OPAALS PROJECT

Contract n° IST-034824

WP3: Autopoietic P2P Networks

Del3.7 – Biological and evolutionary approaches for P2P systems and their impact on SME networks

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

Contract Number: IST-034824

Project Acronym: OPAALS

Deliverable N°: D3.7

Due date: March 2009

Delivery Date: July 2009

Short Description: This report covers three main aspects: 1) Gradual elaboration of transaction scripts, 2) Modeling network dynamics for simulation and 3) Integration of interdisciplinary approaches in EvESim and the simulation of an basic semantic search

Author: SUAS (Thomas Kurz, Thomas J.Heistracher), IPTI (Fernando Colugnati), UniS (Amir Reza Razavi, Sotiris Moschoyiannis)

Partners contributed: SUAS, IPTI, UniS

Made available to: OPAALS Consortium and European Commission Versioning

Versioning

Version	Date	Name, organization
0.1	14/06/2009	(first submission) SUAS, IPTI, UniS
1.0	21/06/2009	Submission for internal review SUAS, IPTI, UniS
2.0	24/08/2009	Final submission SUAS, IPTI, UniS

Quality check

Internal Reviewers: Niall Brennan (LSE), Jukka Huhtamäki (TUT)

Dependencies:

Achievements*	Use of new method for Social Network Analysis and Simulation. Reasoning about local interactions to ensure only the desired behaviours arise – this is important since the local interactions give rise to temporary networks which are the main building block of the overall interconnected P2P network.
Work Packages	WP3, WP10
Partners	SUAS, IPTI, UniS
Domains	Computer Science, Social Science
Targets	Inferring global properties from local interactions so that the P2P network is formed of smaller and simpler networks, and fusing these with evolutionary approaches to reflect the dynamic topology of the underlying P2P network.
Publications*	Fernando A.B. Colugnati – IPTI: Dynamic social network modeling and perspectives in OPAALS frameworks. 2nd OPAALS Conference, Tampere – FI Dynamic social network modeling – a Forum use case, International Conference on Applied Social Modeling and Simulation 2009, Paris Dynamic modeling of social network – a forum case study, 5th UK Social Network Conference 2009, London S. Moschoyiannis, A. Razavi, P. Krause. Transaction Scripts: Making Implicit Scenarios Explicit. In *Proc. ETAPS 2008 – FESCA'08*, ENTCS, Elsevier, 2009. *In press*
PhD Students*	None
Outstanding features*	Migration of Simulation Framework to the OPAALS P2P infrastructure under construction.
Disciplinary domains of authors*	T. Kurz (Information Technologies, Software and Systems Engineering, Interpersonal Communication) T. Heistracher (Information Technologies, Software and Systems Engineering, Biophysical Modelling) Fernando A.B. Colugnati (Statistics and Social Network Analyst)

	Sotiris Moschoyiannis (Mathematics, Computer Science, interaction models, formal semantics, behavioural analysis) Amir Razavi (Computer Science, Distributed Systems, Transactions)

The information marked with an asterisk (*) is provided in order to address Recommendation n. 4 from the Year 2 review report

Contents

Executive summary

This report covers cross-disciplinary aspects of biological and evolutionary approaches for P2P systems and explains ways to improve the impact of this research on SME networks based on the distributed Evolutionary Environment Simulator (EvESim).

A formalisation of a coordination model for distributed transactions is provided which constitutes a building block for autopoietic P2P networks. By help of this formal framework, the coordination of concurrent and distributed service invocations between coordinator components involved in a long-running transaction in a digital ecosystem can be described.

In addition, an actor-oriented model for the description of the co-evolution of network structures is provided, thus enabling a bottom-up perspective from micro changes (actor tie selection) to macro changes (change in network topology). In the co-evolution model, many of the objective functions are related to clustering, such as transitivity effect, transitive triplet and preferential attachment.

Finally a partial implementation of these findings in the current EvESim framework is outlined. The architecture and infrastructure of this distributed simulation framework is described together with the core components that themselves are implemented as infrastructure services. Currently two different service containers (Soapod and Servent) are supported by EvESim. Real-world topology information can be imported to EvESim. For better monitoring and demonstrating the co-evolutionary development of the network structure and the fitness of the services mediated, a visualisation based on Google Earth is provided. First simulation results based on a real-world social network topology from Brazil demonstrate the importance of service migration for semantic search performance and encourage the concept of '*knowledge services*' that is intended to foster creation, usage, and exchange of knowledge in dynamic digital societies.

1 Introduction

The following document tries to set the ground of a more integrated understanding of theoretical research and practice-oriented emulation of the OPAALS infrastructure. One of the main challenges of this report and the related work is the diverse understanding of terms, models and approaches in the research disciplines as well as in the related application. This report still provides a considerably separated view on the approaches, but at the same time it shows how these approaches will be merged and combined in the EvESim simulation framework. The reason for keeping the approaches separate from each other is that we intend to give an in-depth report first and then show the cross linkages or mixing points in Chapter 4 and the Conclusion.

This deliverable falls within the realm of Task T3.8 in WP3, which among other things aims to simulate the behaviour of the transaction model and experiment with the effects of the local interactions - between nodes engaging in a transaction - on the underlying network topology. We have seen in previous deliverables of WP3 and elsewhere that transactions are the main building block for the OPAALS P2P network as they result in Virtual Private Transaction Networks (VPTNs), as described in detail in Deliverables D3.2, D3.3, which in turn give rise to an overall P2P network that comprise interconnected VPTNs through the use of the design concept of Dynamic Virtual Super Peers (DVSPs).

In Task T3.8 we have set out to simulate various modes of interaction in terms of service invocations that take place sequentially, in parallel (concurrently), or there is choice between services to be called. We have seen in Deliverable D3.2 and D3.3 that such information is captured in the so-called *transaction scripts* that are built for a given *transaction context*. It is important that the intended behaviour of a transaction, in terms of the underlying service interactions, is captured properly. Chapter 2 of this deliverable provides a technique for refining the specification of a long-running transaction so that only desirable scenarios are possible when the

transaction actually executes. We focus on how transactions are set up, mostly in terms of the complex interactions involved in their execution, aiming to ensure that the resulting behaviours correspond to the intended interactions only. The implementation aspects of the formal framework we describe in Chapter 2 can be found in [1] which includes the full XML schemas, the corresponding parser and a few examples to demonstrate the approach. The work described in Chapter 2 has been published in ETAPS 2008 [2] and preliminary ideas have also appeared in IEEE-DEST 2008 [3].

Dynamic social modeling is introduced, bringing to this problem a probabilistic approach. In this model, network evolution is thought of as a bottom-up process where the emerged topology is originated by micro-steps of selection and adaptation of the existing node, given their net neighborhood.These two processes can be modeled separated, although not independently, permitting to assess the influence that selection has in adaptation, and vice-versa. Also, this modeling can provide a different way to simulate networks, since it allows a better control over some network closure motifs, giving specific loads to them, so that final topology can really achieve the expected scenario of the infrastructure. Further, some other parameters can change over time (steps of simulation) providing an important tool to simulate interventions of any kind in the network.

Following the corresponding task description of T3.8, with EvESim we focus for the moment on pre-defined topologies. Partner IPTI provided data from a research community in Brazil in order to set up the topology of a real-world researcher network. As a preparation for future tests of the semantic search component and a broad reusability of the example simulation case, we decided to implement a simple simulation for semantic search on top of this data. The results can be found in Chapter 4.

The simulation cases of clustering and critical mass analysis performed earlier [4] were migrated to the new distributed EvESim framework and now show the sequential composition of services by the use of Genetic Algorithms. As the need for service compositions in OPAALS will be encapsulated in the infrastructure components, we desist from implementing additional sequential, alternative or interaction-based service compositions on the simulation level for now. Instead we focus on simple user interfaces for easy adoption by users from different domains and on additional visualization to ease dissemination of the DE paradigms to externals and on integration of social network analysis data and output. The currently developed P2P infras-

tructure – including transactions and so forth – will then be tested in phase III of the research project by running the already available simulation cases together with additional ones that relate to the Open Knowledge Space (OKS).

2 Gradual elaboration of transaction scripts

We have seen in previous deliverables of WP3 and elsewhere that the local interactions, in the form of long-running transactions are a core element of the DE architecture. Transactions are the main building block for the OPAALS P2P network as they result in Virtual Private Transaction Networks (VPTNs), as described in detail in Deliverables D3.2, D3.3, which is where Dynamic Virtual Super Peers (DVSPs) bubble up from. In this chapter we focus on how these transactions are set up, mostly in terms of the complex interactions involved in their execution, aiming to ensure that the resulting behaviours correspond to the intended interactions only. With respect to previous work on the coordination model for distributed transactions, this refinement of hte specification is done at the level of the corresponding transaction scripts, which were introduced in Deliverable D3.1 and developed in greater detail in D3.2. The transaction scripts capture the ordering of execution of the underlying services and is what is used to simulate transaction behaviour with EveSim as outlined in D3.3 and explored further in this deliverable.

The discussion that follows draws upon the work described in our paper [2] on refining transaction scripts that appeared in the European joint conferences on Theory and Practice of Software (ETAPS 2008), in the Formal Engineering approaches to Software Components and Architectures (FESCA'08) satellite event.

2.1 Aspects of transaction coordination

The adoption of the internet and the emerging paradigm of *computing as interaction* has fostered an environment where many distributed services are available through different components. This aspect of a digital ecosystem has increased the demand to automate business activities and workflows among networked organisations and has

increasingly placed focus on *long-running transactions* that correspond to conducting business activities involving a number of partners.

The specification of a transaction in this context typically comprises a number of activities which rely on the execution of several underlying services from different components of different providers, some of which may take minutes, or hours, or even days to complete - hence the term *long-running* transaction. This requires the orchestration of the participating components and the underlying service executions as well as the the provision for compensating mechanisms which in case of a failure (network/platform disconnection or delay, service unavailable) make it possible to undo parts of the transaction that have actually already happened. We will not be concerned with recovery management in this chapter - tjhis aspect of the work has been described in Deliverable D3.2 and subsequently published in [5] and elsewhere.

