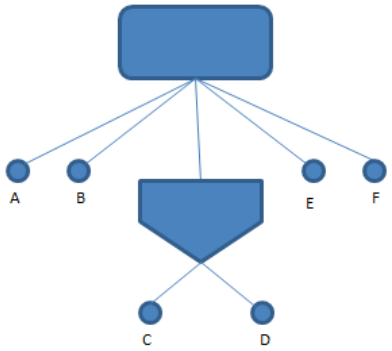


When the message finally arrives, the coordinator of the service D is still executing. The coordinator waits the service finish and immediately executes its rollback.

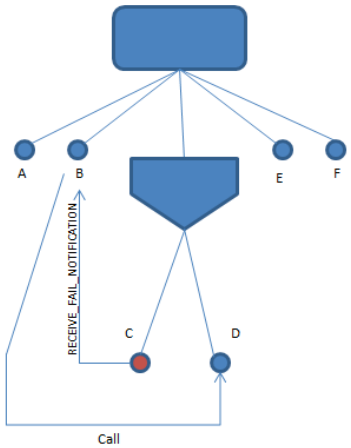
From here, the rollback flow goes back to the normal.

2.9 Forward Recovery

Given the uncertain nature of peer-to-peer networks where there is no guarantees that the services and peers will be always available, the forward recovery was added into the Flypeer. Through the forward recovery the transaction can recover a fail and continue executing instead of execute the whole rollback.



In the example above, when the Service C fails, the coordinator executes its rollback and sends a RECEIVE_ROLLBACK_NOTIFICATION message to the previous service [B]. When the message arrives in the coordinator of the service B, the rollback of the service is not executed because the coordinator checks that there is an alternative service [D] to the service C.



Then the service D is called and the flow of the transaction is back to the normal.

In the previous case, if any service fails again, the transaction will be rolled back because there are no more alternative services to be executed.

3 INTERFACES TO DIGITAL ECOSYSTEMS (NAICA)

The features of a system can be advanced and innovative, but if end users will not have simple interfaces in order to interact with the same system, probably these innovative features will be ignored and end users will look for other systems. Complex interfaces are frequently perceived as barrier by end users, and they are one of the main reasons for failure of modern information systems.

Among the features available in digital ecosystem implementation for the OPAALS project, through *Flypeer* peer-to-peer infrastructure, developed keep in mind a standard SOA (Service-oriented architecture) architecture, one of the key features is surely the ability to establish long running transactions in a distributed and loosely-coupled manner among the nodes of the p2p network.

Thanks to these transactions it is possible to define the so-called *Virtual Private Networks* (VPTNs) among the nodes involved in the transaction and these VPTNs are the basis on which is possible to build the *Dynamic Virtual Super Peers* (DVSPs), perhaps the distinctive feature of the P2P model developed by the University of Surrey for OPAALS and the main goal in deploying the infrastructure.

In order to taking advantage of all P2P benefits of Flypeer must be easier for businesses to create, deploy, execute services and must be simple to create and execute transactions.

In this document we will describe how to specify a business transaction that involves several services provided from different small and medium enterprises (SMEs) in a digital ecosystem, that is realized by a p2p network based upon Flypeer, by means of a natural language based description (*Semantic for Business Vocabulary and Rule* - SBVR - has been used) in order to minimize the programming effort. The discovery of services involved in these transactions has been done via SBVR vocabularies, which are semantically annotated to the WSDL description of these services.

3.1 INTRODUCTION

Such business transactions, also called long-lived or long-running transactions because of their duration, are the result of interactions between participants of a network that want to reach a specific target. These long-running transactions between SMEs in a digital ecosystem for business can be either a simple usage of a web service (rarely in B2B relationships) or a mixture of different levels of composition of several services from various service providers.

In OPAALS vision, the digital ecosystem is based on a peer-to-peer network, so in this context each business transaction is the result of peer-to-peer interactions. In such dynamic and evolutionary network, these transactions must take place in a distributed and loosely-coupled manner: for this reason an innovative transaction model, which ensure a distributed transaction management, has been developed by OPAALS researchers (see [1], [2], [3], [4] and, above all, [5]).

Flypeer infrastructure [6] realizes the OPAALS p2p network requirements. Flypeer also implements the previous mentioned transaction model.

In order to specify a long running transaction between nodes in a Flypeer-based p2p network, we must define the so called Transaction Context. In the Transaction Context is delineated the context in which a transaction is executed, in terms of parties involved, services they provide and the specific way in which they should be deployed. This describes how services are composed to obtain the overall functionality.

It is possible to obtain the Transaction Context programmatically or by editing a configuration file. However, both these ways require a minimum amount of programming expertise that business people typically don't have. For this reason a user interface to simplify this task has been thought.

Starting from the original idea [3] to adopt the *REpresentional State Transfer* (RESTful*)* [7] [8] architecture to provide access to resources in a Flypeer-based network web services, *Semantic Business Vocabulary and Business Rules* (SBVR) [9] was chosen as service description language for the purpose of declarative service composition because SBVR provides sufficient capabilities for integration with minimal human intervention.

Due to novelty of the distributed transaction model for a RESTful architecture, there is not yet a working implementation that supports this transaction model, and even Flypeer still does not support RESTful transactions. Moreover, last not least, the absence of a distributed repository to store REST service descriptions (something like UDDI in a standard WS-* architecture) represent a problem. However, similarly at original idea just described, we decided to keep SBVR as service description language to add semantics to them, because of the potential benefits SBVR can provide realizing simpler human-machine interactions, but we move towards a more standard

WS-* SOA architecture, using therefore also WSDL.

SBVR has been used to provide a semantic description of services. This description was linked by means of semantic annotation mechanism to a WSDL syntactic description of the same service so that the system works properly. While the syntactic descriptions provide information about the structure of input and output messages of an interface and about *how* to invoke the service, semantics are helpful to describe *what* a Web service actual does.

With this annotated SBVR description we have laid the basis for the foundation of a declarative semantic-based composition of web services in a standard WS-* architecture leveraging on SBVR, because SBVR can provide integration advantages when used as a language to express both requests on these services and composition services.

In this context and based on elements seen so far we have developed the ***SBVR Transaction Editor***, a Flypeer interface that facilitates the composition of web services, starting with a description of the composition expressed in SBVR, that is more similar and close to the natural language used by business people.

The result of this computation will be a configuration file, the *TransactionContext.xml*, which is used by Flypeer to instantiate and run a transaction (another way to instantiate a transaction in Flypeer, as previously mentioned, is programmatically). Using simple interfaces for writing complex transactions could, in fact, meet the ability of inexpert users, allowing them to create and publish advanced services composing available web services, even without being web services developer. This can lead to further development and a

more rapid enrichment of the digital ecosystem in terms of features offered to participants.

Other information, source code and documentation about a prototypical version of the ***SBVR Transaction Editor*** can be found here: http://kenai.com/projects/sbvrtxeditor

Afterwards, in section 3.2, will be shortly described transactions in Digital Ecosystem, the underpinning OPAALS transaction model and the types of service compositions supported by this model. Then an overview of Flypeer available implementation for these features will be provided, in order to know what service composition types are currently at disposal. In section 3.3 we will see how basic elements of transactions, i.e. services, can be discovered and called by means of WSDL description and how we make use of semantic annotations and SBVR vocabularies. Information in 3.2 and 3.3 sections are the starting point for the description of implementation and behaviour of the *SBVR Transaction Editor* component, given in section 3.4. After, in section 3.5, a short description of the output of the developed component will be provided. In the end, conclusions and future perspectives will be outlined in section 3.6.

3.2 Transactions and Services compositions in Flypeer

In a Digital Ecosystem, SMEs offering their products as a service that can be consumed or used by other SMEs either in isolation or as part of a chain of services in a larger business activity, typically a long-running business transaction, in order to meet particular needs of the various partners. The mix and match of services, often supplied from different providers, however does imply that these can be put together or composed in various modes depending on the specific business scenario being executed.

In addition, OPAALS Digital Ecosystem has a set of distinctive characteristics that made it unique and that can be summarized telling this is realized as an *Autopoietic P2P Network*. With Autopoietic we underline its dynamic topology support, given by self-organization behaviour and by a continuous adaptation to usage and environment conditions. It is, moreover, based on upon a p2p network that must satisfy the stringent requirements of not have a *single-point-of-failure*. This network of peers (each peer is a SMEs) has been thought as a *loosely coupled service-oriented* environment. The loosely-coupling requirement, necessary for such changing network, forced the development of a new interaction model between peers. Another distinctive feature of OPAALS network is its innovative *distributed transaction model*, necessary to execute long-running business transactions coordinating the participant's business activities and to manage rollback or *recovery mechanisms* in case of transaction failure in such ever-changing network. According to this new transaction model, coordination must take place in a fully distributed manner, to avoid the single-point-of-failure and to preserve the loosely-coupling of services underlying a transaction. Moreover, this new distributed transaction model must take into account the recovery mechanisms in case of transaction failure in such complex scenarios.

However, what is important here is to underline the basic characteristics of this

transaction model. In this way will be possible to understand how service composition happens. In order to meet all previous requirements, transactions are seen as composed by several nested transactions, also known as nested sub-transaction. A tree structure was introduced in order to represent transactions. A *transaction tree* determines the participants and the respective services required for performing a business transaction, i.e. it sets the *context* of the conversation to follow and is issued by the initiator of the transaction. This *transaction context* is used by interaction model to coordinate components of a transaction in a distributed manner to accomplish the services deployment: it clarifies, therefore, the orchestration of services, determining the set of operations and interactions that must be followed by a composite web service by means of other web services in order to realise its overall functionality.

Although service compositions were analyzed according to the usual four dimensions of analysis (data, process, security and protocol) by University of Surrey (UniS) researchers, support to distributed transactions on the P2P network was given above all for data and process composition aspects. In particular, process-oriented service composition was examined identifying its principal aspects, i.e. *order* (if composition of services is serial or parallel), *dependency* (if there is data or function dependency among the composed services) and *alternative service execution* (if there are alternative services in the service composition that can be invoked, either in a sequential or in a parallel manner) aspects.

From these aspects, different *types of service composition* were identified: *Data-oriented*, *Sequential process-oriented*, *Parallel process-oriented* and *Alternative*.

A detailed description for these service composition types can be found in chapter 3.1.2 of [5]. Moreover, more detailed information regarding *Autopoietic P2P Network*, in particular about P2P Layer, interaction model, distributed transaction model developed at UniS for OPAALS project, and supported service composition types described in this short introduction can be found in previous chapter and in [5],[1],[2],[3] and [4].

Notice that previously mentioned different kinds of services composition can be combined in a more complex composition to satisfy the user request.

Regarding the implementation situation, Flypeer, the p2p infrastructure chosen to realize the OPAALS Digital Ecosystem and to implement the above mentioned distributed transaction model, at the moment provides support to three types of transactions (more information regarding Flypeer implementation can be found in [6]):

- ***Sequential***: when actions occur one after the other;
- ***Alternative***: when one action occur and, if it fails, another one takes place;
- ***Parallel***: when two actions occur at the same time.

These supported types of transactions represent the building blocks for our work, the starting point for our reasoning: knowing that a transaction is composed by a set of nested sub-transactions, that each subtransaction can be either a single service or, in turn, a composition of services and that a subtransaction can only be composed in a sequential, parallel or alternative way in order to obtain a more complex transaction, let us the possibility to define how to obtain and express these building blocks in a language closer to

business people language. Building of the whole transaction definition will be obtained by combining these simple building blocks.

In this way we started defining the architecture, the needful logic and the implementation solutions for the editor component so that we can obtain the wanted output, i.e. a transaction context xml file representing the user specified transaction.

Once the xml file is obtained, it will be passed to the Flypeer component called *TransactionInitiator*: this will instantiate the involved services in participating peers and will start the transaction.

In section 3.4 will be described how to specify a transaction by means of SBVR.

3.3 SBVR SEMANTICALLY ANNOTATED SERVICE DESCRIPTION

In section 4 of [3] and in chapter 2 of this work, in a RESTful architecture ([7], [8]) context, SBVR [9] was proposed as the basis of a resources description language and also as user requirements description language. The benefits of this innovative approach appear manifold. Above all the process of resource selection based upon user's requirements should become easier and should happen with a minimal human intervention or in a completely automatic manner.

Following the line traced in the above mentioned approach, we try to take advantages in having a unique description language both for resources both for requirements even in web services composition scenario in the context of Flypeer P2P Network, an almost standard SOA architecture. In such context, a P2P digital ecosystem, in fact, services represents the resources and they are typically described in standard WSDL format but aren't consumed via standard SOAP protocol.

In order to benefit from the proposed approach, we will use SBVR as means to add semantic information to services, in other words we will adopt SBVR as a sort of semantic service description language. Also user requirements for services composition are formulated leveraging SBVR. We can leverage SBVR because it represents new approach to knowledge representation. In this way we can use the same language to express both the semantics of services both the semantic of requests on these services, which are the composition rules. This is made in accordance with the OMG's *Model Driven Architecture* (MDA)[15] model, which try to represent specifications as a Computer Independent Model (CIM): we will adopt this model also for web services composition.

A lot of advantages can be identified in this approach. First of all, a semantic representation of service information can facilitate discovery and service composition. In addition, the transaction representation in a business-friendly language, more similar to the

language of business people, can lower barriers of adoption of this new system, and this can be fundamental for SMEs adoption of the system.

From a practical point of view, to add semantic information to web services for searching and consuming them, we will leverage upon a mechanism to enable semantic annotation of Web services descriptions, precisely the *Semantic Annotations for WSDL and XML Schema* (SAWSDL) [14] mechanism. SAWSDL permits the annotation of WSDL files with extra information regarding semantic aspects for each element in the WSDL file. It is, moreover, perfectly compatible with SOAP protocol that will be used by Flypeer to communicate with the world outside the Digital Ecosystem. By adding a semantic description to the WSDL file we get "semantically" searchable services.

It is possible to add, therefore, semantic information to each part of a service description, i.e. to its definition, to its inputs, to its outputs, to its operations. At the moment, however, we will provide only the semantic annotation for the service definition, using the *modelReference* attribute of SAWSDL.

In this version we will, therefore, annotate only <service> elements in WSDL files, as in Listing 3.1:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns:tns="http://www.example.org/FlowerDeliveryService/"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        name="TaxiYYYBookingService"
        targetNamespace="http://www.example.org/FlowerDeliveryService/"
        xmlns:sawsdl="http://www.w3.org/ns/sawsdl">


//..... ....