In this chapter we will be concerned with managing the dependencies that arise between services of concurrent and distributed components within a transaction, aiming to get a thorough understanding of the behaviour patterns the sequences of service invocations should exhibit to increase confidence in a successful outcome. Dependencies may exist due to the required ordering on service invocations (e.g. book a hotel only after booking the flight) or due to sharing of data (one service uses the results of another).

Current transaction models targeting web services such as the *Business Transaction Protocol* (BTP) [6] and *Web Services Transactions* (WS-Tx) [7] do not consider a formal model for the coordination of the services involved in the execution of a transaction. This makes it difficult to eradicate *emergent* behaviour - that is, behaviour not intended but resulting from the complex interplay of the interactions themselves, e.g. race conditions. For this reason both BTP and WS-Tx seem to be geared towards centralised control (using the WS-Coordination [7] that requires tight-coupling of the underlying services, which is against the basic premise of service-oriented computing (SOC) [8], and also some knowledge of the internal build-up of the participating components, which violates the local autonomy of the participants' platforms.

In previous work [9, 10] we have described a true-concurrent model, based on *vector languages* [11], for capturing the behaviour of components in terms of their interactions and this work has been largely reported in Deliverable D3.2. In this chapter, we use UML 2.0 sequence diagrams [12] to describe the service interactions between various coordinator components involved in a transaction and translate

them into the formal language of the vector-based description of behaviour.

We describe how reasoning against order-theoretic properties of vector languages can identify missing behaviours that infer additional scenarios. These were simply unthought in the initial design specification or indicate emergent behaviour, e.g. due to the subtle interplay between concurrency and nondeterminism in the interaction. Our approach is effectively used to elaborate the initial scenarios of interaction to more comprehensive ones, which are gradually refined to exclude emergent behaviour and include all desirable orderings of execution.

The remainder of this chapter is structured as follows. In Section 2.2 we describe how to model transactions, focusing on their structure and interactions. In Section 2.3 we present a formal language for describing interactions between coordinator components of a transaction. In Section 2.4 we show how the formal language can be used to reason about the dependencies between services of different coordinator components and uncover implicit scenarios of execution in the initial sequence diagram. The chapter finishes with some concluding remarks in Section 2.5.

2.2 Modelling long-running transactions

In this section we briefly describe the structure of a transaction and then show how service interactions between the coordinator components of the participants can be modelled using UML2.0 [12].

We have seen that long-running transactions involve interactions between multiple service providers, which need to be orchestrated in order to increase expectations of a successful outcome prior to deployment. The orchestration required needs to be performed in a way that respects the loose-coupling of the underlying services - a basic premise of SOA [8]. For this reason, each networked organisation (service provider / consumer) provides its services, and also requests services of others, through a *coordinator component* that manages the communication between different platforms and the deployment of the corresponding services. Without going into much detail, it can deploy (upon receiving a call) the services in its Local Service Repository and can make calls to services it requires from others components, within a given transaction, whose service descriptions (e.g. in WSDL) are listed in its Global Service Repository. The structure of a coordinator component is shown in Fig. 2.1.

In earlier work [13], reported in Deliverable D3.1, we have described a transaction

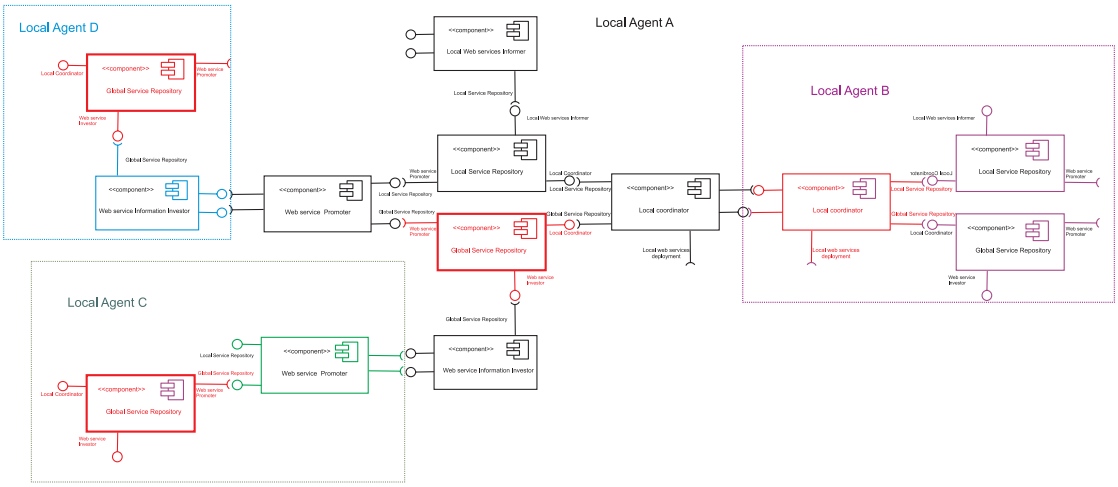

Figure 2.1: Coordinator components in a transaction

model for the distributed coordination of multi-service transactions. This is based on log structures, given in the form of directed graphs, which allow the coordinator component of each platform to only need to know about its own services and their dependencies on other components' services - in fact, it needs to know only what happens immediately *before* and *after* deployment of its own services. In what follows we provide a brief overview of this approach and place it within the context of the discussion to follow in subsequent sections. Further details can be found in Deliverables D3.1 and D3.2 and more succinctly in [13], [3].

A transaction in our approach is represented by a tree structure that captures nested subtransactions and exemplifies the local coordination that is required for the services involved to be performed in unison. Drawing upon the latest work on an extended service-oriented architecture for a business environment [8], we have considered five different composition types or *composers* which allow for various modes of service interaction in our model.

Sequential: execution of a service is dependent on the previous one; this composer handles both Sequential with Commit Dependency (SCD) and Sequential with Data Dependency (SDD) process-oriented service composition.

Parallel: the services are executed concurrently; composer handles Parallel with Commit Dependency (PCD), Parallel with Data Dependency (PDD) and Parallel without Dependency (PND) process-oriented service composition.

Sequential Alternative: the services will be attempted in succession until one

produces the desired outcome, as specified by some criterion (e.g. cost).
Parallel alternative: alternative services are executed in parallel and once a service produces the desired outcome, the rest are aborted.
Data-oriented: this composer handles data-oriented service composition and deals with released data items both within and outside a transaction.
Delegation: this composer allows (part of) a transaction to be delegated to another platform, e.g. to overcome bottlenecks or low bandwidth connections.

Further details on the composition types considered in our model can be found in [13]. A schema using Netbeans 6 for generating XML descriptions of transaction contexts, as specified by the composers used in the corresponding transaction tree, is given in Fig. 2.2.

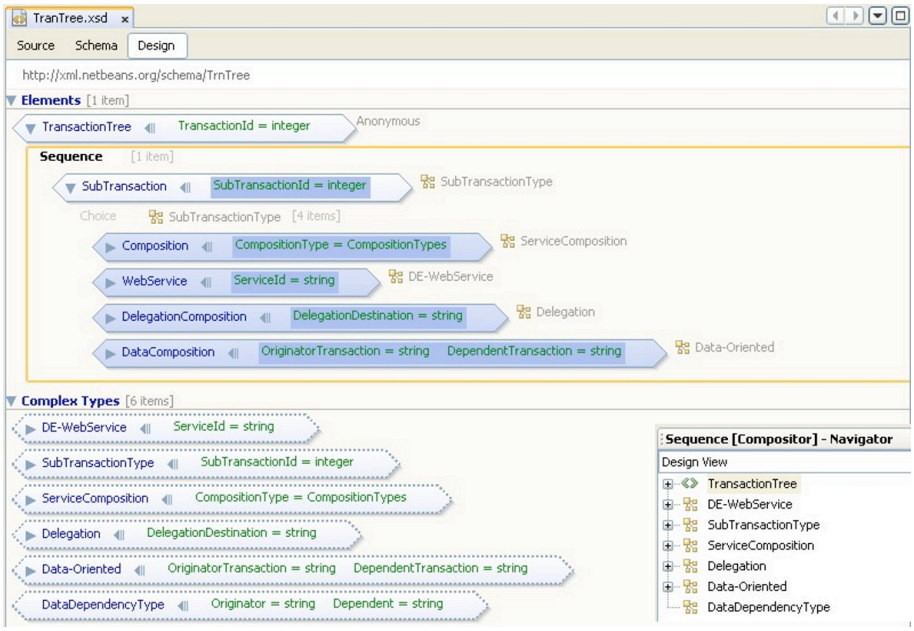

Figure 2.2: XML schema for describing transaction contexts

In this chapter we will be concerned with the sequential, parallel and sequential-alternative composition types. Fig. 2.3 shows a transaction tree with four basic services - $a1$ and $a2$ 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. For example, the tree specifies that the service calls $a2$ and $c1$ are children of a sequential composer and hence $c1$ can only happen after $a2$. The connecting lines on this composer indicate data dependencies between the

corresponding services' deployment. The corresponding XML description for the example transaction tree is derived from the schema of Fig. 2.2 and can be found following [1].

The scenario described in Fig. 2.3 has appeared in [13] and has been simplified somewhat here, but is still complicated enough to illustrate the key ideas. The service interactions implied by a transaction tree can be modelled using a UML sequence diagram. Fig. 2.4 shows the three coordinator components of the transaction and the messages (service invocations) exchanged between them during the execution of the transaction in our example.

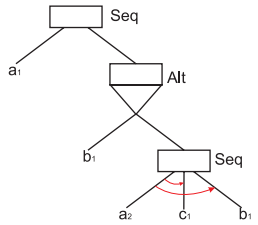

Figure 2.3: A simple transaction in a tree structure

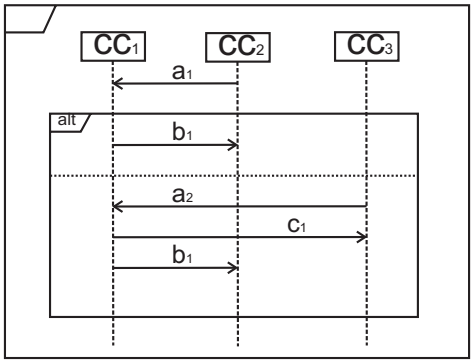

Figure 2.4: Behavioural scenario of a simple transaction

The transaction tree, together with the behavioural scenarios described in the corresponding UML model, define the *transaction context* which is used by the Initiator of the transaction this will be one of the coordinator components) to specify the different levels of compositions of the underlying services of the participating coordinator components. This can be represented in a context schema allowing it to be communicated to the Participants of the transaction. Such a schema is given in [1] along with the Netbeans source.

It can be seen that the behavioural scenarios, as given by the corresponding UML sequence diagram, determine the order of execution of the participating components' services. In the remainder of the chapter we will be concerned with a formal reasoning approach aiming to identify missing behaviours (if any) in the initial scenario-based specification of the transaction context. We shall see that missing behaviours may indicate emergent behaviour (e.g. due to race conditions) or scenarios of execution that were simply unthought in the initial design specification. First, we need to describe a transaction more formally in order to get a thorough understanding of the behaviour patterns the underlying service invocations should exhibit.

2.3 Vector languages for transactions

In this section we introduce a formal language for long-running transactions that captures the dependencies between service executions of the various coordinator components involved, and enables formal reasoning about the interactions to uncover hidden scenarios. In a transactional environment there is a high degree of concurrency since real problems require a number of activities to take place in parallel. We will therefore find the general theory of non-interleaving representation of parallel behaviour found in [11] of great use in what follows.

The semantics is intended to describe the behaviour of a transaction in terms of its services at the deployment level, but not the low-level computations performed by the services themselves. In fact, in certain contexts such as a digital ecosystem for business services are offered by different service providers and it is important that we defer from interfering with the local state of service execution. This means it is appropriate to consider that any action within the transaction model has no significant duration, in the sense that (i) it either occurs as a whole or not at all; (ii) it occurs either wholly before, or wholly after, or wholly in parallel with, every other action.

As discussed in Section 2.2, a transaction involves a number of local agents acting on behalf of different parties (service providers, consumers, or both) which collaborate to perform a business activity. To preserve the local autonomy of each platform, they communicate via their coordinator components which manage the required service interactions (recall Fig. 2.3 and 2.4).

A transaction T is associated with a set of coordinator components C and a set of

actions M. Our interest is in the observable events on the coordinator components and thus actions can be understood as service invocations between the participating components, as shown for example in the scenario of Fig. 2.4. 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 μ is given by $\mu : C \to \wp(M)$. Further, we require that $\bigcup_{i \in C} \mu(i) \subseteq M$.

As can be seen in Fig. 2.3, 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 [11] 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 $\underline{v} : C \to M^*$ such that $\underline{v}(i) \in \mu(i)^*$.

By $\mu(i)^*$ we denote the set of finite sequences over $mu(i)$. Mathematically, the set V_T is the Cartesian product of the sets $\mu(i)^*$, for each i. Effectively, are n-tuples of sequences where each coordinate corresponds to a coordinatro component in the transaction (hence, n is the number of components) and contains a finite sequence of actions that have occurred on 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. For example, the vector $(a1, \Lambda, \Lambda)$ describes that portion of behaviour of the transaction in which an action $a1$ (e.g. service invocation) has taken place on the corresponding component allocated to the first coordinate. The vector $(a1, b1, \Lambda)$ describes that portion of behaviour in which both $a1$ and $b1$ have happened on the corresponding components while the vector $(a1a2, b1, \Lambda)$ describes an occurrence of $a1$ and an occurrence of $a2$ on the component corresponding to the first coordinate, and an occurrence of $b1$ on the second coordinate - nothing has happened on the component corresponding to the third coordinate.

It can be seen from the example given above that there is already an ordering among actions on a particular access point or component, e.g. $a1$ followed by $a2$. This vector-based behavioural description of transactions can also capture the orderings between service invocations on different components, which amounts to actions appearing on different vector coordinates. This requires however a more careful

consideration of the mathematical properties of such vectors which is described in the sequel.

At this stage it suffices to understand that each transaction vector provides a snapshot of behaviour that captures what actions have already occurred and on which part (component) of the transaction. In describing the behaviour of a transaction however we are interested only in those vectors describing (orderings of) actions that we expect the coordinator components to engage in during the course of the transaction execution. In other words, for a given transaction T we are interested in a particular subset of all possible vectors formed over T.

We will use the term *transaction language* to refer to an appropriate subset V of all possible vectors V_T formed over a given transaction T. The idea is that the particular subset of transaction vectors, for a specific transaction, expresses the ordering constraints necessary in the corresponding service orchestration. To identify the appropriate subset of vectors capturing *intended* behaviour only, we turn our attention to the corresponidng UML model that describes the required sequences of service interactions.

We outline how UML 2.0 sequence diagrams [12] can be translated into transaction vectors in Section 2.4 and show how formal reasoning against the order-theoretic properties of the transaction language (given next) can be used to determine the complete set of behaviours of a transaction and inform the refinement of the initial UML model.

2.3.1 Basics of transaction vectors

We now examine the basic mathematical properties of our formal construction so far. The discussion is restricted to those operations used in the remainder of the chapter. A detailed mathematical treatment can be found in [14, 11].

We start by introducing a specific kind of transaction vector, which is used in our model to describe actions (e.g. service invocations) within a transaction.

Column vectors. Let T be a transaction and V_T its set of transaction vectors. We define $A_T = \{\underline{\alpha} \in V_T \setminus \{\underline{\Lambda}_T\} : i \in C \Longrightarrow |\underline{\alpha}(i)| \leq 1\}$ where $|x|$ denotes the lenght of the sequence $|x|$.

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 example, the vector $(a1, \Lambda, \Lambda)$ represents the occurrence of an action $a1$ on the component 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.

Operations on vectors. For $\underline{u}, \underline{v} \in V_T$, 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*)
- $\underline{u} \sqcap \underline{v}$ to be the vector $\underline{w}$ which satisfies $\underline{w}(i) = min(\underline{u}(i), \underline{v}(i))$, for each i
- $\underline{u} \sqcup \underline{v}$ (if it exists) to be the vector $\underline{w}$ which satisfies $\underline{w}(i) = max(\underline{u}(i), \underline{v}(i))$
- 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, $(a1, b1, \Lambda).(a2, \Lambda, \Lambda) = (a1a2, b1, \Lambda)$.

The ordering amongst vectors is defined in terms of the usual prefix ordering operation on sequences appearing on their respective coordinates. For example, $(a1, b1, \Lambda) \leq (a1a2, b1, \Lambda)$ since $a1 \leq a1a2$ and $b1 \leq b1$ 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 '$\sqcap$' and '$\sqcup$' give the greatest lower bound and the least upper bound of $\underline{u}, \underline{v} \in V_T$, respectively, in the usual sense of lattices and domain theory [15]. As we will see, these operations are central to the treatment of concurrency in our approach.

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 ŚcontinuationŠ of $\underline{u}$ that extends it to $\underline{v}$. This operation is central to the treatment of compensations in our approach. We return to this discussion in the concluding section of this chapter.

It is important to stress the fact that all operations on vectors are performed coordinate-wise and this simplifies the proofs but also makes the formal model feasible for implementation. We are now set to show how transaction vectors can be used to capture the dependencies between service interactions of coordinator components within a transaction.

2.3.2 Managing dependencies

The prefix ordering relation on transaction vectors can be viewed as an ordering on partial executions, where each vector corresponds to that portion of behaviour in which the transaction has already engaged in the actions appearing on its coordinates. This can be expressed more succinctly by saying that $\underline{u} \leq \underline{v}$ means that $\underline{u}$ is an earlier part of behaviour leading to $\underline{v}$.

A more careful examination of the mathematical construction shows that we can say more than that. Indeed, we find it useful to determine *immediate predecessors* (or *successors*) of a transaction vector.

Covers. Let $\underline{u}, \underline{v} \in V \subseteq V_T$. We say that $\underline{v}$ *covers* $\underline{u}$ in V, and we write $\underline{u} \lhd \underline{v}$ iff (i) $\underline{u} \leq \underline{v}$ and $\underline{u} \neq \underline{v}$ and (ii) If $\underline{z} \in V$ such that $\underline{u} \leq \underline{z} \leq \underline{v}$, then $\underline{z} = \underline{u} \vee \underline{z} = \underline{v}$.

Thus, whenever $\underline{u} \leq \underline{v}$, and we also have that $\underline{u} \lhd \underline{v}$, then the last actions that went into forming each vector have occurred consecutively - one after the other. This allows to model sequential dependency inside a transaction. Recall the example of Fig. 2.3 where service $c1$ can only be called after $a2$.

Our approach towards modelling concurrent actions, actions that can happen in parallel, draws upon the concepts in Shields' *vector languages* [16] and Mazurkiewicz *trace languages* [17] where concurrent actions are considered as being *unordered*, in contrast to CSP trace theory where concurrent events are understood to occur *in either* order (nondeterministic interleaving).

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. Drawing upon the extension of the independence relation ι to *behaviour vectors* in [11], 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

$$\underline{u}\ ind\ \underline{v} \Longleftrightarrow \forall i \in C : \underline{u}(i) > \Lambda \Rightarrow \underline{v}(i) = \Lambda$$

This definition says that two transaction vectors are independent if the behaviours they describe engage distinct components (correspond to service invocations on different coordinator components) of the transaction. This means the behaviours described by $\underline{u}$ and $\underline{v}$ may occur independently.

In the case of column vectors, independence captures the fact that actions appearing in one vector may occur independently of those appearing in the other. If in addition the vectors representing these actions are adjacent in an expression (of the series of concatenations that went into forming the corresponding transaction vectors), then the actions are concurrent. Hence, whenever two actions are independent and are both enabled (can both occur at some point, after some behaviour) then, their corresponding column vectors commute, i.e. $\underline{\alpha}_1.\underline{\alpha}_2 = \underline{\alpha}_2.\underline{\alpha}_1$, and in the resulting behaviour the two actions are concurrent.

Based on the prefix ordering between transaction vectors in the set V we may also model a choice between actions. That is, actions which are mutually exclusive in that occurrence of one excludes occurrence of the other. In discussing concurrent actions in a long-running transaction, we saw that the two incomparable transaction vectors represent parallel behaviour. The vector they both cover is in fact their greatest lower bound and is obtained by applying the operation '$\sqcap$' given earlier. The fact the two incomparable vectors represent concurrent actions is only because they are bounded above in the set (by the transaction vector which is their least upper bound, given by '$\sqcup$', and is sitting on top of the lozenge). Whenever this latter requirement does not hold we may talk about events in conflict.