<service name="FLYPEER://taxi_booking1"
sawsdl:modelReference="URI_to_file\TaxiBookingSBVRVoc.xml">
     <documentation>This wsdl discribes taxi booking1
service.</documentation>
     <group name="taxi_booking"/>
    </service>
</definitions>
```

Listing 3.1 – an example of WSDL representation with SAWSDL annotations

You can see, highlighted in Listing 3.1, the needful SAWSDL annotations for a WSDL description of an exemplifying taxi booking service: the *sawsdl:modelReference* attribute points to a semantic description for the service. For this description, as already explained, we make use of an ad-hoc XML file representing an SBVR Vocabulary for a service. Typical SAWSDL annotation has the following syntax:

```
sawsdl:modelReference="URI_to_file\ServiceNameSBVRVoc.xml"
```

where the attribute value give the precise URI to the XML representation of the SBVR vocabulary for the given service, i.e. the *ServiceSBVRVoc.xml* file. We propose a naming

convention, given by “ServiceName” + “SBVRVoc” suffix + “.xml” extension, and will be helpful to follow always the same convention. A very simplified version of an xml representation for a SBVR vocabulary of a hotel reservation service is given in Listing 3.2.

```
<?xml version="1.0" encoding="UTF-8"?>
<SBVRVocabulary name="CommonHotelReservation" xmlns:sbvr="sbvrns"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="sbvrns
it.naica.sbvrschema">

      <Terms>
            <Term type="noun">hotel</Term>
            <Term type="noun">room</Term>
            <Term type="individual">Hilton</Term>
            <Term type="noun">client</Term>
            <Term type="individual">London</Term>
            ...
      </Terms>

      <Facts>
            <Fact type="characteristic">room is available</Fact>
            <Fact type="binary">client books room</Fact>
            <Fact type="binary">client books hotel</Fact>
            <Fact type="binary">user books hotel</Fact>
            <Fact type="characteristic">hotel is in London</Fact>
            ...
            ...
      </Facts>

      <Ruleset>
            <Rule>client can book a room only if room is available </Rule>
            ...
            ...
      </Ruleset>
</SBVRVocabulary>
```

Listing 3.2 – an example of xml representation for a SBVR vocabulary

A more accurate definition of SBVR vocabularies and ruleset and their logical formulation can produce better result in for the service discovery mechanism, described in the following section.

Even if there are still a lot of things to do, above all in rules semantic formulation and interpretation, the approach is very promising because it tries to bind the semantic representation to the easiness in using a structured natural language.

3.4 SBVR Transaction Editor

The aim of the *SBVR Transaction Editor* is to obtain, in a business-friendly manner, the *Transaction Context* object of a transaction defined by a user without writing lines of code or editing complex configuration files. The chosen way is to generate an xml file, called ***TransactionContext.xml***, in which is defined the Transaction Context of a long-running transaction. Flypeer [6] uses this file to instantiate and to run transaction in its p2p network.

To obtain this file we will leverage upon a semantic description of composition and upon a semantic search of web services involved in transaction by means of SBVR [9]. The user will type his composition rules, that we also called Production Rules, in accordance with a SBVR Structured English (SBVR SE [9]) notation. An ad-hoc parser has been developed to interpret these rules. The development of this parser represents the first step of the transformation process from the SBVR textual description of the composition into the xml file representing the Transaction Context. The parsing component was developed using ANTLR [11] and the generated classes are in the *it.naica.sbvr2Tx.parser* package.

Due to the fact that Flypeer supports three different basic type of composition, sequential, parallel and alternative, as previously said, then three different kinds of SBVR Production Rules will be used in order to meet the requirements.

The lack of SBVR in supporting temporal logic [12] noticed in previous work [10] was faced also in expressing these sequential, parallel and alternative production rules. This because of SBVR metamodel does not provide a temporal logic support to express temporal constraints on the truth value of a proposition. As a consequence, it is impossible to totally leverage the SBVR metamodel, using understandable, simple and natural statements, to represent services compositions that are heavily based on the sequence of activities and on other temporal concepts. In fact SBVR statements are based on the first-order logic expressions and, so, it is not properly correct to give them a temporal interpretation.

An example is given by *if-then* rules: the SBVR logical formulation of this kind of rules is given as “Implication”, defined in SBVR specification as the “binary logical operation that operates on an antecedent and a consequent and that formulates that the meaning of the consequent is true if the meaning of the antecedent is true”. This means that if the <antecedent> condition is true, at the same time also the <consequent> condition is set at true, as we can see in Figure 1.

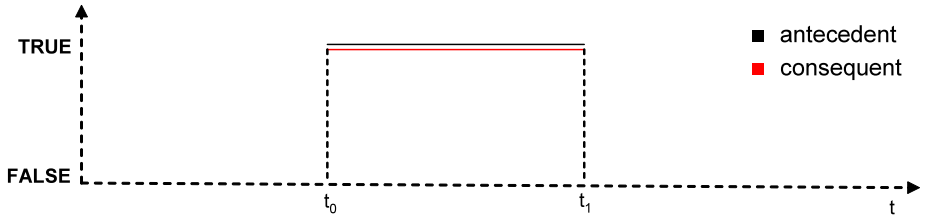


Figure 1 – truth conditions in an if-then expression.

Therefore antecedent and consequent conditions are simultaneously verified in the arbitrary time range [to, t1].

This characteristic represents a problem when we must define a temporal sequence between antecedent and consequent because they are related to the execution of web services or sub-transactions within a transaction.

For example, if we have a typical SBVR rule like ” if the *customer* *books* the *flight* then the *customer* *books* the *hotel*” (take notice: the syntax highlighting is in accordance with SBVR Specification [9]), that is <antecedent> is the activity “the *customer* *books* the *flight*” (i.e. a flight booking web service call and execution), and <consequent> is the activity “the *customer* *books* the *hotel*” (i.e. a hotel booking web service call and execution), in pure declarative logical interpretation it means that the two activities are contemporarily

executed. The desired scenario, needed to satisfy the temporal requirements, is depicted in **Figure 2**.

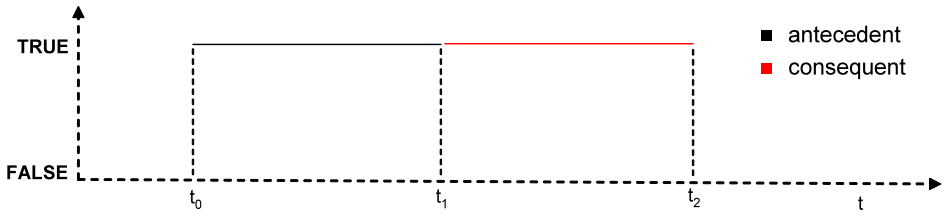


Figure 2 – desired truth conditions scenario

A temporal dimension has been introduced: only after the positive verification of the truth condition of the antecedent in the time range [to, t1], then the truth condition of the consequent will be set to the true value in the time range [t1, t2].

In order to realize the previous scenario, we introduce new keywords (e.g. "after", "while", but only "after" was used in the SBVR Transaction Editor parser) in SBVR SE, allowing the formulation of new statements that involve temporal meanings.

With these keywords in place, new patterns have been introduced to express composition rules supported by Flypeer. This introduction represents a sort of procedural-extended approach for SBVR rules, in addition to usual declarative interpretation of the Business Vocabulary. In this way the knowledge contained in the Business Vocabulary, which belong to the business domain, can be reused.

We will use the following rules patterns to express Flypeer supported transaction (see section 3.2):

Rule pattern	Transaction Type
After *<fact type A>* then *<fact type B>*	Sequential Transaction
After *<fact type A>* then *<fact type B>* and *<fact type B>*	Parallel Transaction
After *<fact type A>* then *<fact type B>* or *<fact type C>*	Alternative Transaction

where "After, then, or, and" are keywords for SBVR and *<fact type A>* are binary fact types.

Fact types are representations of actions the user want to be executed in a transaction, in other words, web services or subtransactions (e.g. if a user write "after the *customer books* the *flight* then the *customer books* the *hotel"* then a web service for booking a flight is

executed and, sequentially after, is called the one for hotel booking).

Transaction types represent a sort of building block: a complex transaction is made of several nested subtransactions that we can express with the above specified rule patterns. Also relationships between subtransaction are limited to the three transaction types. The nesting mechanism is obtained by the *Composer* component, which analyzes and rearranges subtransactions.

For example, we consider the typical simple travel agency scenario seen before. The user, that is the travel agent, wants to start a transaction by booking a flight for its client because the customer wants to choose its destination according to flight price opportunities/special offers. After flight has been booked, then an available hotel must be booked. The user starts by typing the following composition rules in the specific text field, as you can see in Figure 3 – User Production Rules panel:

```
after the customer books the flight then the customer books the hotel
```

With this rule he aims to obtain a sequential composition. First a flight booking web service is called and executed, for example the “British Airways Booking Service”. Only after its correct execution, the second web service, i.e. the hotel booking one, let say the “Hilton Booking Service”, will be called. Notice that rules must be compliant with SBVR SE syntax so they could be correctly parsed, otherwise parsing is impossible.

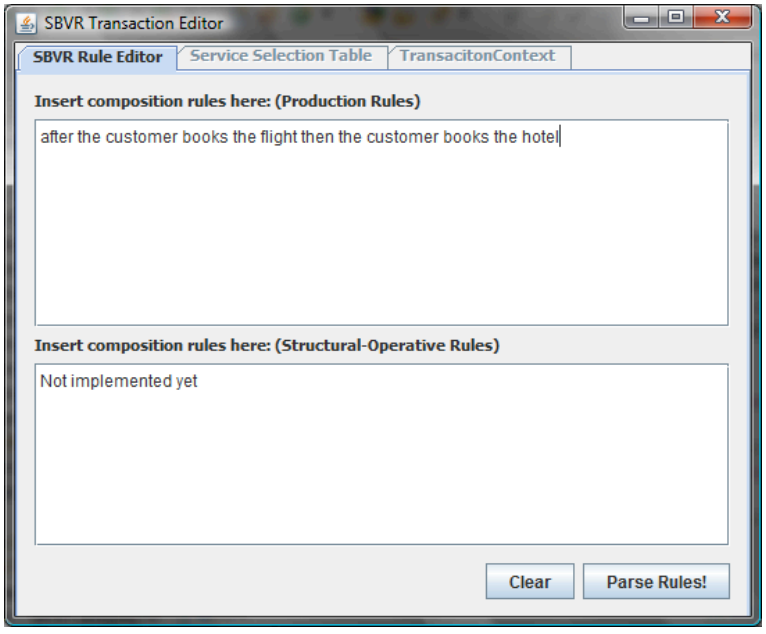


Figure 3 – User Production Rules panel

Each fact type specified in composition rules must be paired with a working web service. This pairing operation is left to the user, who will operate a manual composition. This task, however, is simplified for the user by the functionality of the *SBVR Transaction Editor*,

as the number of services that is possible to select for each of the expected fact types under the rule of composition is reduced considerably by making a selection from the SBVR vocabularies.

This kind of semantic search will narrow the results set.

Thanks to the features of *Discovery Service* provided by JXTA [13], framework used in the implementation of Flypeer, service descriptions stored in the *Local Repository* and the *Global Repository* (see [5]) are retrieved. These service descriptions are given in WSDL. The WSDL descriptions are then parsed and analyzed. In particular, it is analyzed the semantic content attached to the WSDL file (and thus the service) via annotations SAWSDL. This semantic content conveyed via an XML file representative of SBVR Vocabulary and Rule Set, is then compared with the Production Rules of the composition. From this comparison we obtain a set of web services, whose Vocabularies are fitting with the requirements in the rules. So, summing up, from the list of deployed services is returned a list of services matching with composition rules.

The results of the semantic search are reported to the user, who can easily select from a combo box a service associated with each fact type. In order to facilitate this operation information about each service, taken in its WSDL description files, are displayed in a simple interface, as shown in Figure 4.

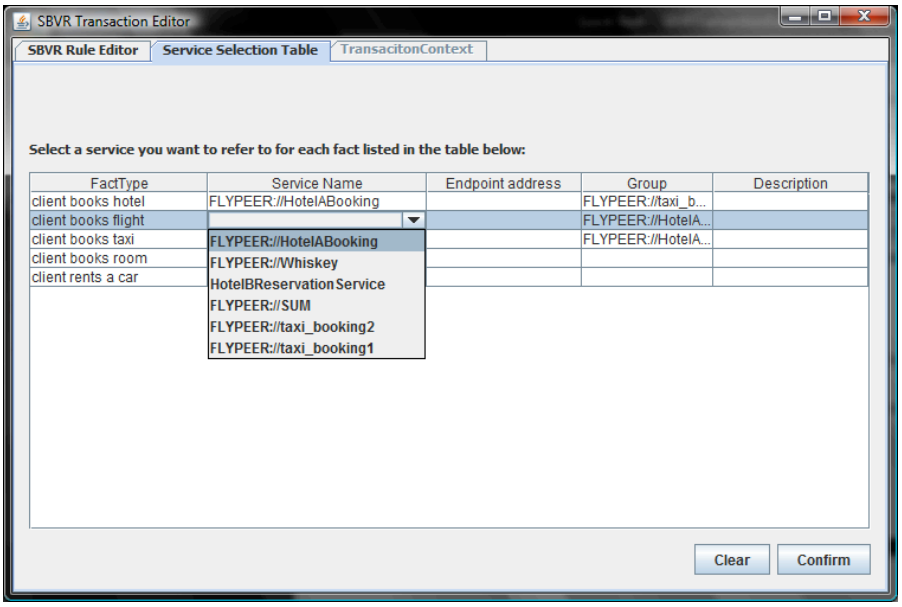


Figure 4 – User interface for service binding

In the future could be provided the support for Structural and Operative Rules, to let the user type rules (i.e.in SBVR these rules are those like “it is obligatory that...”, “it is necessary that...” etc) to refine and improve the semantic search results, so that they can better fit to user requirements. When structural and operative rules will be added, the search results will be improved. In this way, hopefully, user could even not be obliged to choose service: they will be automatically associated. At the moment it is obligatory that user must choose

which service must be bound to each fact (if not he gets an error).

After the user complete this pairing operation of services, then composition rules are processed and filtered. This processing phase is necessary because, as mentioned in section 3.2, the transaction may be composed of several nested subtransactions, and because we have expressed through SBVR SE only the three transaction type supported by Flypeer (sequential, parallel, alternative).

In fact, it is possible to associate the same services for different composition (associated with different fact type), and especially since such services and composition can be recombined to obtain more complex transactions, all the rules of composition should be tested and validated to properly obtain a transaction tree. The logical check operations are performed by classes of the package *it.naica.sbvr2Tx.Element* and especially the class *Composer*.

A simple example about *Composer* functioning is provided.

We suppose that user writes two composition rules:

```
after a then b
after b then c and d
```

These rules will produce two different subtransactions:

```
seq(a,b)
seq( b, par(c,d) )
```

where with *seq(x,y)* we mean the sequential composition of activity x and y, i.e. only after x is completed then will start the execution of activity y. Instead *par(x,y)* is a parallel execution of x and y activities. These are transformed by the *Composer* in the following composed transaction, performing the nesting of subtransactions:

```
seq( a, seq( b, par(c,d) ) )
```

Then it is simplified as shown below:

```
seq( a, b, par(c,d) )
```

As we can easily understand from this trivial example, the *Composer* component has a crucial importance because it must generate a correct transaction context. It actually covers a lot of cases, but it must still be improved.

The *SBVR Transaction Editor* output result is a transaction tree descriptive file, named ***TransactionContext.xml***. This file is obtained starting from the SBVR SE formulation of composition rules without the user has to write a single line of code. An example of TransactionContext.xml files is given in next section.

In order to obtain semantically more relevant results, it is necessary to develop a consistent logical formulation for SBVR service vocabularies and for SBVR composition rules. In this way, having an interoperable machine readable formal representation of a kind of ontological base, it is possible to better operate upon this data, automating services composition and refining results.

3.5 TRANSACTION CONTEXT OUTPUT

The *SBVR Transaction Editor* output is the xml representation of the *Transaction Context* for the specified transaction. This is one of the most important files, which defines the sequence of the services will be executed in the transaction. In fact, the actual execution of the transaction relies on the order of execution and this order is given by the *transaction tree*, on which the coordination mechanism is based. As we have already said, this xml file describes the tree structure used to represent transactions according to the distributed transaction model to which we have previously referred. More information about the transaction context can be found in sections 4.2 and 6.2 of [5]. Information regard the *transaction context schema* can be found in section 8.1 of [5] and in, for implementation details, in [6].

We briefly summarize its key elements.

As obvious, because it represents the transaction tree, the root of the transaction context xml file is the **TransacitionTree** element. A ***transactionId*** attribute of this element is the unique identifier that each transaction must have. This value is assigned by the transaction initiator. The root contains the main **SubTransaction** elements represent sub transactions inside the tree. They must have the ***subTransactionId*** attribute and this is the id of this subtransaction and must be unique. The SubTransaction element may contain the **WebService** or **Composition** elements. The WebService element represents a service that will be executed and will have a ***serviceId*** parameter, which is the id of this service. The *Composition* element defines a order of execution of the services. There are three types of services compositions supported by Flypeer: *Sequential, Parallel or Alternative*. To specify the kind of composition, users must assign the value at the *compositionType* attribute of *Composition* element. An example from [6] of TransactionContext.xml is shown in Listing 3.3.

```
<TransactionTree xmlns='http://xml.netbeans.org/schema/TrnTree'
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xsi:schemaLocation='http://xml.netbeans.org/schema/TrnTree file:/C:/TranContext/TranTree.xsd'
    TransactionId="0001">
    <SubTransaction subTransactionId="010">
        <Composition compositionType="SEQUENTIAL">
            <SubTransaction subTransactionId="011">
                <Composition compositionType="SEQUENTIAL">
                    <SubTransaction subTransactionId="0111">
                        <WebService serviceId="JXTA-HOTEL-BOOKING/FABIO">
                            <ServiceDescription>
                                A VARIG Flight Booking Service
                            </ServiceDescription>
                        </WebService>
                    </SubTransaction>
                    <SubTransaction subTransactionId="0112">
                        <Composition compositionType="ALTERNATIVE">
                            <SubTransaction subTransactionId="01111">
                                <WebService serviceId="GOL-FLIGHT-BOOKING">
                                    <ServiceDescription>
                                            A VARIG Flight Booking Service
                                    </ServiceDescription>
                                </WebService>
                            </SubTransaction>
                            <SubTransaction subTransactionId="01112">
                                <Composition compositionType="SEQUENTIAL">
                                    <SubTransaction subTransactionId="011121">
                                        <WebService serviceId="JXTA-HOTEL-BOOKING/PAULO">
                                            <ServiceDescription>
                                                    A VARIG Flight Booking Service
                                            </ServiceDescription>
                                        </WebService>
                                    </SubTransaction>
                                    <SubTransaction subTransactionId="011122">
                                        <WebService serviceId="JXTA-HOTEL-BOOKING/OUTRO">
                                            <ServiceDescription>
                                                    A VARIG Flight Booking Service
                                            </ServiceDescription>
                                        </WebService>
                                    </SubTransaction>
                                </Composition>
                            </SubTransaction>
                        </Composition>
                    </SubTransaction>
                </Composition>
            </SubTransaction>
            <SubTransaction subTransactionId="012">
                <Composition compositionType="ALTERNATIVE">
                    <SubTransaction subTransactionId="0121">
                        <WebService serviceId="VARIG-FLIGHT-BOOKING">
                            <ServiceDescription>
                                A VARIG Flight Booking Service
                            </ServiceDescription>
                        </WebService>
                    </SubTransaction>
                    <SubTransaction subTransactionId="0122">
                        <WebService serviceId="TAM-FLIGHT-BOOKING">
                            <ServiceDescription>
                                A TAM Flight Booking Service
                            </ServiceDescription>
                        </WebService>
                    </SubTransaction>
                </Composition>
            </SubTransaction>
            <SubTransaction subTransactionId="013">
                <WebService serviceId="JXTA-HOTEL-BOOKING/DENIS">
                    <ServiceDescription>
                        Hotel Booking Service
                    </ServiceDescription>
                </WebService>
            </SubTransaction>
        </Composition>
    </SubTransaction>
</TransactionTree>
```

Listing 3.3 – a TransactionContext.xml example

Once this file has been successfully generated, Flypeer can use it to initialize the involved services and then it will start the transaction. This task is left to the *Transaction Initiator* object.

The xml file is parsed by Flypeer *TransactionContextParser* class, as shown in Listing 3.4:

```
  TransactionContextParser tcp = new TransactionContextParser();
  tcp.setDocBuilderFactory(DocumentBuilderFactory.newInstance());
  TransactionContext
transCtx=tcp.load(SomeClass.class.getClassLoader().getResourceAsStream("MyT
ransactionContext.xml"));
```

Listing 3.4 – TransactionContextParser class usage

The *TransactionContextParser* will return a *TransactionContext* object (notice that this can be obtained even programmatically by implementing the *TransactionFlow* class, see [6] for more details). This object is then used by the *TransacionInitiator* to start the transaction, as shown below:

```
  TransacitonInitiator  transactionInitiator=new  TransactionInitiator(Peer
peer);
  transactionInitiator.startTransaction(params,transCtx,new
MyFlypeerResponseListener());
```

In this way the transaction context will be sent to other participants of transaction. They will response to the initiator and this will trigger the transaction.

3.6 Conclusion and Future Work

In this work we evaluated the feasibility of an approach based on SBVR for the definition of a business transaction for a digital ecosystem based on Flypeer. We implemented a software component, the *SBVR Transaction Editor*, which acts as a graphical user interface for p2p network of Flypeer and that allows the specification of a transaction from a language (SBVR SE) closer to that of business people (where the term "business" is not to be intended only into commercial business domain such as finance and manufacturing, but also to other domains such as education, government, medicine, and law, as intended in the MDA approach). The same SBVR was used to add semantic information to syntactic descriptions, expressed with WSDL, of web services deployed in the p2p network of Flypeer. The SBVR vocabularies of services were connected to the WSDL file using the mechanism provided by SAWSDL semantic annotation.

The phase of services composition, which is still manual, is facilitated by a semantic search that narrows the search results of services, evaluating the relevance of semantically

annotated vocabularies of individual services with the composition rules.

The results were encouraging, although it has been tested in cases of not too complex transaction.

The benefits of this approach are given by having the same language for both the semantic definition of services, both for the description of the requests on services. Furthermore, leveraging on a structured language as SBVR SE for specification allowing to develop simpler user interfaces for business people, makes the systems easier adoption of Digital Ecosystem, reducing learning time and lowering barriers to entry for inexpert users, especially for SMEs. Furthermore, having worked only with standards such as XML, WSDL and SAWSDL, generally, allows enjoying a wide interoperability.

The disadvantages of the approach are given by a certain slowness of the analysis phase due to the numerous parsing operations (of composition rules expressed in SBVR SE, of WSDL files, of XML service description vocabularies) and to the large number of services to be analyzed: even if it is a one-time effort (once a transaction has been defined, several instances can be run), it takes a considerable amount of time and, however, represent a problem. This is mainly due to the fact that the use of SBVR as resource description language is designed for a REST environment (with fixed and simpler interfaces) and not for a WS-* based environment (where it is necessary to retrieve the WSDL file, parse the WSDL file, to go back to the vocabulary files in the semantically annotated WSDL description of the service, parse the xml SBVR-vocabulary file etc.). Also the lack of a consistent formulation for SBVR semantics does not allow, at this time, to obtain results more semantically relevant.

Among the directions of future research, there is certainly one that aims to develop a consistent logical formulation for SBVR vocabularies descriptions and for SBVR composition rules, essential to harness the SBVR ontological power and to refine searches to obtain the better results. Additionally, by including and implementing control algorithms (e.g. with constraint logic programming techniques) may be able to almost completely automate the composition of services, which would make it much easier to use the OPAALS platform for services exchange. As part of this work there is also the development of a general parser for SBVR. It is also possible to think about sharing mechanisms of vocabularies on semantic basis to strengthen the expressiveness of languages that are based upon such vocabularies.

It is possible to improve the job done, as explained below:

- Adding the ability to parse the structural and operative rules;
- Improving *Composer* module because not all possible combinations have been covered yet;
- Improving the error reporting mechanism;
- Adding syntax highlighting, auto completion, and other nice similar features to help user in typing rules;
- Enhancing parsing capabilities e nesting mechanism (the Composer implementation) if support to new composition types in Flypeer is added.

3.7 REFERENCES

[1] P. Krause, S. Moschoyiannis, A. Razavi *D3.1: Preliminary architecture for P2P network focusing on hierarchical virtual super-peers, birth and growth models*, OPAALS Project, June 2007 (from http://opaals.eu/ repository)

[2] P. Krause, S. Moschoyiannis, A. Razavi. *D3.2: Report on formal analysis of autopoietic P2P network, together with predictions of performance,* OPAALS Project, August 2007 (from http://opaals.eu/ repository)

[3] P. Krause, A. Marinos, S. Moschoyiannis, A. Razavi, Y. Zheng, Mikala P. Streeter, Jesús E. Gabaldón, T.Kurz, SUAS, CreateNet. *D3.3:Full Architecture Definition,* OPAALS Project , July 2008 (from http://opaals.eu/ repository)

[4] S. Moschoyiannis (Unis), M. L. Darking (Lse), L. Rivera Leon (T6), A. Passani(T6), J. Val (Ita), P. Malone (Wit), M. McLaughlin (Wit), P. Tsatsou (Lse), J. Stanley (Cam), M. Iqani (Lse), A.Razavi (Unis), P. Krause (Unis) *D3.6: Consensus detailed architecture of the OPAALS DE*, OPAALS Project , May 2008 (from http://opaals.eu/ repository)

[5] P. Krause, A. Razavi, *D3.10: Integrated autopoietic DE architecture*, OPAALS Project , September 2009 (from http://opaals.eu/ repository)

[6] Flypeer – Dynamic P2P Infrastructure — Project Kenai, 2008, http://kenai.com/projects/flypeer

[7] R.T. Fielding, "*Architectural Styles and the Design of Network-based Software Architectures*"
University of California - Irvine, 2000.

[8] L. Richardson and S. Ruby, "*RESTful Web Services*" O'Reilly Media, Inc., 2007

[9] OMG. *Semantics of Business Vocabulary and Business Rules Specification* (SBVR), January 2008. http://www.omg.org/spec/SBVR/1.0/PDF/ - Last accessed: June 24,2010.

[10] R.Eder, A.Filieri, T.Kurz, A.Margarito. *D2.3 - Extended vocabulary and rule set for an existing scenario,* OPAALS Project, November 2008 (from http://opaals.eu/ repository)

[11] *ANTLR*, ANother Tool for Language Recognition, http://www.antlr.org/ - Last accessed: June 24,2010

[12] Stan Hendryx. *Semantic Interpretation in Terms of the SBVR Logical Formulation of Semantics Vocabulary*. August 2008.

[13] Home Web page for Project JXTA: http://jxta.dev.java.net

Project JXTA specification: http://jxta-spec.dev.java.net
Project JXTA reference implementation (http://jxta-jxse.dev.java.net)
- Last accessed: June 24,2010

[14] *World Wide Web Consortium (W3C): Semantic Annotations for WSDL and XML Schema (SAWSDL) Specification* (http://www.w3.org/TR/sawsdl/ and *Semantic Annotations for WSDL and XML Schema - Usage Guide* (http://www.w3.org/TR/sawsdl-guide/) - Last accessed: June 24,2010

[15] *OMG Model Driven Architecture* MDA http://www.omg.org/mda/ - Last accessed: June 24,2010

4 SEMANTICS OF BUSINESS PROCESS VOCABULARY AND PROCESS RULES (IITK)

Semantics of Business Vocabulary and Business Rules (SBVR), an OMG standard, provides a meta-model for the semantic and declarative models of business vocabulary and business rules. Logical formulation of SBVR facilitates IT people to interpret these models generated by business people. However, an important aspect of a business is the process models, and it is outside the scope of SBVR. Knowledge intensive and dynamic nature of business processes require such a declarative meta-model for process modeling in order to provide flexibility and adaptability. In this work, an approach for process modeling using SBVR's methodology is proposed. We have made an initial attempt to define a declarative meta-model, *Semantics of Business Process Vocabulary and Process Rules (SBPVR)*. SBPVR divides knowledge of business processes into three parts; process concept types, process fact types and process rules. Process concept types and fact types represent structure of processes and process rules provide guidance over the structure and flow of processes at execution time. In the chapter, these types are further categorized to represent the various elements of process models. SBPVR provides flexibility and adaptability to the process model through its declarative nature and fact oriented approach.

4.1 INTRODUCTION

Enterprise modeling entails building conceptual models for various aspects of business, e.g., structure, processes, constraints, resources etc. These models work as communication standards for business stakeholders and this modeling provides efficiency, effectiveness and agility to the business.

Business Process Modeling and *Business Rule Modeling* are the most prevalent approaches for modeling behavior of business. In process modeling, operational behavior of an enterprise is represented through process models. Rule modeling approach follows the methodology of separating business rules from process models and defining rule models as static nature of business. Following *business rules approach* [15], Object Management Group (OMG) has given *Semantics of Business Vocabulary and Business Rules (SBVR)* [6], a declarative meta-model for describing business vocabulary and business rules. Benefits of SBVR are its declarative nature, rule modeling approach, natural language representation and formal logical backbone. However, process related concepts are outside the scope of SBVR.

Both the process modeling and rule modeling approaches have different benefits in terms of semantic representation. Representational analysis done by Michel et al. [11], shows that any single approach is not capable of representing all business constructs. Rule modeling approach focuses on decision points which regulate the business. On the other hand, process modeling approach tries to minimize the amount of work required in processing, but ignores decision points. Michel et al. showed that combination of a rule modeling language and a process modeling language covers maximum representational constructs.

From control flow perspective, process modeling can be categorized into two types:

- **Procedural Modeling** defines a process as an exact sequence of process elements (tasks, events, etc.) and routing elements (gateways). Examples of procedural process modeling languages are; BPMN [8], BPEL [13], EPC [17], WorkFlowNets [23], UML Activity Diagram [7] etc.
- **Declarative Modeling** defines a process as a set of process elements and declarative statements representing constraints over them. Examples of declarative process modeling languages are; Case Handling [22], Penelope [5], ConDec [14], DecSerFlow [21], Constraint Specification Framework [16] etc.

In procedural modeling, structure and execution path of a process is fixed at design time. Compared to this, a declarative process model only defines "what" this process offers to the business, leaving out the information of "how" to achieve it. In practice, a designed process model might be executed in different operating conditions of business. In knowledge intensive business processes, sometimes it is not feasible to specify exact execution path of a process at design time. Process models also evolve as a result of process analysis and process improvement. Thus, for knowledge intensive and dynamic processes, flexibility and adaptability are essential requirements of process models.

The most popular and widely used notation for process modeling is BPMN due to its simple graphical notation, which is closer to business people. However, process models prepared using BPMN are procedural in nature and contain the same problems mentioned above. There also exist several declarative languages in literature but most of them cover only a specific part of the process model. For example, Penelope only describes sequence and timing constraints of a process model. Constraints in ConDec are for control flow. Case-Handling mainly targets data-constraints and authorization.

To overcome these problems, there is a need for a generic declarative process meta-model which can be integrated with rule meta-model (SBVR). In this work, we have applied the SBVR approach to model business processes. A generic meta-model for declarative business process modeling, Semantics of Business Process Vocabulary and Process Rules (SBPVR) is proposed. SBPVR provides conceptual vocabularies to define process elements, their semantics and process rules. SBPVR also supports natural language representation (Structured English) for the process models.

Structure of the chapter is as follows; Section 6.2 gives a brief introduction to SBVR and existing process modeling languages. Section 6.3 defines vocabulary and rule model of SBPVR. Section 6.4 presents benefits of SBPVR. Section 6.5 concludes the chapter with future scope.

4.2 Related Literature

4.2.1 SBVR

SBVR categorizes business knowledge into three parts; *Concept types*, *Fact types* and *Business rules*. Methodology to create SBVR models follows Business Rules Mantra, *"Rules are built over fact types and fact types are based on concepts."* [6] [15]. Figure 6-1 depicts the methodology of SBVR. SBVR separates meaning of a concept from its representation.

SBVR also supports natural language representation (*Structured English)* which enhances its usability for business people. A graphical representation has also been developed by Musham et al. [12].

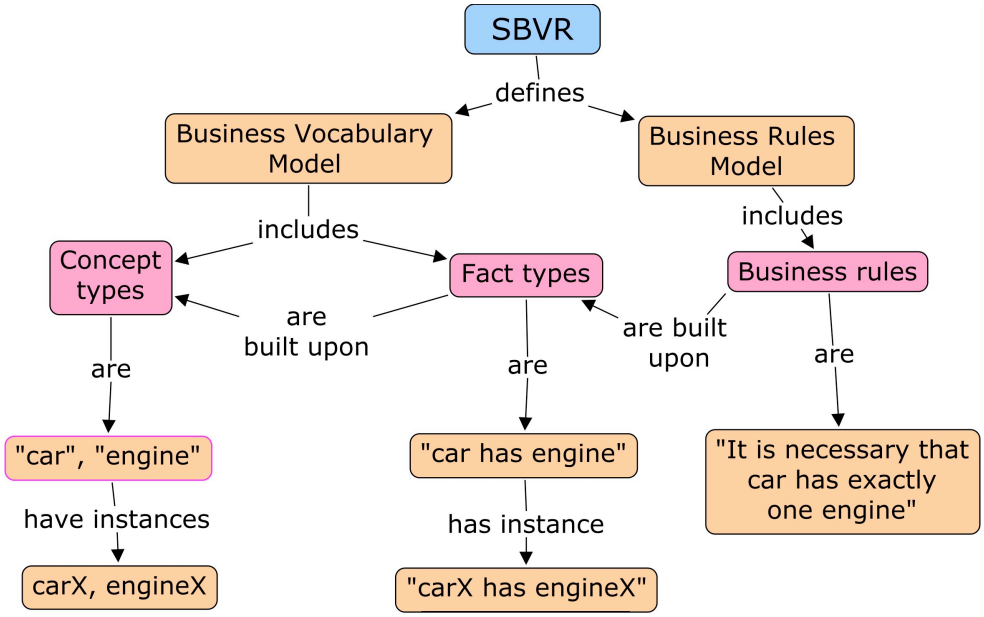


FIG. 6-1 SBVR METHODOLOGY

4.2.2 EXISTING BP MODELING LANGUAGES

There exist several process modeling languages in the literature. These languages differ in the terms of procedural or declarative nature, target area, audience, representation, formalization etc.

BPMN has become de-facto standard for most of the BPM suites due to its simple graphical notation. Languages like BPEL and UML Activity Diagram, which target system model, are too complex to be used by business people. Models defined using BPMN are procedural in nature and lack in semantics.