It might be instructive to make the distinction in terms of pictures and associated Hasse diagrams. In the diagram of Fig. 2.5, $a1$ and $d1$ are sequential ($d1$ can only be invoked after $a1$) in Fig. 2.5(i), they are concurrent in Fig. 2.5(ii) while there is a choice between them (alternative) in Fig. 2.5(iii).

Notice that the set of vectors in (i) does not include $(\Lambda, d1)$, which means that $d1$ never occurs before $a1$ does; in (iii) it does not include $(a1, d1)$ which means there is no valid behaviour of the transaction processing system in which both $a1$ and $d1$ have taken place; in (ii) it includes all four vectors, which means that $a1$ on its own, $d1$ on its own, and $a1$, $d1$ together, are all valid observations of the behaviour of the transaction in which $a1$ and $d1$ happened concurrently. This is indicated by

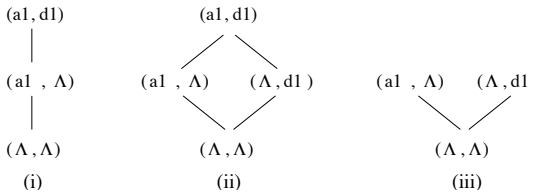

Figure 2.5: Order structure of transaction languages

the familiar lozenge shape that shows the corresponding order structure exhibits the characteristic structure of a finite lattice.

In further explanation, the vector sitting at the bottom of the lozenge is the greatest lower bound '$\sqcap$' of the two incomparable vectors $(a1, \Lambda)$ and $(\Lambda, d1)$ sitting at the middle of the lozenge while the vector at the top is their least upper bound '$\sqcup$'. The lozenge as a whole describes that part of behaviour of the transaction in which $a1$ and $d1$ happened concurrently. Further details on how the ordering relations between actions are manifested in the order structure of the resulting set of vectors can be found in [14].

2.3.3 Well-formedness of the behavioural description

In describing the behaviour of a transaction we are interested in the actions (activations) on its various components. These are captured in our model using *column vectors*. Thus, instead of considering all possible transaction vectors we would like to be concerned with those obtained by concatenations with column vectors only. This gives the behaviour of the transaction in terms of actions of its coordinator components and can be used to enforce the coordination of the underlying services.

We have seen that transaction vectors are obtained by coordinate-wise concatenation. Hence, they can be seen to be built up starting from the empty vector by a series of concatenations with column vectors which represent actions. The study of vector languages in [11, 10] shows that in order to ensure that vectors considered are the result of concatenations with column vectors only, the set of transaction vectors must satisfy certain properties. We introduce these properties next.

The first property captures the fact that a system's computations always have a starting point, and ensure that only a finite number of actions may occur within finite time. This turns out to be the case if whenever two vectors describe an earlier part of behaviour than a third, also in the set, then their least upper and greatest

lower bounds are also in the set. This is formally put in the following definition.

Discreteness. Let $V \subseteq V_T$, then V is *discrete* iff $\underline{\Lambda}_T \in V$ and whenever $\underline{u}, \underline{v}, \underline{w} \in V$ such that $\underline{u}, \underline{v} \leq \underline{w}$ then (i) $\underline{u} \sqcup \underline{v} \in V$ and (ii) $\underline{u} \sqcap \underline{v} \in V$.

Note that $\underline{u} \sqcup \underline{v}$ is understood as asserting that '$\sqcup$' is defined.

Discreteness imposes a finiteness constraint in the sense that it excludes infinite ascending or descending chains of actions with respect to time ordering. In fact, it ensures that situations like those resulting in Zeno-type paradoxes will never arise.

We further require that every occurrence of an action (e.g. service invocation) is recorded in the set of vectors associated with the transaction. This guarantees that any earlier part of behaviour is itself a behaviour and motivates the following definition.

Local left-closure. Let $V \subseteq V_T$, $i \in C$ and $x \in \mu(i)^*$. Then, V is *locally left-closed* iff, whenever $\underline{v} \in V$ and $\Lambda < x \leq \mu(i)$ then there exists $\underline{u} \in V$ such that $\underline{u} \leq \underline{v}$ and $\underline{u}(i) = x$.

The local left-closure property is intended to resolve ambiguities that may arise from not having enough vectors in the transaction language to describe the course of the behaviour in question; not the start or the end, but the 'gaps' in between. This requires that every occurrence of an event is 'recorded' in the language of the transaction. This implies the presence of a distinct *prime* element in V for each occurrence of an action. Primes play a central role in the more general theory of parallelism [11] and in particular with respect to associating vector languages with order-theoretic objects used to determine the temporal relation between occurrences of actions. For the purposes of the present text, and the adaptation of this theory in deriving a formal model for long-running transactions, it suffices to understand that, in this context, the notion of prime refers to transaction vectors which have a unique other vector immediately beneath them. Such an ordering is determined by the *covers* relation among vectors, given earlier.

To establish some terminology for the sequel, we say that the set of vectors $V \subseteq V_T$ associated with a transaction T is *normal* if it is locally left-closed and discrete. This reflects the fact that the guarantees that accrue from these properties are embedded in the behaviour of the corresponding transaction.

In fact, discreteness and local left-closure ensure the well-formedness of the behavioural description of a transaction in our model. The idea is that in checking against these properties we may determine whether the transaction will exhibit the desired behaviour when executed or on the contrary, other non-desirable or simply

unthought scenarios of execution are still possible. This draws upon previous work on vector languages and UML sequence diagrams in [14], which is adapted to refining transaction contexts in the next section.

2.4 Elaborating behavioural scenarios

In the previous section, we have used vector languages to capture the coordination of the service interactions between coordinator components in a transaction. In Section 2.2 we used UML sequence diagrams to describe the order in which the underlying services need to be deployed. In this section, we show how transaction languages, and associated order structures, can be used to determine whether there are any discrepancies between the specified order of execution, given in the sequence diagram, and the actual order in which the services can be deployed.

First, we need to understand how to obtain a transaction language from the corresponding sequence diagram and then show how in checking against *normality* we can identify missing behaviours. In previous work [9] we have shown how vector languages can be obtained from sequence diagrams. Here we only outline the general idea.

There are graphical positions or *locations* along the lifelines of participating instances in a sequence diagram that are of particular significance, especially when the diagram is considered in a formal setting. Our approach draws upon the work in [18] where locations are treated formally to obtain the corresponding model. To obtain the corresponding vector language we associate vectors to the locations of the diagram and use the resulting language to determine the relations between those interactions in order to reflect the meaning of the diagram. Fig. 2.6 shows the relation between the locations in a simple sequence diagram and the corresponding vector language, which is depicted in a Hasse diagram. The XML description of the translation for the vector $(a1, b1)$ is given on the right, and is derived following the schema shown in Fig. 2.7.

The vectors associated with each location are obtained from the vectors of the immediately preceding location, by concatenating the action (the corresponding column vector) associated with the location being considered. By convention the initial location is mapped onto the empty vector $\underline{\Lambda}_T$. By moving down the diagram, from one location to the next, whilst mapping each location to (a set of) vectors, the sequence diagram is translated into vectors.

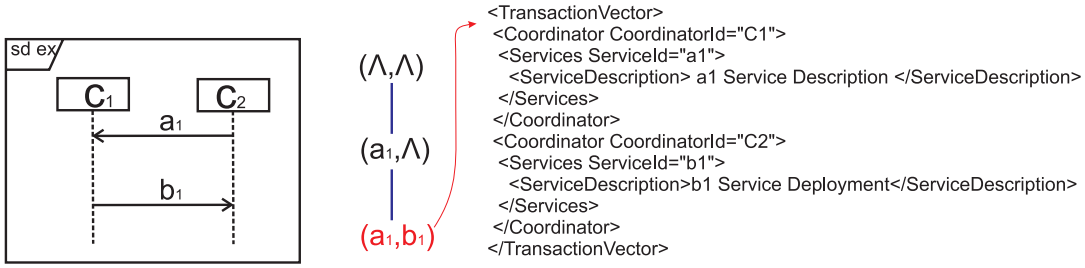

Figure 2.6: A sequence diagram and its corresponding transaction language

There are some cases however in which this rationale does not apply. In particular, locations within different operands of an **alt** or **par** need to be treated differently. This is because we have to take into account the various execution sequences that are possible when encountering these interaction fragments. Note that a location is also used to mark the beginning and the end of interaction fragments superimposed on the diagram. The first location of each operand in an **alt** or **par** fragment is considered in relation to the start location of the fragment rather than its immediately preceding location. The vectors of the end location of an **alt** fragment with k operands are considered in relation to the last location of each operand - to reflect the fact there are k alternative scenarios. The vectors of the end location of a **par** fragment are carefully obtained to reflect the fact the actions appearing within are effectively unordered. Full details of the formal construction behind the translation can be found in [9].

The schema for the formal translation given in Fig. 2.7 is used for deriving XML representations of the corresponding transaction language that reflects its order structure. As we will see, these *transaction scripts* can be used to identify implicit behavioural scenarios in the corresponding transaction context (Fig. 2.3 and 2.4).

Our approach can handle both synchronous and asynchronous communication. The synchronous case is captured by a shared action, e.g. it would be represented by a column vector such as $\underline{\alpha} = (a1, a1, \Lambda)$ which denotes the simultaneous occurrence of sending and receiving $a1$. Asynchronous communication is captured by distinct column vectors, e.g. $\underline{\alpha}_1 = (\Lambda, a1, \Lambda)$ and $\underline{\alpha}_2 = (a1, \Lambda, \Lambda)$ describing the consecutive actions of sending and receiving $a1$, which would result after concatenation with the corresponding transaction vectors into vectors related by '$\lhd$' which infers immediate causality.