Compared to procedural languages, most of the declarative languages cover only a specific part of a process model. Sadiq et al. [16] proposed a constraint specification framework for flexible business processes. This approach can be viewed as a starting point for declarative modeling as only a part of the process model is described declaratively. Aalst et al. [22] have given a declarative paradigm to model flexible business processes where execution of the process depends upon the current case i.e. data produced by the process so far. However, it is very restricted due to data-driven approach and does not cover all possible real scenarios. Pesic et al. [14] have given a declarative language ConDec, based on Linear Temporal Logic which consists of tasks and constraints (mainly for control flow). Graphical representation of ConDec is also complex to be used by business people. Goedertier et al. [5] have presented a declarative language, Penelope which mainly focuses on sequence and timing constraints of a process model.

Languages like WorkFlowNets [23] and EPC [17] are supported by mathematical formalization (petri-nets, π-calculus etc.). The downsides are their procedural nature and limited coverage of various aspects of process modeling.

4.3 Semantics of Business Process Vocabulary and Process Rules

SBPVR follows the methodologies of SBVR [6] and Business Rules Approach (BRA) [15] to represent process models. Following the fact-oriented approach, SBPVR divides process knowledge into three parts; *process concept type*, *process fact type* and *process rules*. Figure 6-2 explains these categories with instances from real world. Definitions of these categories are:

- *Process concept type* represents a dynamic entity in the process model. For example, task, event, interaction etc. Instance of a process concept type represents the state of affairs happening in the business.
- *Process fact type* represents either characteristic of a *process concept type* (unary fact type) or relation between two or more *process concept types* (binary or n-ary fact type). Instance of a fact type represents relation between instances of *process concept types*.
- *Process rules* define constraints over the structure and flow of the process.

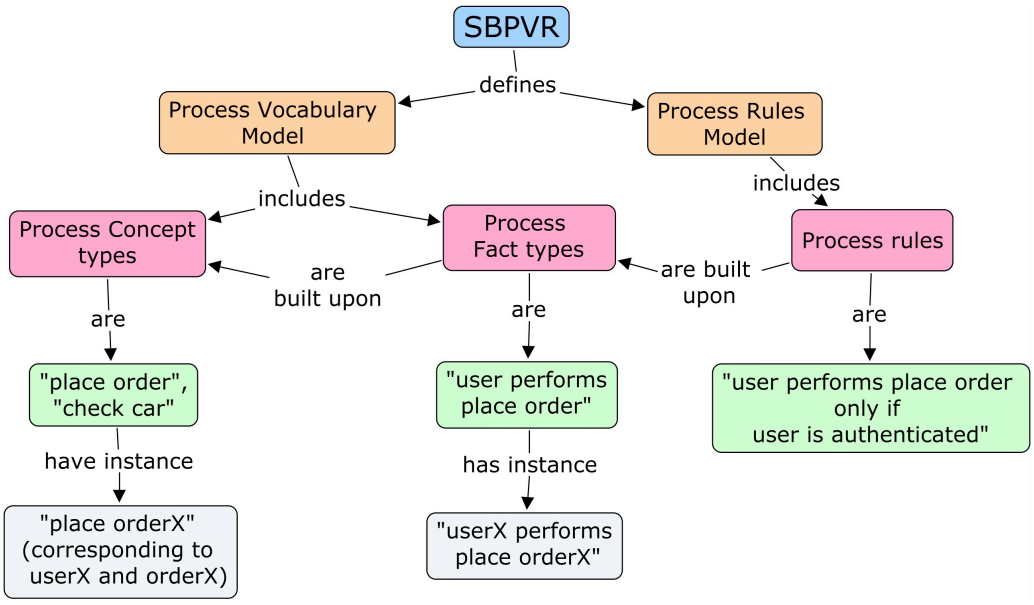


FIG. 6-2 SBPVR METHODOLOGY

4.3.1 Vocabulary for Describing Business Process Vocabulary

This section specifies the vocabulary to be used for describing elements in the process models. Elements of process model and their semantics are derived from existing work in the field of process modeling (BPMN [8], Penelope [5], EPC [17], BPEL [13], Case Handling Paradigm [22]) and in the field of process specification (PSL [18] and ebXML [3]).

To provide a generic and extendable meta-model, these elements and their semantics are categorized in a hierarchical fashion. This categorization is described in the next subsections.

4.3.1.1 Process Concept Type

Process concept type is the abstract class for all dynamic entities in the process model. Semantics of a *process concept* are its relations to the happening on its enactment and changes occurred due to its performance. Happening is the meaning of the activity of change in business. For example, meaning of a task type, *"Place Order"* is the actuality of an instance of *Customer* giving an instance of *Order* to an instance of *Salesman.*

A *process concept* may be defined in the context of a business process. A *process concept type* can generalize or specialize another *process concept type*.

Each *process concept* incorporates characteristics which make it unique in the model. Examples of such characteristics are, "*Concept Type*" (task, event) , *"State"* (Start, Error, Finish) etc. Semantics of a *process concept* are independent of the way of its execution. Figure 6-3 shows the categorization of *process concept type* as explained in the next subsections.

4.3.1.2 Work Type

Work type specializes *process concept type* and formulates work to be performed or coordinated by an agent or a coordinator respectively. *Work type* is a general concept for the elements which involve work to be performed (tasks, activities and whole processes).

In addition to characteristics of *process concept type*, *work type* has a role binding with an *agent* (SBVR: concept type). A *work type* may belong to one speech community or can represent collaborative work between two speech communities. It has a place holder for its owner (SBVR: concept type). A *work type* has a purpose associated with it.

Work type has a placeholder for business rules which guide its enactment. Output of *work type* can be represented in terms of changes in the *state model*. Following are the specializations of *work type* on the basis of granularity:

- *Task type* represents unit amount of work in the process model.
- *Activity type* represents a set of tasks, interactions and events.
- *Individual process* represents a complete process which achieves some business goal or provides some service.

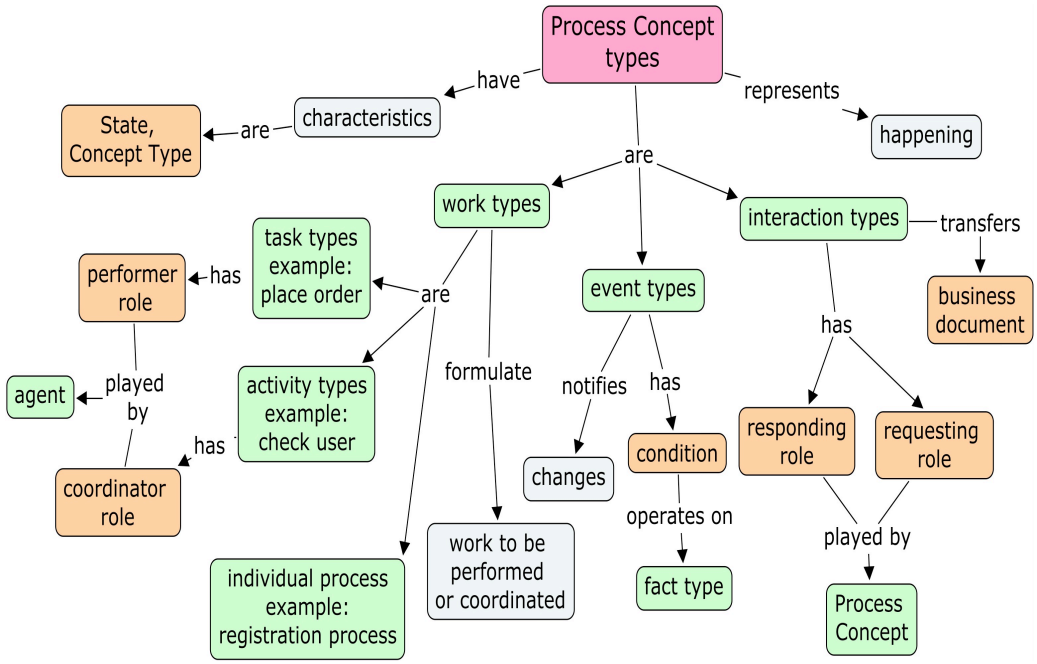


FIG. 6-3 SBPVR PROCESS CONCEPT TYPES

4.3.1.3 INTERACTION TYPE

Interaction type specializes *task type* in which a business document (message or artifact) is being transferred during its enactment. For the generalization of both orchestration and choreography processes, interactions are first class concepts in the SBPVR. An *interaction type* has two role bindings, *requesting role* and *responding role* associated with it. This binding represents transfer of business document from responding role to requesting role.

4.3.1.4 EVENT TYPE

Event notifies changes in the *state model* to *process concepts*. Happening of an event is instantaneous. Event has triggers associated with it which cause its occurrence in the process.

4.3.1.5 STATE MODEL

State model represents state of the business at one particular point of time. SBPVR models are represented by process schema, which includes *process concept types*, *process fact types* and *SBVR:conceptual schema*. *State model* is an instance of process schema in one possible world scenario at one point of time.

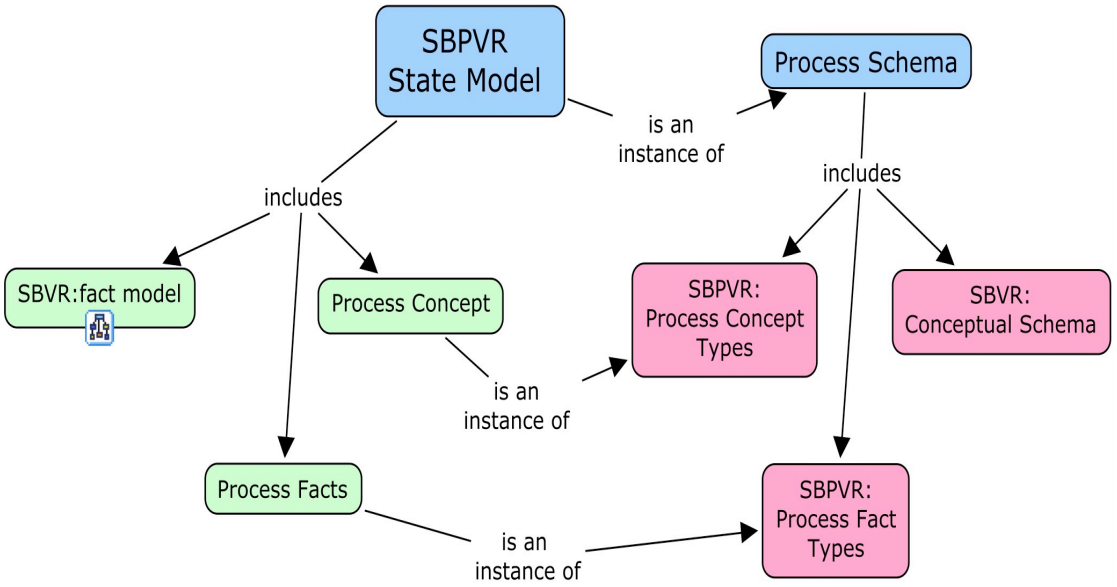


FIG. 6-4 SBPVR STATE MODEL

4.3.1.6 PROCESS FACT TYPE

A process fact type is a relation between two concept types (at least one is process concept type). The relation can be between a dynamic entity (process concept type) and a static entity (SBVR:Concept type) or only between dynamic entities. A process fact type is based on a verb concept (process fact type role) which binds two or more entities in semantic relation.

Semantics associated with a process fact is the kind of relationship between concepts. SBPVR divides these relationships in 7 categories. These categories and respective examples are shown in Figure 6-5.

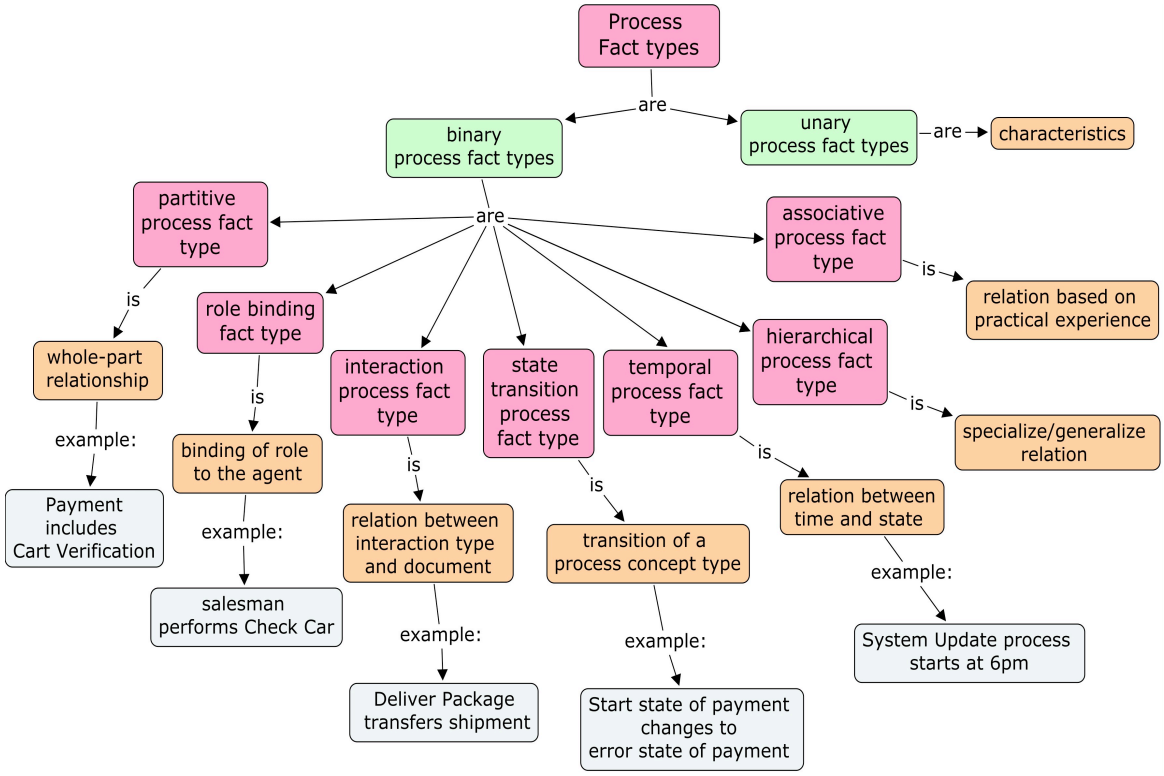


FIG. 6-5 PROCESS FACT TYPES IN SBPVR

4.3.2 PROCESS RULES IN SBPVR

In SBPVR, process rules are statements which constrain or guide business behavior in the context of business processes. In traditional procedural languages, these constraints or design decisions are either hidden or defined implicitly. These approaches negotiate flexibility, traceability and modifiability of process model. In SBPVR, process rules are extracted from the process diagram and represented separately.

A process rule can have two kinds of guidance (based on SBVR's categorization of guidance):

- *Structural Guidance* claims necessity on the structure of process models and cannot be violated during the process enactment.
- *Operative Guidance* claims obligation on the behavior of process models.

Another popular categorization of rules is given by Taveter el al. [20]. They have classified rules into four categories; *Integrity constraints, Derivation rules, Reaction rules and Deontic assignment*. Muninder et al. [2] have shown that business protocols can be described using commitments over the agents. Hay [9] has categorized rules into *structural assertion, action rules and derivation rules*. He also included authorization rules.

Goedertier et al. [4] have given sixteen types of rules divided into three categories (Control flow, Data Aspects and Organizational Rules). These categories are influenced by work of Jablonski et al. [10]. SBPVR does not aim to restrict itself to few specific types of rules but to provide an abstract classification of process-aware business rules.

In SBPVR, process rules are categorized into five categories, influenced from the work of Wagner et al. [20]. Since SBPVR only targets the rules which are defined in the context of a process, definition and boundaries of above categories are defined in the context of a process. Following subsections describe these five categories and for each category, they also specify few kind of types belonging to that category. SBPVR does not aim to restrict itself to only the mentioned types and they can be extended by enterprises. Figure 6.6 shows the categorization of process rules in SBPVR.

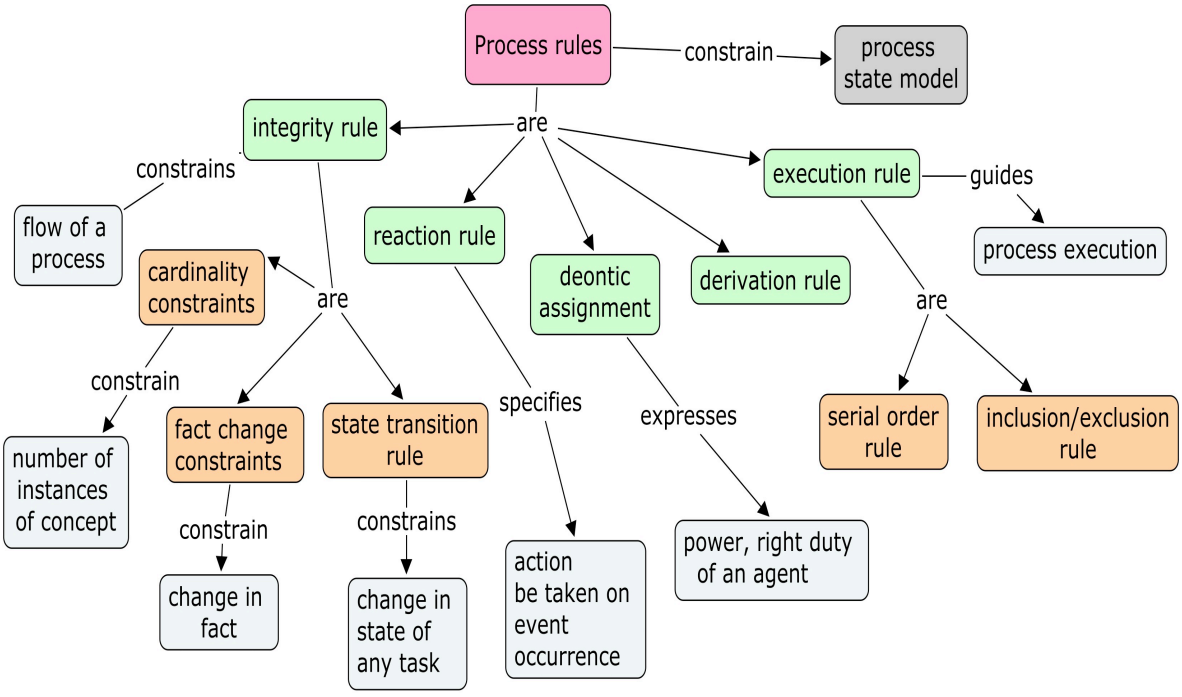


FIG. 6.6 PROCESS RULES IN SBPVR

4.3.2.1 INTEGRITY RULE

Integrity rule constrains flow or integrity of a process model. In SBPVR, dynamic entities and state transitions are represented by *concept types* and *fact types* respectively. *Integrity rules* guides to maintain integrity of the *state model* in the operations where instances are added or removed from it. Following are the few types of *integrity rules*:

- *State transition constraint:* This rule controls transition of a *process concept* from one state to another. Constraints are formulated by existence or non-existence of *facts* in the *state model*. Examples of such rules:

  It is necessary that accept-order starts only if user has order and user is valid (activity precondition).

  It is necessary that accept-order is completed only if order is accepted (activity post-condition).

- *Fact change constraint:* This type of rule constrains the change of the *fact types* in the *state model*. Example is:

It is not possible that fact type, order is rejected, changes to order is accepted.

- *Cardinality Constraint:* This type of rule constrains the number of instances of a *concept type* in the context of another *concept type*. For example:

  There exists exactly one accept-order activity as a part of handle order process.

4.3.2.2 Reaction Rule

This type of rule specifies actions to be taken on the occurrence of *process fact types* in the *state model*. This rule constrains state transition of one *process concept* in the presence of another *process fact* (e.g., event being triggered). Example of reaction rule is:

When pending customer event triggers, It is necessary that handle customer process starts.

4.3.2.3 Derivation Rule

Derivation rules are rules where element of knowledge is derived from existing concepts. SBPVR supports full derivation using if-and-only-if (equivalence) or partial derivation (using if) keywords. Example:

Accept order activity is completed if and only if user is valid and order is accepted.

4.3.2.4 Deontic Assignment

This rule expresses power, right or duty of an agent on its role in process model. Examples of this type of rules are:

- *Authorization:* In this type of rule, commitments are assigned to agents to perform or co-ordinate work concept. This assignment depends upon the properties of work concept, properties of agents and state model. Examples:

  It is the duty of salesman that salesman perform take-order activity.

  It is the right of supervisor that supervisor performs reject-order activity.

- *Event subscription constraint:* This type of rule constrains the agents to perceive events in the process. Example:

  It is not permissible that a vendor perceives accept order event if that order is of price less than $1000.

4.3.2.5 Execution Rule

This rule constrains or guides execution order of process elements. Although, SBPVR does not specify exact execution path of a process model, some constraints or advices to execution platform might be necessary to define at design time. For example, exclusiveness of two activities can constrain execution plan of process model. Examples for this type of rules are:

- *Serial order constraint:* This rule constrains whether two activities can be executed in sequential or parallel order. Example:

It is not possible that interview-process and written-exam of a candidate occur at the same time.

- *Activity inclusion/exclusion constraint:* This rule specifies inclusiveness or exclusiveness of activities. Example:

  If reservation process includes flight-booking and reservation process includes hotel-reservation than It is necessary that reservation process includes free-car-pickup process.

Examples of processes modelled in SBPVR are represented in [1].

4.4 Benefits of SBPVR

4.4.1 Flexibility

From a process modeling language perspective, SBPVR is analysed for flexibility of the process model at execution time. Taxonomy for flexibility given by Schonenberg et al. [19] is used for this analysis.

- *Flexibility by design* demands that multiple execution paths can be defined for a process model if possible. Thus, the execution platform can choose to execute any possible path. In SBPVR, execution path is not fixed at design time and execution platform can choose any possible path of process model.
- *Flexibility by deviation* is the ability of a process model that in a process instance, execution of two tasks can be altered at the runtime without changing its definition. In SBPVR, two tasks can have no dependency, necessity claim or obligation on execution order. Apart from necessity claim, any path can be altered at run time.
- *Flexibility by under-specification* is the ability of an execution platform to execute partially defined process model. As this flexibility depends upon the execution platform, support from modeling language is required to have concepts of *"Late binding"* and *"Late modeling"*. In SBPVR, each concept is defined separately with its semantics. Output of any process concept is represented as changes in state model. Thus, SBPVR provides modeling level support for this flexibility.

4.4.2 Adaptability

Each element in SBPVR is defined separately with its semantics (characteristics). Addition of a new concept does not change existing definition of the process. Due to fact oriented and business rules approach, a process concept and its dependents can be easily located. So change in existing concept will only affect its definition and dependents.

4.4.3 SBVR Based Approach

The fact-oriented approach of SBPVR represents process knowledge at the lowest level of granularity, which is generally used by business people in communication. The hierarchical structure of SBPVR's meta-model not only defines all the basic constructs for process modeling but also provides scope for extension. SBPVR meta-model enables integration

between business vocabulary model, business rules model and business process model. This integration provides reusability and uniform representation of knowledge. This integration also enables identification of business rules affecting process models. The operative guidance (obligation) of process rules permits users to specify rules which can be violated at run time.

4.5 Conclusion

In this work, we have made an initial attempt to define a generic meta-model named, SBPVR, for declarative business process modeling. SBPVR follows SBVR's fact oriented approach and process models in SBPVR are built over SBVR vocabulary & rule models. SBPVR categorizes process knowledge into process concept types, process fact types and process rules. We have extended these categories further, to represent various process elements and their semantics. Benefits of the declarative nature of SBPVR are flexibility and adaptability which are critical requirements for knowledge intensive and dynamic process models. SBPVR enables integration between rule model and process model, which covers maximum representational constructs using uniform methodology.

SBPVR has its scope as a communication standard for process models in business process management and process-aware system development. However, the current version of SBPVR requires substantial exploration in terms of representational and semantic issues. In future, we would like to see SBPVR as a complete meta-model with semantic formulation and transformation rules to system level execution languages (e.g., BPEL, PSL etc.). Further, a textual (Structured English [6]) or graphical (Visual Syntax [12]) syntax can be developed for SBPVR.

4.6 References

1. Ashish Agrawal. Semantics of Business Process Vocabulary and Process Rules and a Visual Editor of SBVR. Master's thesis, Indian Institute of Technology Kanpur, India, 2009. http://www.cse.iitk.ac.in/users/agrawala/thesis.pdf.
2. Amit K. Chopra and Munindar P. Singh. Commitments for flexible business processes. In AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, pages 1362–1363, Washington, DC, USA, 2004. IEEE Computer Society.
3. J. Clark, C. Casanave, K. Kanaskie, B. Harvey, N. Smith, J. Yunker, and K. Riemer. ebXML business process specification schema version 1.01, 2001.
4. Stijn Goedertier, Raf Haesen, and Jan Vanthienen. EM-BrA2CE v0.1: A Vocabulary and Execution Model for Declarative Business Process Modeling. SSRN eLibrary, 2007.
5. Stijn Goedertier and Jan Vanthienen. Designing compliant business processes with obligations and permissions. Business Process Management Workshops, pages 5–14, 2006.
6. Object Management Group. Semantics of Business Vocabulary and Business Rules (SBVR), Version 1.0. http://www.omg.org/spec/SBVR/1.0/.
7. Object Management Group. Unified Modeling Language: Activity Diagram. http: //www.uml.org/.

8. Object Management Group. Business Process Modeling Notation (BPMN), January 2008. http://www.omg.org/spec/BPMN/1.1/.
9. David Hay. Defining business rules; what are they really? Guide business rules project, final report, 2000.
10. Stefan Jablonski and Christoph Bussler. Workflow Management: Modeling Concepts, Architecture and Implementation. International Thomson Computer Press, September 1996.
11. Michael zur Muehlen, Marta Indulska, and Gerrit Kamp. Business process and business rule modeling: A representational analysis. In EDOCW '07, pages 189– 196, Washington, DC, USA, 2007. IEEE Computer Society.
12. P. Musham, S. Singh, R. Bahal, and P. Tv. Visual SBVR. In Digital Information Management, 2008. ICDIM 2008. Third International Conference on, pages 676– 683, November 2008.
13. OASIS. Web Services Business Process Execution Language (WSBPEL). http: //www.oasis-open.org/home/index.php/.
14. M. Pesic and W. van der Aalst. A declarative approach for flexible business processes management. pages 169–180. 2006.
15. Ronald G. Ross. Principles of the Business Rule Approach. Addison-Wesley Information Technology Series, 2003.
16. Shazia W. Sadiq, Maria E. Orlowska, and Wasim Sadiq. Specification and validation of process constraints for flexible workflows. Inf. Syst., 30(5):349–378, 2005.
17. August-Wilhelm W. Scheer. Aris–Business Process Modeling. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1998.
18. C. Schlenoff, Gruninger M., F. Tissot, J. Valois, J. Lubell, and J. Lee. The process specification language (PSL): Overview and version 1.0 specification, 2000. NISTIR 6459, National Institute of Standards and Technology.
19. Helen Schonenberg, Ronny Mans, Nick Russell, Nataliya Mulyar, and Wil M. P. van der Aalst. Towards a taxonomy of process flexibility. In CAiSE Forum, pages 81–84, 2008.
20. Kuldar Taveter and Gerd Wagner. Agent-oriented enterprise modeling based on business rules. In ER '01: Proceedings of the 20th International Conference on Conceptual Modeling, pages 527–540, London, UK, 2001. Springer-Verlag.
21. Wil M. P. van der Aalst and Maja Pesic. Decserflow: Towards a truly declarative service flow language. In The Role of Business Processes in Service Oriented Architectures, 2006.
22. Wil M. P. van der Aalst and Mathias Weske. Case handling: a new paradigm for business process support. Data Knowl. Eng., 53(2):129–162, 2005.
23. Wil M.P. Van Der Aalst and Arthur H. M. Ter Hofstede. Verification of workflow task structures: A petri-net-based approach. Inf. Syst., 25(1):43–69, 2000.

5 EVESIM AND DIGITAL ECOSYSTEMS (SUAS)

The EvESimulator, or short EvESim, is a simulation framework designed for Digital Ecosystem (DE) platforms. The term simulation is often used interchangeably with emulation or even testing, because EvESim is capable to run on a DE and in parallel simulate a DE. Besides that, EvESim constitutes a collaborative platform for interdisciplinary research acting as a framework for understanding, visualising and presenting the social network concepts to contributors. Its development started back in the DBE and it was extended and adapted for the differentiated needs of the community as it was needed.

EvESim supports at the moment three groups of stakeholders, or in other words is targeted at three types of users. These can be identified when screening interest groups for interdisciplinary collaboration in the field of distributed architectures and social networks: 1) analysts, 2) catalysts, and 3) instance creators.

We refer to analysts as experts from social network analysis. The target stakeholders in this area have usually little or no programming skills but they have much knowledge as regards the analysis and interpretation of network data. Analysts have already well-established simulation frameworks and analysis tools at hand, but in some cases they run out of computing power or they need customized visualization and analysis functionalities beyond the level these current tools offer.

In order to implement customized simulation cases, analysts take advantage of simulation catalysts. Catalysts are maintaining and extending the framework. By implementing easy-to-use front-ends for the simulation and emulation of social networks, they extend on the one hand the present framework in order to allow analysts to run simulations on different distributed infrastructures. On the other hand they are exposing an interface for the instance creators for providing different infrastructures.

The so-called instance creators are providers for implementations of distributed service architectures. To give an example, this could be a provider for a distributed social network or a P2P network implementor. The interest of this group in the present framework might be to access data of existing social networks in order to emulate their infrastructure based on existing network topologies and behaviors.

In the following, a short generic example of the collaboration of analysts (SNA from IPTI), catalysts (SUAS) and instance creators (Flypeer implementation group at IPTI) for the emulation of a distributed document editing tool in OPAALS: Research and development of a medium-sized enterprise needs to develop a collaborative document editing tool that works on a P2P basis. The analyst evaluates the behaviour of potential users, collaborating on documents and papers. One existing research network is used for acquisition of real-world data as input for the simulation network. In the following, the peer-to-peer system is chosen as infrastructure (Flypeer), which is also used for the potential underlying infrastructure of the distributed collaborative document editing tool to be developed. Now the enterprise can run emulations of the document editing tool based on social network analysis data on top of different peer-to-peer systems in order to decide on the best

configuration and optimum infrastructure.

A more detailed description of this example and an extensive description of the EvESim architecture, use-cases and visualisation capabilities can be found in *D10.20 Emulation and testing of OPAALS P2P infrastructure by utilisation of EvESim*. This report acts as the final report on EvESim in OPAALS and gives a detailed description of its functionality. The current built of the framework as well as HOWTOs for installation and configuration can be downloaded at http://evesim.org. Here is also further reading material available, which includes HOWTOs and examples for Flypeer services. The codebase of EvESim can be found in the sourceforge SVN repository at http://sourceforge.net/projects/evesim/.

6 DISTRIBUTED FILE SYSTEM (WIT)

6.1 OBJECTIVE

The objective of this task is to design a system that allows users to store personal files securely in the cloud of nodes that make up the OPAALS network. This system would use other components developed in the OPAALS project including Flypeer, IdentityFlow and TrustFlow, and would serve as a demonstrator for these technologies.

6.2 THE PROBLEM OF DISTRIBUTED STORAGE

Many attempts have been made to develop Peer to Peer Distributed storage systems, and these can be roughly divided into two camps, the commercial systems which use Centralised control systems such as WUALA, and the purely P2P systems such as freenet[http://freenetproject.org/]. What they share in common is that users contribute a portion of disk space and bandwidth to storing other peoples files, in return they can make use of other peoples drive space.

P2P technology has been shown to be very efficient for the purposes of File sharing, that is to distribute a file from one user to many others, without any encryption. In the file sharing scenario one user can upload a file much more cost effectively than they could using a centralised approach. The bandwidth costs are spread between those downloading the files, and this cost is accepted by those downloading as a necessary cost to receive the file. The leading example of a P2P file sharing protocol is Bittorrent, representing roughly half off all internet traffic (IPOQUE). When we look at the problem of storing private files in a P2P system, the same benefits of sharing do not apply. A user who stores an encrypted file for another user, has no access to that file and receives no direct benefit for storing that file. Of course the user does get the benfit of being able to store his own encrypted files.

6.3 REPLICATION AND ERASURE CODES

A file stored in a network of user nodes must be spread across multiple nodes, either by means of over sampling (erasure codes) or simple replication depending on file size. This means that any file will be available even when some nodes are offline. For large files, over sampling of the file means that the original file can be recreated from a subset of fragments. The downside of these methods is that overall more space is required to store a file, and this in turn means more bandwidth is required to write these redundant copies or fragments to multiple nodes. The number of copies required is dependent on the uptime of the nodes and the level of availability required.

Node requirements

Each user node will need to provide a disk space equivalent to multiple times(>5) their required storage space. Bandwidth is required for the constant replication and rebalancing of files as nodes leave (unavailable for certain period) or join. Bandwidth is also required to serve the file or fragments to the files owner when requested.

For storage of files node stability is vital with a server that is permanently online of most value, and a machine that appears sporadically adding little value to the network.[10] (

One problem with a pure P2P system was that if each user were only to be able to use the service in proportion to the resources they contributed, then the service would not be of high enough quality (availability and bandwidth) for many consumers. This required that some backbone servers (or superusers) must be provided to compensate for users effectively leaching on the network. The consumers would have to contribute to the running costs of these backbone servers. Without these super nodes new users to the system would have difficulty storing files until they have built enough trust with other users. There is trade off between a completely open system, and a system that provides a minimum level of performance.

6.4 DESIGN