Since we have not explicitly discussed simultaneity within the formal framework

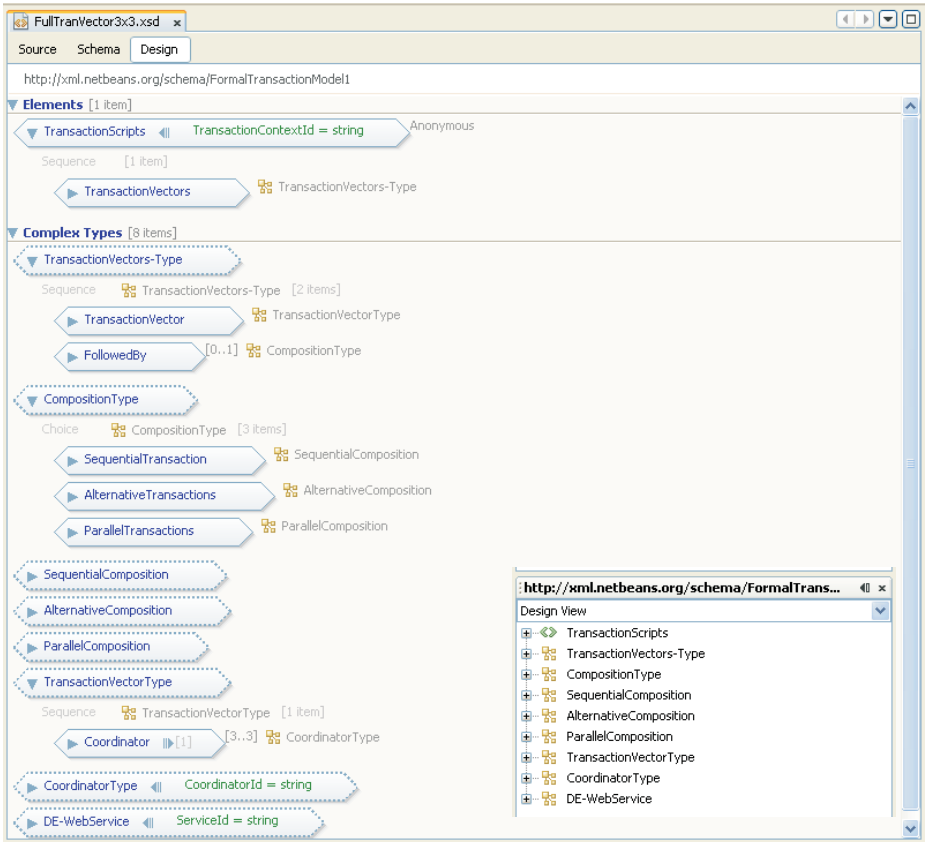

Figure 2.7: XML schema for the formal translation of behavioural scenarios

in this chapter, we will assume asynchronous communications only and hence use $\underline{\alpha}_1 = (\Lambda, a1, \Lambda)$ and $\underline{\alpha}_2 = (a1, \Lambda, \Lambda)$, and in particular, we will be concerned with $(a1, \Lambda, \Lambda)$ as it is receiving $a1$ that corresponds to the actual service deployment which is of primary interest in a transactional setting.

Recall the sequence diagram of Fig. 2.4 describing the interactions between services of the coordinator components. The transaction language that models the behaviour represented in the sequence diagram is given in Fig. 2.8. The corresponding XML description can be found online following [1].

In Section 2.3 we argued about *normality* in transaction languages and identified properties, discreteness and local left-closure, that ensure the well-formedness of this tuples-based description of behaviour. By careful examination of the transaction language in Fig. 2.8, it can be seen that while it is locally left-closed, the discreteness property is violated. Indeed, the vectors $\underline{u} = (a1a2, \Lambda, \Lambda)$ and $\underline{v} = (a1, b1, \Lambda)$ are both smaller than vector $\underline{w} = (a1a2, b1, c1)$ and their great-

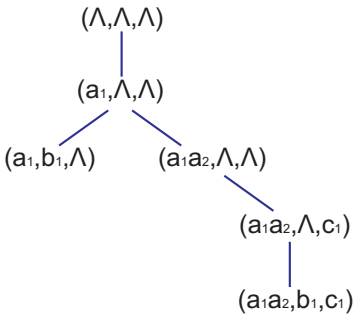

Figure 2.8: Transaction language for the interaction of Fig. 2.4

est lower bound ($(a1, \Lambda, \Lambda)$) is in the set, but their least upper bound, given by $(a1, b1, \Lambda) \sqcup (a1a2, b1, \Lambda)$, is not.

According to our mathematical framework, this vector should be added to make V_T discrete and thus also normal. The effect of adding in the missing behaviour, as shown in Fig. 2.9, is that there is now potential concurrency between $a1$ and $b1$ (first occurrence of) and $c1$ and $b1$. So there are two additional scenarios of execution, on top of the two scenarios described explicitly in the sequence diagram of Fig. 2.4. The component developers can now determine whether these scenarios describe desirable behaviour or not.

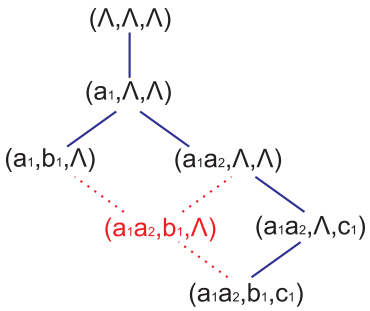

Figure 2.9: Discrete transaction language for the interaction of Fig 2.4

The service calls $c1$ and $b1$ can take place concurrently since, although they were initially designed to occur sequentially (recall the sequential composer in the transaction tree of Fig. 2.3), there is a dependency between $a2$ and $b1$, and between $a2$ and $c1$, but there is no dependency between $c1$ and $b1$, and hence they are not necessarily related by immediate causality (as the sequence diagram of Fig. 2.4 would indicate). This is reflected in the refined sequence diagram of Fig. 2.10 where

the implicit scenarios have been made explicit. Upon receiving a call for service deployment $a2$ the component $CC1$ can proceed to do $c1$ and $b1$ in any order, including at the same time.

We note that the formal model also indicates potential concurrency between $a1$ and $b1$ (the first occurrence of $b1$). This is a situation known as *asymmetric confusion* in Net theory [19] - a situation where the choice between an action happening on its own and concurrently with some other action is never actually resolved. Within reason, we would expect the application programmer (of the coordinator component $CC1$) to prohibit concurrency in this case at the implementation level. At the design level, this can be done by adding a type of acknowledgement message $a3$ that will infer immediate causality, as shown in Fig. 2.10 (right). The corresponding XML representations can be found online following [1].

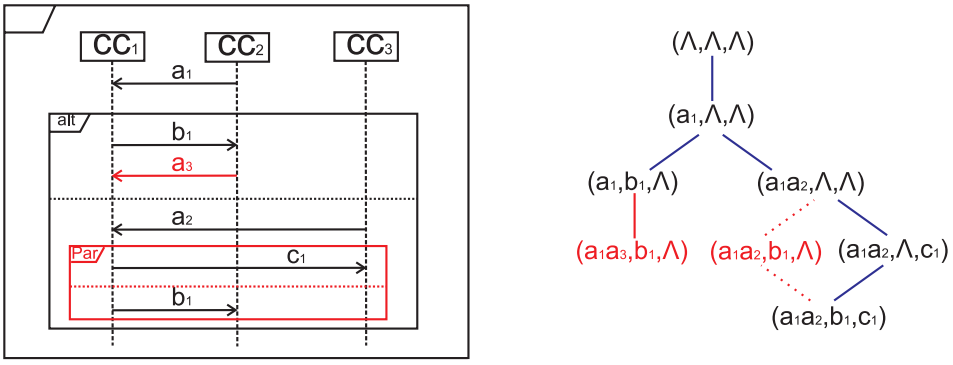

Figure 2.10: Transaction language of the elaborated behavioural scenarios

Fig. 2.11 shows the corresponding transaction tree which is now optimised to reflect the complete set of scenarios of execution.

2.5 Concluding notes

We have described a formal framework for the coordination of concurrent and distributed service invocations between coordinator components involved in a long-running transaction. In particular, we have used transaction trees (structure) and UML sequence diagrams (interactions) to specify a transaction. We then showed how a formal description of this initial transaction context can be used to reason about the corresponding behavioural scenarios and identify implicit scenarios of interaction which may indicate emergent behaviour. We have seen that our formal

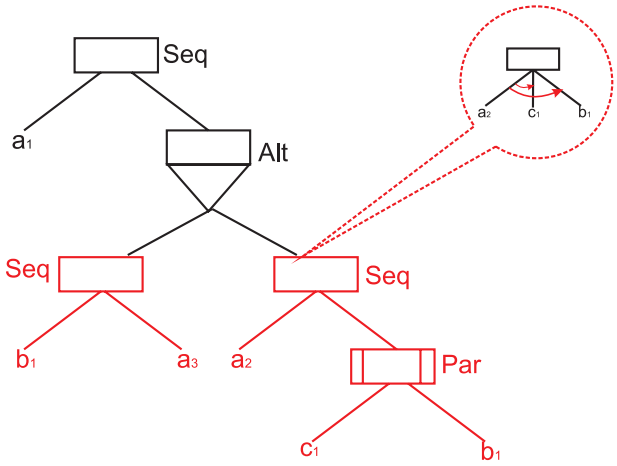

Figure 2.11: Optimised transaction tree

framework can determine the complete set of behaviours and make all possible scenarios explicit in the corresponding scenario-based specification.

Schemas for deriving XML representations of both a transaction context and the corresponding transaction scripts (reflecting transaction vectors) needed to refine the behavioural scenarios of the initial transaction context, were also given. The complete source files can be found online following [1].

In [5] we have been concerned with dependencies due to data sharing and have presented an extended lock mechanism that ensures consistency and drives the roll-back procedure if some failure later in the transaction makes recovery necessary. Work is in progress on integrating the transaction language model presented here with the lock mechanism so that the orderings between transaction vectors trigger the appropriate lock scheme whenever necessary.

The approach described in this chapter has focused on dependencies between (services of) coordinator components within a transaction. Dependencies may also exist across transactions due to the need for releasing some results of a transaction - often referred to as *partial results* - to another transaction before it commits. An extension to address partial results, based on [20], and compensating actions (using the right-cancellation operator '/') has been provided in Deliverable D3.2 and the corresponding extension in temrs of (compensating) transaction scripts is currently under investigation.

3 Modeling network dynamics for simulation

3.1 Dynamic Social Network Model

To introduce the complete co-evolution model, we need first to understand the actor-oriented approach, and after that how to include the behavioral aspect. In terms of the methodological development, this was the chronological sequence of models presented by the main research group.

3.1.1 Actor-oriented model

Snijders introduced stochastic actor-oriented models ([21], [22], [23]) firstly modeling just the evolution of the network. Actor orientation means that, for each change in network, the perspective is taken of the actor whose tie is changing, such that this actor (which we will call i actor) controls the whole set of tie variables included in the i-th row of the adjacency matrix that defines the network. This is called a micro-step, so that the analysis of the dynamic has a bottom-up perspective, from micro changes (actor-tie selection) to macro changes (change in network topology).

To overcome the methodological frailties already exposed, the idea is to consider a time continuous stochastic process where the state space is formed by all network configurations of directed graphs given a set of actors. The observed network is modeled by means of a parametric model for the transition probabilities between these states. In network panel data, each panel will present a configuration that pertains to the whole large set of the state space. So that, the explanation of dynamics is formulated by means of the transition probabilities, having the first observation conditioned upon. In other words, the first observation is the starting value of the stochastic process. As can be seen, this state space grows squared

exponentially, and a simple binary network with 10 actors will present approximately $2^{(10(10-1))} = 1,23x10^{25} =$ possible states. So that, three basic assumptions are made to reduce the complexity ([24]):

1. the transitions between panel measurements are manifestations of an underlying process that takes place in continuous time;
2. actors do not coordinate their actions, but act conditionally independent from each other, given the current state of the network;
3. actors change at most one tie at a time, that is, create one new link or dissolve one existing link in one microstep.

So that, the modeling task is reduced to:

- Modeling the change in the microstep
- Modeling the occurrence of this microstep, over time

The first task, a, is solved by means of a multinomial logistic model that will be used to a maximization of a stochastic utility function, here called the Objective function. Task b) consists of a specification of a probability model for the actor's individual waiting time (usually the Exponential model), called the Rate function. By this approach, the time dependence of the evolution process is implicitly modeled as an emergent consequence of model dependence on time. Both model components include dependences on previous network state (topology), time and actor, but not for the whole history of the process (Markov assumption).

At this point some notation should be introduced. Let $X(t) = X1, X2, ..., Xn'$ be the adjacency matrix that defines the network topology at time t, where Xi is an i, X, n vector. So that, we define the model components.

Rate function

The rate function $\lambda i(x)$ for actor i is the rate at which changes occur in this actor's ties, Xi. Formally, it can be defined as:

$$\lambda_i(x) = lim_{dt \to 0} \frac{1}{dt} P(X_{ij}(t + dt) \neq X_{ij}(t), forsomej \in \{1, ..., n | X(t) = x\})$$

The simplest way to model it, is to consider this rate, constant among all actors, or $\lambda i = \lambda$ for $i = 1, ..., n$. Based on (1) and the constant λ assumption, we have that waiting times D between successive mini-steps will follow the Exponential probability distribution, with density function $\lambda^{e-\lambda d}$, for $d > 0$. Consequently, the expected number of total changes in the whole network, between two consecutive time points t_a and t_b is $n\lambda(t_b - t_a)$.

The constant assumption can be relaxed, and this can be a function of attribute variables, so that we have $\lambda(\eta, W)$, where η is the effect parameter vector for the attribute W. Also, it can be modeled depending on the network structure. More details can be found in [24] and [25].

Objective function

The meaning of this function is not as easy as the rate function, mainly for no-practitioners, since it involves some network concepts like transitivity and balance among others. In the previous sections, it was presented as the functions that will govern the mechanism of network evolution, so it is related to the topology of the network.

The objective function can be understood as a representation of the preference distribution of the actor i the set of all possible networks. When actor i makes a change (a microstep) he/she will change the configuration of his/her network neighborhood. Let $X_i = \sum_j X_{ij}$ be the number of all ties actor i has and $n-1-X_{i+}$ the number of actors to whom he/she is not tied. Denoting $x = X(t)$ as the state of the network at time t, the new network formed after a tie is changed by the actor is denoted $x(i \rightarrow j)$. The choice for this change is modeled as follows:

Let $U(j)$ be a random latent variable that represents the unexplained part of the preference for i to j, or a latent preference. This variable is assumed to have a symmetric distribution around 0 (a white noise) and independently generated by each micro-step. The idea is that actor i will change the digraph in a way that maximizes

$$f(x(i \rightarrow j)) + U(j)$$

A convenient traditional distribution for $U(.)$ is the Gumbel distribution, with mean 0 and scale parameter 1. Under this assumption, the probability that i chooses to change the digraph x_{ij} is given by the logistic function:

$$p_{ij}(x) = \frac{e^{f(x(i \to j))}}{\sum_{x_i} e^{f(x(i \to j))}}$$

Here, $f(.)$ is the function that is a weighted sum on a parameter vector $\beta_1, \beta_2, ..., \beta_L$ such that

$$f_i(\beta, x) = \sum_{k=1}^{L} \beta_k s_{ik}(x)$$

Functions $s_{ik}(x)$ are well known metrics in Social Network Analysis (SNA), like:

1. Density effect (out-degree)

$$s_{i1}(x) = x_{i+} = \sum_j x_{ij}$$

2. Reciprocity effect

$$s_{i2}(x) = x_r = \sum_j x_{ij} x_{ji}$$

3. Transitivity effect

$$s_{i3}(x) = \sum_j \sum_h x_{ij} x_{ih} x_{jh}$$

among others like Balance, Number of geodesic distances 2, Popularity and Activity effect. Table 1, in the Appendix shows a number of Objective Functions.

So, the parameters β give the importance of the effect in the actors' choice in changing ties. It is possible to estimate uncertainty for these parameters and even perform hypotheses tests to assess the statistical relevance of each of the $s(.)$ effects estimated. The choice of which and how many effects should be included in the models depends on a previous exploratory analysis and, of course, in the size of the network, since we have to have enough degrees of freedom for estimation. Remember also that the difficulty in interpreting and computational time cost increase as the number of parameters do.

3.1.2 Behavior component

So far, all the definitions of the dynamic modeling stated are only about the network evolution, a process that emerges from actors' choices. The addition of the behaviour component is done in a straightforward way, as in the following.

For each behavioral variable, that must be a discrete outcome, a separated behavioral state space is handled consisting of all possible individual scores of the variable, and so observed transition on each behavioral dimension is modeled in an analogue way of micro-steps, decomposing all changes and using parametric model for the probability transitions. These behavioral microsteps again are modeled as multinomial logistic distribution based on random utility objective function and their changes following an Exponential distribution based on a rate function.

The integration of this component to the network evolution is done by:

1. specifying a Cartesian product of the separate state spaces as a joint state space;

2. assuming conditional independence of the occurrence of the different types of micro-steps;

3. extending the separate objective and rate functions to allow for dependence on the respective dimension of the state space

In step 3 the interdependence between network and behavior dynamics is introduced in the model. The Markov property is inherited from the separate components which form the complete model and actor-driven nature. This is reflected in the locus of occurrence of any micro-step. It is the actor who decides the changes he/she has under control, the outgoing ties and the behavior, having the velocity of these changes modeled by the rate function and the evaluations of consequences once a decision change is made modeled by the respective objective function.

3.2 Co-evolution model for simulation

We already presented modeling aspects of the co-evolution of a network and behavior. The next step in this formalization would be the estimation of that parameters. However, this is not part of the scope of D3.7, rather a complete description will be available in D10.8. The idea at this point is propose how to introduce these ideas in the EvESim framework, providing ideas for simulations of distributed service composition, in particular sequential, alternative and concurrent interaction-based service composition, to address a distributed P2P scenario.

This suggestion will be based on the development of the concept of Dynamic Virtual Super-Peer (DVSP) ([26]). In this concept nodes from a network form, by means

of a choice/selection algorithm, a super-peer that can provide information/services demanded from the whole network. The selection of peers is made by the Stability Function, that tries to translate to a unidimensional metric the multidimensional concept of stability and service provider.

The first aspect in terms of the dynamics is the possibility to control the rate at which new ties are stabilised, using the Rate Function. This rate can be defined as fixed or varying according to some conditions of the existing network. The changing rates, for behavior and for selection, can be added as additional parameters, and possibly have close relations to the "'Migration Rate"', already presented in the menu but not implemented (according to D9.2, from the preceeding DBE project). The ideal way to do this should be studied, since it will introduce to the Fitness Function time dependence in the parameters. The mathematics involved and the impact in computation time is another point to be weighed in a cost-effectiveness analysis for implementation. Tables 3.1 and 3.1 give some effects that can be implemented.

The main concern in this approach is the clusterisation, or how groups of peers (machines) are formed to provide a DVSP. In SN language this is translated to Clustering Coefficient. In the co-evolution model, many of the objective functions are related to clustering, such that *Transitivity Effect, Transitive Triplet and 3-cycle and Preferential Attachment.* Regarding P2P simulations in Evesim, the variables used in the stability function can be also included, perhaps separating sets for behavior variables and selection dynamics. Some hypothetical examples of effects presented and interpreted in Table 3.1 and Table 3.2, based in a study of substance abuse among adolescents. Making analogies, those effects can be transformed in flexible variables that define different types of peers. However, just after analyses from real data are made, we will have a better idea about what variables can be included.