The Design of The DFS takes into account the pre-existing components of OPAALS and the overall philosophy of avoiding centralised control mechanisms. This complicates the design of a Distributed File system over simpler systems with centralised indexes and control servers.

The DFS design is dependent on a Network (or at least one), of trusted and relatively stable nodes with which a user can communicate. These nodes expose a Service interface that allows users access a personal file Directory. They have the ability to Upload, download and List files in their directory. Files are encrypted before they are uploaded and the key is not shared.

In the DFS, File Replication is carried out by pulling rather than pushing files. A node contacts another trusted node and requests its latest files that it needs replicated. This means only nodes that have a high uptime and trust levels will have high levels of file replication. The downside is that every file resides on a single node until it is requested by another. They key point of this approach is to create a system where good behaviour by a node gives a better service quality to that user, and avoids the situation where bad users can push files to nodes, and lie about their own behaviour.

The DFS was designed to support the saving and retrieving of files, and therefore modifying an existing file means saving a new copy. Supporting File modification by sharing the changes or difference between copies was determined to be to complex an issue to address in this task.

6.5 IMPLEMENTATION

THE DFS Implementation was primarily implemented as a test and demonstration service for the Components on which it is built(Flypeer,IdentityFlow,TrustFlow) and is just a prototype solution..

The DFS service uses the Flypeer service infrastructure, which is itself built upon the JXTA framework for P2P services. Each operation is a separate Flypeer service, allowing the composition of DFS operations with other services in complex Transactions. The 3 main services are listed below

GetFile: This service allows a user to retrieve a file with a given name and location

Listfiles: This operation will list all files belonging to the current user. The files are described in an XML file giving full folder structure.

AddFile: This operation will add a file(or folder) to a specific location on a users filespace.

6.6 FILE SIZE LIMITS

There is a limit on the size of file that can be added to the DFS due the limits of the underlying infrastructure . A method of splitting up a large file into multiple chunks using the Bittorrent approach was investigated, and this was developed into a alternative method up moving large files. However this approach was limited by the NAT and Firewall issues built specifically to prevent this behaviour (e.g. File sharing). A Bittorrent client that used JXTA was developed, however this solution put a lot of strain strain on JXTA relay nodes when nodes behind firewalls were communicating. If you wish to move large files, a direct connection is preferred; otherwise others can incur bandwidth costs if they are used to relay large amounts of traffic.

6.7 ENCRYPTION

All user files are encrypted before they leave the users own computer, and the key is a pass phrase not shared with any other parties. The javax.crypto package and the DES standard are used for the encryption process. The security level created by the encryption is dependent on the length and complexity of the pass phrase chosen by

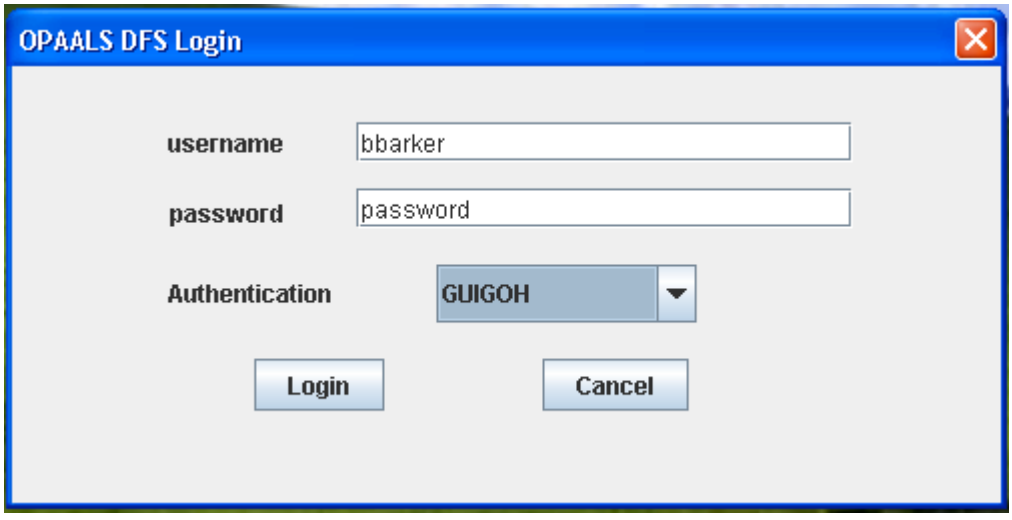
the user.

6.8 Service Discovery and Trust

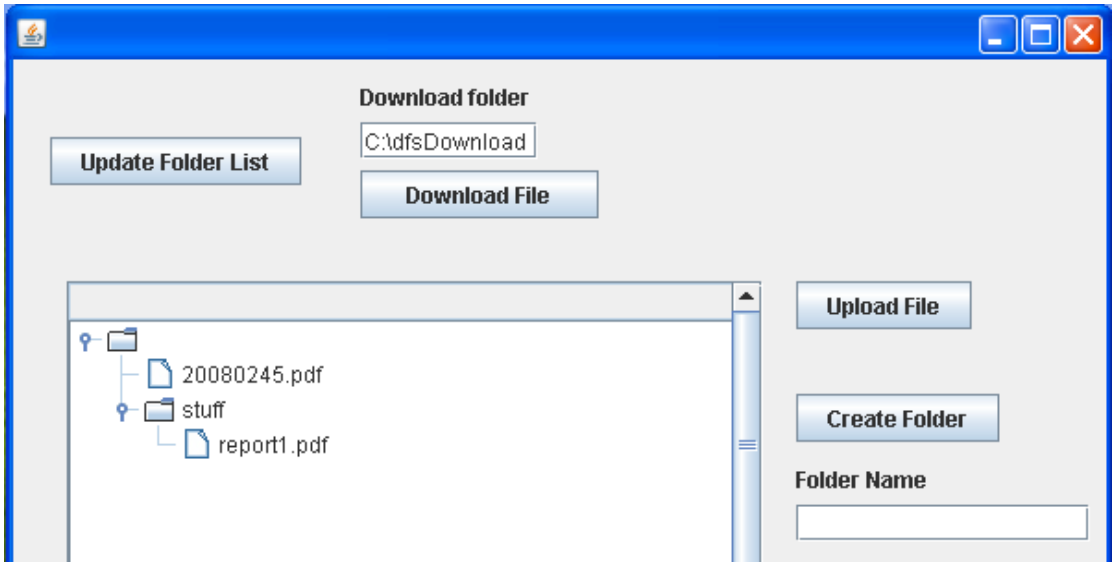
Flypeer uses a simple system for Service naming and has limited abilities to search fro services. In order for a user to reliably connection a Node which will contain the users personal files, we use the OPAALS trust system TrustFlow (http://sourceforge.net/projects/trustflow/). This ensures that encrypted files are shared among those nodes most trusted, and are therefore the most reliable source when looking to retrieve the files.

6.9 The Client Interface

A Graphical interface for the DFS was creates to allow easy access to the user files. It uses a simple username and password to login to know Identity provider (Using the IdentityFlow sub-project). This Identity is used to authenticate with the DFS service via Flypeer. There is a delay in the current implementation of the Identity verification which adversely affects the usability of the client.



After logging in the client automatically connects to the DFS service and lists the files currently in the users directory. In the above image the user has selected to authenticate against the Guigoh service, other alternatives including LDAP and XMPP (e.g. Sironta)


Download folder
C:\dfsDownload
Update Folder List
Download File
Upload File
20080245.pdf
stuff
report1.pdf
Create Folder
Folder Name

7 OPTIMIZATION BY PROXY (SURREY)

In this paper, we describe the pattern of optimization by proxy, which can be found in most large-scale algorithmic systems. The pattern occurs when a computationally limited optimizer uses a proxy property as a shortcut indicator for the presence of a hard-to-measure target quality. However, when intelligent actors with different motivations are involved, the existence of the algorithm reifies the proxy into a separate attribute to be manipulated with the goal of altering the algorithm's results. What follows is an arms race of continuous updates by the algorithm designers and continuous interrogation and manipulation of the proxies by the optimized entities. We examine the types of distortion that can be introduced, focusing on two web-relevant examples of optimization by proxy, and discuss potential alternatives.

7.1 INTRODUCTION

The first thing a newly-hatched herring gull does is to peck on its mother's beak, which causes her to give it its first feeding. Puzzled by this apparent automatic recognition of its mother, Dutch ethologist and ornithologist Nikolaas Tinbergen conducted a sequence of experiments designed to determine what precisely it was that the newborn herring gull was attracted to [1]. After experimenting with facsimiles of adult female herring gulls, he realized that the beak alone, without the bird, would elicit the response. Through multiple further iterations he found that the characteristics that the newborns were attracted to were thinness, elongation, redness and an area with high contrast. Thus, the birds would react much more intensely to a long red stick-like beak with painted stripes on the end than they would to a real female herring gull. Objects of this class, able to dominate the attention of an animal away from the original target were later called 'supernormal stimuli' and have been commonly observed in nature ever since.

This is interesting from a web science perspective as a similar pattern appears when algorithms intended to make optimized selections over vast sets of candidates are applied on social systems. The algorithms' designers substitute the problem of measuring a characteristic hard to quantify such as relevance or quality, with a proxy which is computationally efficient to measure. As long as the fundamental assumption that the proxy indicates the presence of the desired property holds, the algorithm performs as intended, yielding results that to the untrained eye seem 'magical'. Google's PageRank, in its original incarnation [2], trying to optimize for page quality, optimizes for it indirectly, by analyzing the link structure of the web. As the web has grown, such algorithms, and their scalability characteristics, have helped search engines dominate navigation on the web over previously dominant human-curated directories. Similar algorithms are used in social news sites, recommendation engines and fraud detection systems.

7.2 EFFECT OF OTHER ACTORS

When there is only a single party involved in the production, filtering, and

consumption of results, or when the incentives of the relevant group of actors are alligned, such as in the herring gull case, the assumption of the algorithm remains stable and its results remain of high quality.

In distributed systems such as the Web however, when the proxy is in the control of intelligent actors that can manipulate the proxy attribute, and they stand to benefit from distorting the results of the algorithm, then the existence of the algorithm itself and the motive distortions it creates alter the results it produces. In the example of PageRank, its early results owe to the fact that the link structure they crawled was effectively a byproduct of the buildup of the web. By bringing it to the attention of website owners as a distinct concept however, they have incentivized them to manipulate it separately, through techniques such as link farming and keyword stuffing, effectively making the altered websites act as supernormal stimuli for the algorithm. In this sense, the act of observation and the publication of results alters that which is being observed. What followed was an arms race between the algorithm designers and the external agents, each trying to affect the algorithm's results in their own preferred direction, with the algorithm designers controlling the algorithm itself and the external agents controlling part of the data it is applied on.

7.3 DISTORTIONS INTRODUCED

One approach by algorithm owners is to keep secret the operation of the algorithm, creating uncertainty over the effects of manipulation of the proxy. This is effectively security by obscurity and can be counteracted by dedicated interrogation of the algorithm's results. In the case of PageRank, a cottage industry has formed around Search Engine Optimization (SEO) and Search Engine Marketing (SEM), essentially aimed at improving a website's placing in search engine results, despite the secrecy of the algorithm's exact current operation. While a distinction can be made between black-hat and white-hat practitioners, the fact remains that the existence of these techniques is a direct result of the existence of an algorithm that optimizes by proxy. Another trend related to the mechanics of search engine algorithms is what is called Googlebombing, where typically a grass-roots movement decides to link to a certain page using a predetermined keyword in the link text. This results in searches for that keyword returning the targeted site. The more successful the campaign, the higher in the results the target website appears. Googlebombing is used as a form of Internet graffiti, creating an artificial association between a keyword and a website. As a response to the various distortions, algorithms are enriched with heuristics to identify them. This, as the arms race progresses, converges to the point where the proxy approaches the original target more and more, and hence the external actors are forced to simulate the algorithm's target quality to the point where "sufficiently advanced spam is indistinguishable from content".

This of course would hold only if processing power were not an issue. However, if performance was not an issue, far more laborious algorithms could be used to evaluate the target attribute directly. Optimization by proxy, being a computational shortcut, is only useful when processing power is limited. In the case of the Web

search, there is a natural asymmetry, with the manipulators able to spend many more machine- and man-hours to optimization of the result than the algorithm can spend judging the quality of a single item. Thus, algorithm designers can only afford to tackle the most broadly-occurring and easily distinguishable forms of manipulation, while knowingly ignoring the more sophisticated or obscure ones appearing at the long tail.

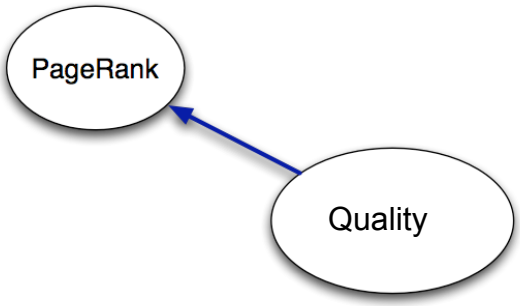


FIGURE 1. PAGERANK'S ROLE AS A PROXY MAKES IT AN INDICATOR FOR THE UNDERLYING CAUSE.

7.4 FAKING IT – A BAYESIAN VIEW

We can obtain a little more insight by considering a simple Bayesian network representation of the situation. A key guide to algorithm design is the identification of some measure that intuitively will be highly correlated with quality. In terms of PageRank in its original incarnation, the reasoning is as follows. High quality/highly relevant web sites will attract attention from peers who are also contributing related content. This will "cause" them to link into the web site under consideration. Hence if we measure the number of highly ranked web sites that link into it, this will provide us with an *indication* of the quality of that site.

The key feature is that the causal relationship is *from* the underlying quality (relevance) *to* the indicator that is actually being measured (see figure 1).

This simple model raises a number of issues with the use of proxies. Firstly, one needs to be aware that it is not just a matter of designing a smart algorithm for quantifying the proxy. One also needs to quantify the strength of association between the proxy and the underlying concept.

Secondly, unless the association is an extremely strong one, this makes use of the proxy a relatively "lossy" test for the underlying concept. In addition, if one is going to use the proxy for decision-making, one needs some measure of confidence in the value assigned to the strength of the relationship – a second-order probability that reflects the level of experience and consistency of the evidence that has been used to determine the strength of the relationship.

Finally, one needs to be aware of the consequences of performing inference in the reverse causal direction. In modeling this as a Bayesian Network, we would use the conditional probability distribution p(PR | Q) as a measure of the "strength" of the relationship between cause and proxy (where "PR" is a random variable representing the value of PageRank, and "Q" is a random variable representing the value of the

(hidden) cause, Quality). Given a particular observation of PR, what we need to determine is p(Q | PR) – the distribution over Quality given our observation on the proxy. This (in our simple model) can be determined through the application of Bayes' rule:

$$p(Q \mid PR) = \frac{p(PR \mid Q)p(Q)}{p(PR)}$$