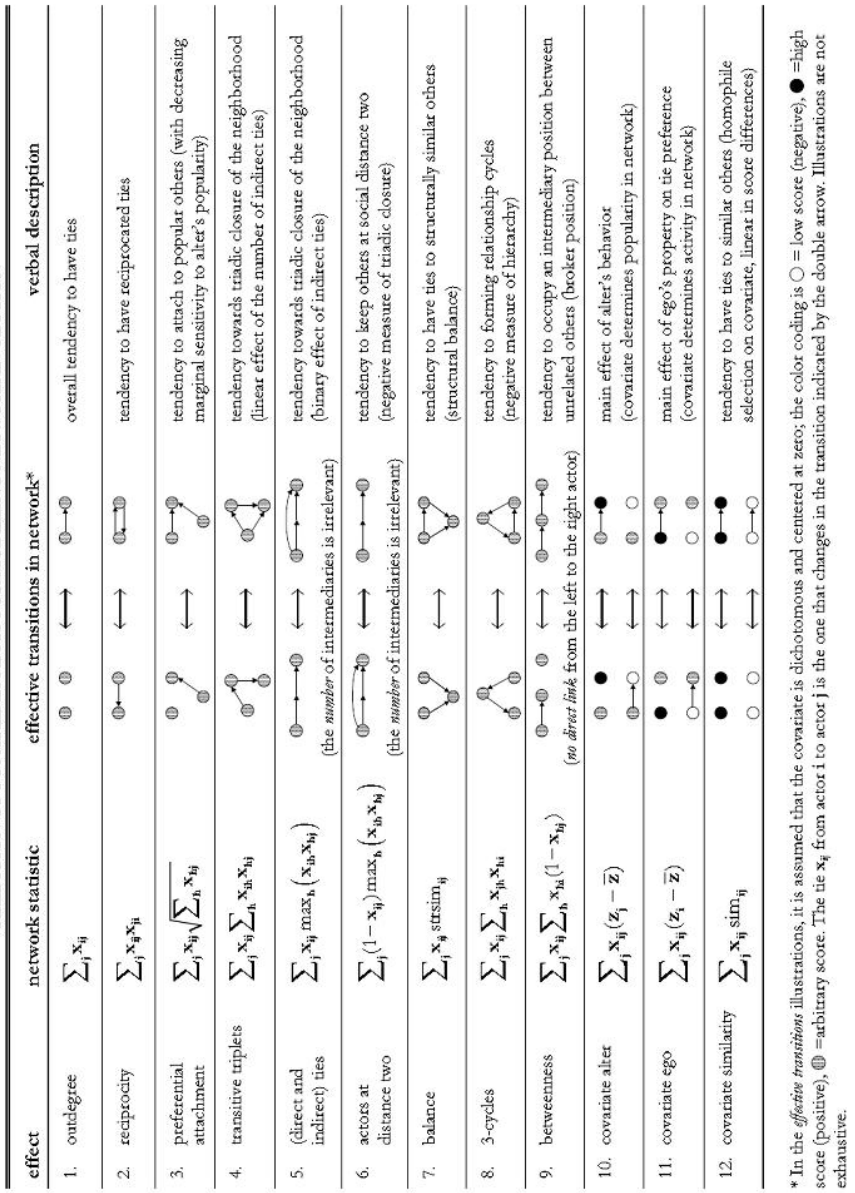

TABLE 2
SELECTION OF POSSIBLE EFFECTS FOR MODELING NETWORK EVOLUTION

effect	network statistic	effective transitions in network*	verbal description
1. outdegree	$\sum_j x_{ij}$		overall tendency to have ties
2. reciprocity	$\sum_j x_{ij} x_{ji}$		tendency to have reciprocated ties
3. preferential attachment	$\sum_j x_{ij} \sqrt{\sum_h x_{hj}}$		tendency to attach to popular others (with decreasing marginal sensitivity to alter's popularity)
4. transitive triplets	$\sum_j x_{ij} \sum_h x_{ih} x_{hj}$		tendency towards triadic closure of the neighborhood (linear effect of the number of indirect ties)
5. (direct and indirect) ties	$\sum_j x_{ij} \max_h (x_{ih} x_{hj})$	(the *number* of intermediaries is irrelevant)	tendency towards triadic closure of the neighborhood (binary effect of indirect ties)
6. actors at distance two	$\sum_j (1-x_{ij}) \max_h (x_{ih} x_{hj})$	(the *number* of intermediaries is irrelevant)	tendency to keep others at social distance two (negative measure of triadic closure)
7. balance	$\sum_j x_{ij}\, \mathrm{strsim}_{ij}$		tendency to have ties to structurally similar others (structural balance)
8. 3-cycles	$\sum_j x_{ij} \sum_h x_{jh} x_{hi}$		tendency to forming relationship cycles (negative measure of hierarchy)
9. betweenness	$\sum_j x_{ij} \sum_h x_{hi} (1-x_{hj})$	(*no direct link* from the left to the right actor)	tendency to occupy an intermediary position between unrelated others (broker position)
10. covariate alter	$\sum_j x_{ij} (z_j - \bar{z})$		main effect of alter's behavior (covariate determines popularity in network)
11. covariate ego	$\sum_j x_{ij} (z_i - \bar{z})$		main effect of ego's property on tie preference (covariate determines activity in network)
12. covariate similarity	$\sum_j x_{ij}\, \mathrm{sim}_{ij}$		tendency to have ties to similar others (homophile selection on covariate, linear in score differences)

* In the *effective transitions* illustrations, it is assumed that the covariate is dichotomous and centered at zero; the color coding is ○ = low score (negative), ● =high score (positive), ◍ =arbitrary score. The tie x_{ij} from actor i to actor j is the one that changes in the transition indicated by the double arrow. Illustrations are not exhaustive.

Figure 3.1: Source: Steglich, C., Snijders T, Pearson M. Dynamic networks and behavior: separating selection from influence. Technical report, under review. Available at: `http://www.gmw.rug.nl/~steglich/sites/index.htm`.

TABLE 3

SELECTION OF POSSIBLE NETWORK EFFECTS FOR MODELING BEHAVIORAL EVOLUTION

effect	network statistic	effective transitions in behavior*	verbal description
1. shape: linear and quadratic	$(z_i - \bar{z})$ and $(z_i - \bar{z})^2$	○ ⟷ ●	*The two parameters together define a parabola shape of the objective function, allowing to capture the basic shape of the observed distribution of the behavioral variable.*
2. average similarity	$\left(\sum_j x_{ij} \mathrm{sim}_{ij}\right) \Big/ \left(\sum_j x_{ij}\right)$	○→● ⟷ ●→● ●→○ ⟷ ○→○	assimilation to neighbors' average behavior (small neighborhoods pull as much as big ones)
3. sum of similarity	$\sum_j x_{ij} \mathrm{sim}_{ij}$	○→● ⟷ ●→● ●→○ ⟷ ○→○	assimilation to neighbors' average behavior (size of neighborhood determines size of effect)
4. average alters	$\left(\sum_j x_{ij}(z_j - \bar{z})\right) \Big/ \left(\sum_j x_{ij}\right)$	○→● ⟷ ●→● ●→○ ⟷ ○→○	main effect of neighbors' average behavior (contagion / influence, but not necessarily assimilation)
5. indegree × behavior	$(z_i - \bar{z}) \sum_j x_{ji}$	○→◍ ⟷ ●→◍	effect of own popularity in the network on behavior
6. outdegree × behavior	$(z_i - \bar{z}) \sum_j x_{ij}$	○→◍ ⟷ ●→◍	effect of own activity in the network on behavior
7. isolation × behavior	$(z_i - \bar{z})\left(1 - \max_j(x_{ji})\right)$	↙○→↗ ⟷ ↙●→↗	effect of being isolated in the network on behavior

* In the *effective transitions* illustrations, it is assumed that the behavioral dependent variable is dichotomous and centered at zero; the color coding is ○ = low score (negative), ● =high score (positive), ◍ =arbitrary score. Actor i is the actor who changes color z_i in the transition indicated by the double arrows. Illustrations are not exhaustive.

Figure 3.2: Source:Steglich, C., Snijders T, Pearson M. Dynamic networks and behavior: separating selection from influence. Technical report, under review. Available at: `http://www.gmw.rug.nl/~steglich/sites/index.htm`.

4 Simulation and Emulation Framework

As EvESim is a simulation framework running on digital ecosystem platforms, the term simulation is often used interchangeably with emulation or even testing framework. With the planned migration of the already service oriented implementation of EvESim to the currently developed OPAALS P2P system, described in Chapter 2.2, EvESim will be one of the first running applications on top of this OPAALS infrastructure. Thus, EvESim is not only a simulation framework, able to simulate digital ecosystems and social networks, but also itself a running application in such a digital ecosystem. Consequently, we cannot separate the terms simulation and emulation in every sense for EvESim. Additionally, the research on social networks, described in Chapter 3 will be implemented as behavioral configuration of the system as regards to the usage scenarios. That means, that findings from Social Network Analysis (SNA) can indicate what user behavior is expected in a digital ecosystem and therefore, act as input for both, network emulations of the DVSP as well as simulations on semantic search, for example.

The following sections describe the structure, behavior and results of the first simulations on EvESim. They include already data from SNA of co-authorship networks in Brazil. As soon as the DVSP infrastructure, outlined in Chapter 2.2, is implemented, the same simulations can be done on the OPAALS infrastructure without major adaptations.

4.1 Architecture and Infrastructure

The original intention of EvESim was the simulation of a fully distributed Digital Business Ecosystem. Such simulations of fully distributed systems do not necessarily require fully distributed simulation frameworks. Nevertheless, the need for

complexer modeling, additional features, and better performance resulted in the architecture presented in this paper. The following section provides an overview of the current technical infrastructure and architecture, starting with a brief recap of the initial framework architecture.

4.1.1 Distributed Simulation Framework

The reasons for choosing a service-oriented architecture for EvEsim is described in detail in [27]. Summarising, the main factors for running EvESim in a fully distributed manner are the following:

1. Complexity of models, of Digital Ecosystems and the wide application area.
2. Limitations of the underlying Repast[1] toolkit for Digital Ecosystems simulations.
3. Lack of performance of the simulations.
4. Difficulties in estimating the behaviour of a real P2P system of choice.

The first factor is related to the complexity of models especially when the abstraction level of the simulation is close to reality. For example, the simulation of automatic service composition by comparing natural language service descriptions for networks with more than 100 actors requires considerable computing power.

The utilisation of Repast and the variety of built-in functionality like a basic agent-model, scheduling and visualization functionalities, seemed to make it the perfect choice initially. Nevertheless, the focus of EvESim is network simulations and therefore Repast showed too much overhead during the initial simulations. Additionally, all modifications in the Repast code-base like customized visualizations and changes in the basic agent-model were complex.

Running simulations on a single machine, especially with complex simulation models, lead to simulation runs of more than 24 hours for one simulation. For a comparison of several hundred simulation results or even the adaptation of network configuration parameters based on simulation results, a single simulation needs to be faster for Digital Ecosystem research.

For estimating the real-world behaviour and for showing a new concept like Digital Ecosystems to new potential stakeholders in a wider community, simulations

[1] http://repast.sourceforge.net

with off-the-shelf software packages are not sufficiently convincing. Although there is considerable data available for estimating the network behaviour and performance of various systems, the behaviour of a concrete system needs to be tested and emulated under real-world conditions as well.