What this is reminding us of us that the prior probability distribution on Quality is a major factor in determining its posterior following an observation on the proxy. Remember the textbook "clinician's fallacy" – a positive test for a rare disease actually leaves a low likelihood that the patient is suffering from the disease through the dominance of the prior (e.g, [6]).

Can this kind of model help us? Potentially it can if we are able to obtain some form of feedback or evaluation of the performance of a proxy. Well known learning algorithms can be used to update the probability distributions in such a network (see [7] for a simple introduction, or [8] for a much more technical review). In addition, one can place monitors on the rate of convergence of the probability distributions to see if there is a statistically significant disagreement between the predictive power of a proxy and the actual evaluated outcome in terms of quality (e.g. [9]). We need to perform a little more analysis but this ought to provide a relatively effective basis upon which to discount a proxy should some form of "stuffing" become prevalent for a particular proxy.

One other way forward is to explicitly model the use of multiple proxies using a Bayesian approach. To do this properly, one would need to include in the model a representation of the distributions (second order probabilities) over possible outcomes, in order to be sure that proxies that are being faked are suitably discounted. However, this does then turn the relative weakness of the proxies as predictors into a strength, as the impact of the fakes will be relatively rapidly discounted, whilst the remaining proxies can collaborate to provide much higher confidence in a prediction of Quality than any one on its own.

As should be clear from the age of many of the references in the above, none of this is new material. However, we do believe that it is important to bring this material into scope when discussing algorithms for optimizing selections.

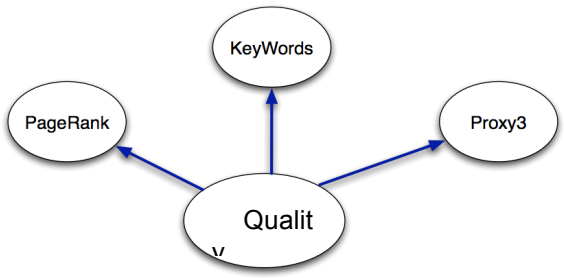


FIGURE 2. BN MODEL USING MULTIPLE PROXIES. SUCH A MODEL WILL NATURALLY ADAPT TO USE THE MOST INFORMATIVE PROXY IF IT CAN BE COMBINED WITH A FEEDBACK PROCESS. NOTE ALSO THAT THE INDICATORS/PROXIES ARE MARGINALLY DEPENDENT, BUT CONDITIONALLY INDEPENDENT GIVEN THE TRUE VALUE OF QUALITY. THIS AGAIN

EMPHASIZES THEIR RELATIVE WEAKNESS AS PREDICTORS IF USED IN ISOLATION.

7.5 PATHS TO A SOLUTION

Whereas algorithms have been introduced to the web as a way of taking the weight off humans, eventually we still have to supervise the algorithms, to continuously align the internal assumptions with the execution context, and interpret results appropriately. This, however, still means that many false positives and false negatives are required until a distortion is noticed by the supervisors, and there is a computational limit to the countermeasures that can be applied. Potential improvements include the consideration of costly signals[3], the use of communities with better aligned incentive cultures that encourage quality (such as Wikipedia), or the use of casual games as a way to provide vast training datasets for the algorithms [4]. However, as long as algorithms attempt to construct objective answers with a less than precise internal model of what constitutes a good answer, and as long as there is a strong enough incentive for adversaries to skew the results, the arms race will continue.

Looking at newer models that cover the above discussed are utilised to cover use cases of web search and news, often when traditional means fail, a radically different model seems to be emerging. To see it we need to look beyond the services that traditionally offer these services. Relative newcomers such as Q&A sites, social networks, and social search platforms provide alternative avenues for users to find news and answers.

Focusing on the distribution of news items in social networks such as Facebook and Twitter, users share the items that are important to them with others that follow them. Re-broadcasting features such as Twitter's 're-tweet', first appearing as a community norm and then formalized in the platform, allow diffusion to happen much more naturally. This is a radically different model than the top-down approach used by traditional media and even social news aggregators such as Digg and Slashdot. When results are determined for each user according to the sources of authority that the user trusts, and with an appropriate feedback loop for returned results, actors who contribute distorted information are inevitably sidelined, effectively rendering these distributed networks immune to optimization by proxy. This natural resistance to spam is in stark contrast to the huge amount of effort that goes into avoiding it in centralized aggregators, with varying results.

What this indicates in terms of optimization by proxy, is that a potential antidote is distributing the filtering logic. In the centralized single-algorithm paradigm, filtering is done through a single though elaborate model, uniformly applied to billions of pages. In its distributed counterpart, each user filters the news items they come in contact with, and shares them, with the results of the sharing being felt on the users' own reputation. Effectively there are millions of algorithms applied on the same data, without a large single target for manipulators to aim at. What's more, the judges, instead of computationally constrained algorithms, are actually well-motivated humans. In search, new entrants such as Aardvark, recently acquired by

Google, have taken such a route.

These efforts are relatively new, and there is no clear evidence that any of them has been built with avoiding the effects optimization by proxy in mind, or even to serve the use cases we examined, as their stated purpose is different. To that extent, centralized and decentralized features often coexist, with the services being vulnerable to optimization by proxy to the extent that they introduce vulnerable features. One such example is Twitter's Trending Topics feature which attempts to algorithmically determine which topics are currently talked about throughout the network. As soon as a topic is uniformly broadcasted as 'trending', users are drawn to viewing the ongoing discussion, and spammers are rewarded for adding to the discussion keyword- and hashtag-laden advertising messages. This continues until the advertising stifles actual discussion, real users abandon the topic, and it stops trending. The cycle then restarts with a new set of trending topics. Q & A sites such as stackoverflow.com present another interesting hybrid, combining social dynamics and voting with the goal of creating answers of objectively high quality.

In all these examples, to the extent that distribution of filtering is applied, the trade-off for this new-found effectiveness is the increased burden to initially bootstrapping the network, as well as giving up the common point of reference that common result sets provide. However, with multiple people or algorithms to compete with, malicious content creators have less of a single target to manipulate, and therefore are forced to chose between getting ever closer to quality content in order to drive traffic, or giving up, both positive outcomes for the users.

7.6 CLASSIFYING THE APPROACHES

At this point, it may be helpful to pull together and classify the examples discussed so far, as to make the distinctions clear (see Table 1). The manual approach is the oldest one, with a group of human editors curating results for a much larger audience. The algorithmic approach is more modern, and in certain areas, such as search, has largely displaced the manual approach. It enjoys the benefits of greater reach and greater responsiveness, side-effects of its inherent scalability. However, we argue that it is vulnerable to a systemic fault in optimization by proxy. The effects of this weakness are not immediately obvious but tend to degrade results over time. A potential solution, by way of drastic paradigm shift, can be seen in new distributed alternatives. Here, distribution does not apply to the actual infrastructure, but rather to the filtering logic, which is handled directly by the end users. Each users' decisions have only a local effect, depriving attackers of a large and inflexible target to aim at.

TABLE 1. CLASSIFYING THE VARIOUS APPROACHES TO FILTERING

	Search	**News**
Manual	Human-curated Directories (Yahoo Directory, Alta Vista)	Traditional News Media (newspapers, TV)
Algorithmic	Algorithmic Search (Google PageRank)	Social News Aggregators (Slashdot, Digg,

		Hacker News)
(Hybrid) Distributed	Social Search, Q&A (Aardvark, Yahoo! Answers, Stack Overflow)	Social Networks (Twitter, Facebook)

7.7 CONCLUSION

While most teams that work in large-scale algorithmic systems are intimately aware of the characteristics of this pattern, we have not seen it being explicitly identified and analyzed before at this level of abstraction. Our aim in this paper has been to surface the concept as a stand-alone term, offering a common point of reference for its many instances, and hopefully sparking further discussion around its characteristics, proposed countermeasures, and fields of applicability. We have focused on the web-relevant examples of search and news, but as illustrated by the herring gull example in the introduction, it can be seen almost everywhere. Employers judging potential employees by the name of the university they attended, companies rewarding staff, especially in sales, with a productivity bonus, even academic funding bodies allocating funds according to bibliometrics, are only a few examples, and are all vulnerable to optimization by proxy. While highly experienced implementors in all fields get to know the pattern through experience, newer entrants have to rediscover its ins and outs through new real-world failures. By exposing this common pitfall, we hope to pave the way for future work in managing the problem for those who have no other option but to face it. One potential avenue may be a toolkit that packages the most successful strategies for applying optimization by proxy, monitoring its effectiveness, and potentially recognizing when it is no longer an effective strategy altogether.

7.8 REFERENCES

[1] Tinbergen, N., Perdeck, A.C., 1950, “On the Stimulus Situations Releasing The Begging Response In The Newly Hatched Herring Gull Chick”, Behaviour, 3, 1-39.

[2] Page, L., Brin, S., Motwani, R., Winograd, T., 1998, “The pagerank citation ranking: Bringing order to the web”, Technical Report. Stanford InfoLab.

[3] Zahavi, A., Zahavi, A., 1997, “The Handicap Principle: A Missing Piece of Darwin’s Puzzle”, New York: Oxford University Press.

[4] von Ahn, L., “Games With A Purpose”. IEEE Computer Magazine, June 2006. pp 96-98.

[5] Jøsang, A., Hayward, R., Pope. S., “Trust Network Analysis with Subjective Logic”, Proceedings of the Australasian Computer Science Conference (ACSC'06), Hobart, January 2006.

[6] Elstein, A.S. and Schwartz, A, “Clinical problem solving and diagnostic decision making: selective review of the cognitive literature”, British Medical Journal, 2002;324:729-732.

[7] Krause, P.J., “Learning Probabilistic Networks”, Knowledge Engineering Review, 13: 321-351, 1999.

[8] Heckerman D., Geiger D. & Chickering D. “Learning Bayesian Networks: The Combination of Knowledge and Statistical Data”. Machine Learning, 20, 197-243, 1995.

[9] Spiegelhalter D.J. & Lauritzen S.L., “Sequential Updating of Conditional Probabilities on Directed Graphical Structures”. Networks, 20, 579-605, 1990.

8 AN SBVR FRAMEWORK FOR RESTFUL WEB APPLICATIONS (SURREY)

Competing visions have been jostling to define the long-term future of the Web: WS-* Web Services, the Semantic Web, and RESTful Web Services. This chapter presents the initial steps towards a Rule-driven, REST-based architecture for the Web that can enable use cases that the Semantic Web and WS-* communities require. The key enabling ingredient is the use of SBVR models as a media type for resource description that allows models to be exposed and consumed. With formal description of data, advanced scenarios such as inference, service composition, and transactions are feasible within an architecture that is backwards-compatible with today's Web.

8.1 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.

8.2 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, Semantic annotation	Predicate (RDF Triple)	Data analysis, Automated reasoning	RDF, RDFS, OWL, SPARQL
WS-* Web Services	Enterprise Systems	Governance, Reliability, Security	Service/ Procedure	Service Composition, Transactions	SOAP, WSDL, UDDI,
RESTful Web Services	Distributed Systems, Software Architecture, Web Development	Scalability, Evolvability, Loose coupling, Linkability	Resource	Web Sites, Feeds, Search Engines	HTTP, HTML, URI, ATOM

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.

8.3 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. 5. 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 under probation is registered for* at most three courses.
Grade A or B or C or D or F	Student *has* first name	
First name	Student *has* last name	
Last name	Student *is marked with* grade *for* course	
	Module *is available for* course	

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.

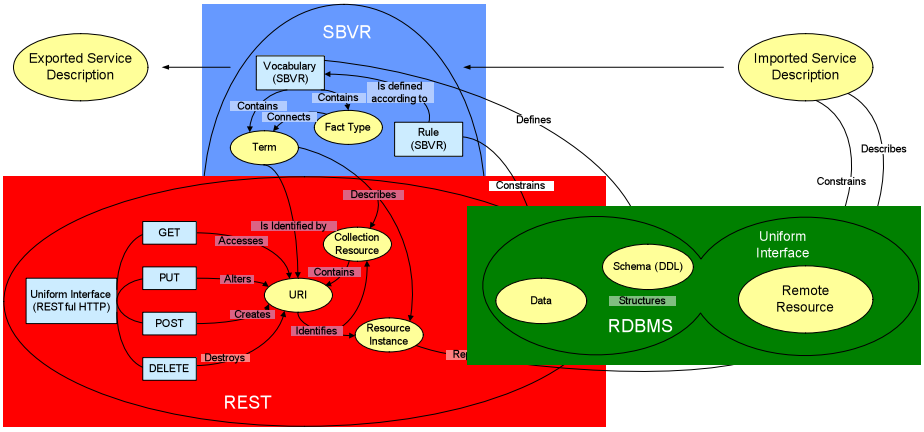


Fig. 6.

8.4 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).

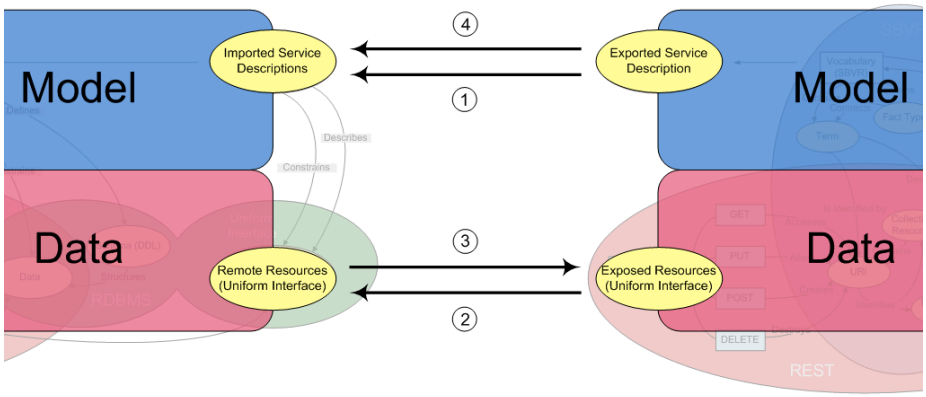


Fig. 7.

The exact mechanism by which inconsistencies are noted in the behaviour of the resource is shown in figure 8. 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.

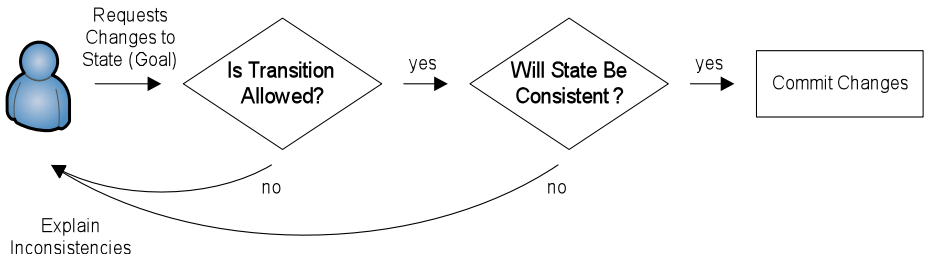


Fig. 8. The meta-process of interaction between a client and the internal logic of a service.

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.

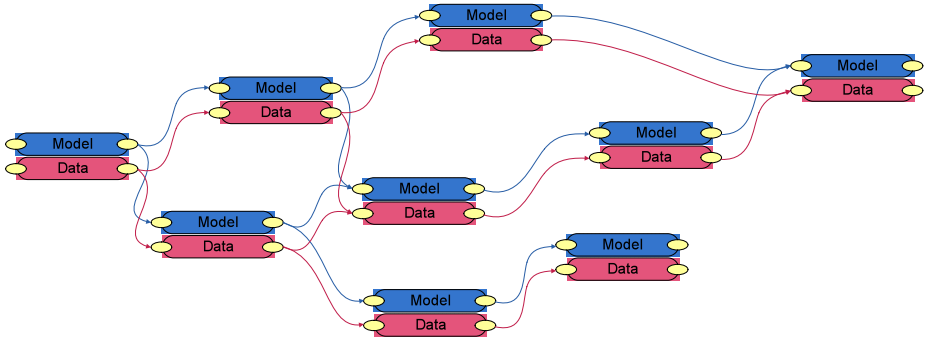


Fig. 9. The Web of Models overlaid on today's Web of Data. As data updates flow from clients to servers, updates in the models propagate in the reverse direction.

8.5 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.

8.6 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.

8.7 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)