4.1.2 Infrastructure

All components of EvESim are implemented as services in compliance with the Service Oriented Architecture (SOA) paradigm [28]. To avoid confusion between the EvESim services and the abstract simulated services in the DE, the EvESim services, which implement the EvESim functionalities, are called *EvESim components* in the following. The services invoked to simulate the usage of services are called *abstract services*, as there is no real-world implementation existing. The user interface for setting parameters and starting the simulation is a classical Java Rich Client.

For instantiating the EvESim components, a service container is needed. As the primary intention of EvESim is the emulation of a DE, the two P2P infrastructure components developed in the course of the DBE project were considered and evaluated: *Soapod* [29] and *Servent* [30]. In [31] as well as on the EvESim Homepage[2], the evaluation of the two DBE-related P2P implementations are compared. Currently Soapod is the service container of choice which consists of a Web Service framework running on top of Apache Tomcat [32] and a service lookup mechanism based on Distributed Hash Tables (DHT) [33]. The broad usage of Tomcat as servlet container, the good scaling properties of DHTs, and the missing support for the Servent mainly lead to the decision to use Soapods. This decision for a Tomcat-based service container works smoothly at the moment. Nevertheless, all EvESim components are designed and implemented independent of the service container. Therefore, with minimal changes of the service interfaces, other service containers can be used as well. The outlook Section 5 of this paper also discusses future plans regarding the infrastructure of EvESim.

4.1.3 Simulation Framework Architecture

In the following we clarify the combination of simulation, emulation and testing in EvESim. In the case of DE simulations, the simulation framework combines all these approaches in one instance. For simulating other networks, the usage of EvESim will

[2] `http://evesim.org`

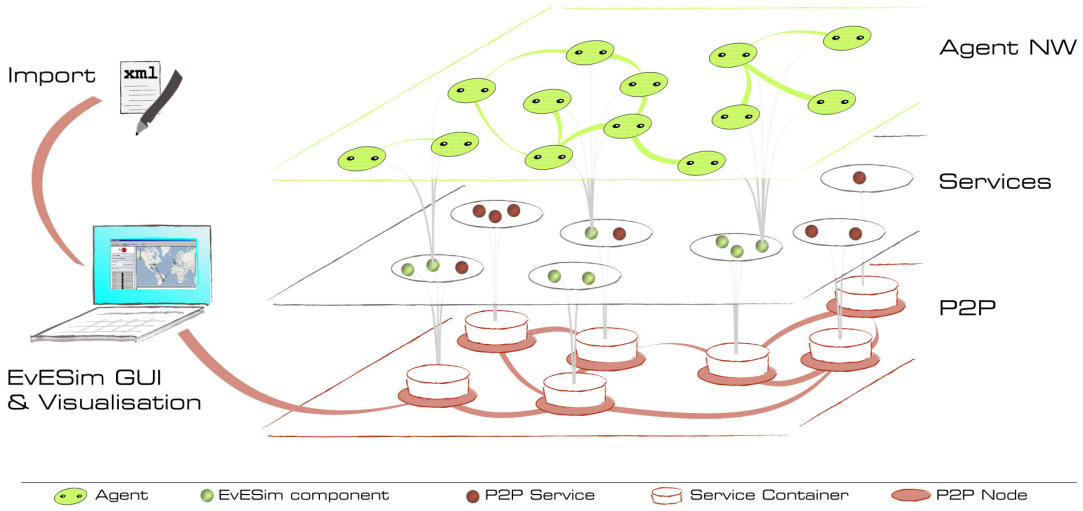

Figure 4.1: Layers of the *EvESim* simulation framwork.

be restricted to a distributed simulation framework, not its emulation and testing potential. Nevertheless, simulations on EvESim always lead to feedback and usage and therefore also lead to user testing and feedback on infrastructure development.

As shown in Fig. 4.1, the EvESim simulation framework runs on top of the P2P infrastructure. This infrastructure is currently a grid of Soapod-based service containers. Services are running within each of these service containers that can be P2P Services with no relation to EvESim or EvESim components. XFire (see [34]) takes care of the communication between the Soapod nodes [29]. Some of these services are EvESim-related *agent pools*. Each agent pool component owns *one to n* agents. Each agent owns a timer which calls specified agent methods at a given frequency. At the initialisation of the simulation, agents are dispersed to all available agent pools in the network. The user interface component only needs an entry point to the network; then the search for available service pools is taken over by the P2P infrastructure.

The initialization of the agent network in the simulation is done as follows: 1) the network topology is initialized, 2) the types of agents, so-called prototypes, are identified, 3) the behaviour of the prototypes can be configured. After these steps are completed the simulation can begin. The network topology and the types of agents can be configured manually in the EvESim user interface, or it can be imported

from an XML file. The structure and usage of this XML file will be explained in Section 4.2 of this paper. The same XML data is also used for the visualisation of the simulated network bases in Google Maps and Google Earth, described below.

4.1.4 EvESim Components and Communication

The EvESim framework consists of obligatory components needed for every kind of simulation and optional components, which are extensions designed and implemented for a concrete simulation case. Beside the user interface for configuring the simulation and for importing network topologies, the following core components of EvESim are currently implemented (additionally see [27], [31]).

- **Agent Pool**: Holds one to n agents which interact independently from each other and have associated timers for calling methods to trigger interaction.
- **Status Report**: By adding a reporting entry in the agent's timer, runtime data is collected for logging and statistical purposes.

Simulation-specific components are the following:

- **Genetic Algorithm**: Optimization of abstract service combinations for clustering and critical mass evaluation in the DBE context [4].
- **Central Service Pool**: A component representing a central repository for critical mass simulation [4].
- **Keyword Search**: Simple keyword search component. Takes as parameters the search string and a list of candidate service descriptions and returns a ranked list of the candidates with fitness value dependent on the matching keywords found in the string (see also Section 4.3 and [28]).

All these components are implemented as services running on Soapod. Technical details on how to build and deploy such components can be found in [29] and [35].

If an agent wants to communicate with an EvESim component, for example with the Keyword Search component it searches the Soapod grid for Keyword Search components by using the Soapod infrastructure. When such a component is found, the agent can directly invoke this component's methods. The advantage of this architecture is that all of P2P services can be called by agents. Consequently, simple EvESim components like a Keyword Search can be replaced later on with

a distributed and semantic searchable repository. At this stage the simulation case can be reused not only to simulate networks, but also to test other P2P components in the same network.

4.2 Import of Network Data and Visualisation

The Eclipse Modelling Framework (EMF) was used as a underlying component for importing data. It provides tools for modelling data and libraries for importing and exporting data corresponding to the previously defined models. The structure described previously is the starting point for the EMF-based import. More concretely, the XML Schema Description (XSD) of this structure is transformed to an *EMF Model*. This EMF model, together with additional information stored for the generation of code called *Genmodel*, is used to generate Plain-Old-Java-Objects (POJO) representing the described structure. The generated code contains the data fields specified in the schema, all of which need getter and setter methods (other information such as methods or exceptions can be included in the EMF model as well but are not currently used in this case). Although the real advantages of this approach appear when developing plug-ins for the Eclipse environment (which is not the case with EvESim), the XML data can easily be loaded and saved using the EMF-XMI functionality, which is responsible for the serialisation and de-serialisation of the XML data including validation against the provided XSD. Besides importing XML data, this approach is also used to export data for the EvESim visualisation component.

One of the key features is the capability to visualise the simulations in a format that can be easily interpreted by users. The latest improvement is the substitution of the existing visualisation component with a Web 2.0 approach: the original visualisation component was dropped and exchanged for two different visualisation techniques running in a Mozilla / XULRunner container within the EvESim. Firstly, Google Maps is used to show the network, including animations, to visualise the service consumption between nodes. And secondly, using the Google Earth Mozilla plug-in, the same information can be visualized in an even nicer way by using an installed Google Earth instance, depicted in Fig. 4.2.

Both visualisation components provide a convenient way to check the location of individual nodes or agents in the network, how these nodes are connected and – even more important – which services they offer and/or consume currently. In order to

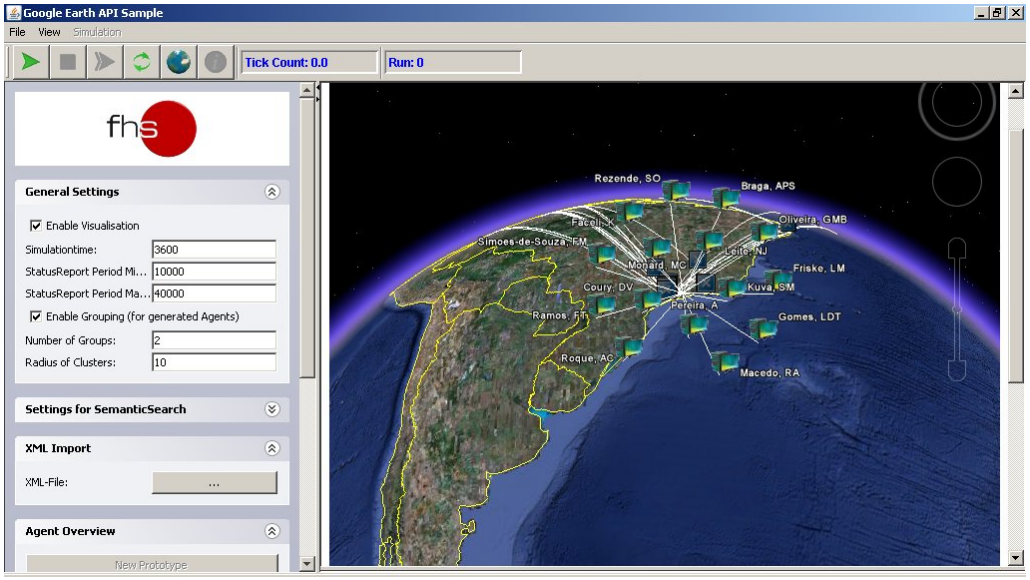

Figure 4.2: *EvESim* with Google Earth Visualisation.

render this geographically correct, the latest version of the data schema was enriched with the required latitude and longitude attributes to add geo-information to the agents.

Technically both of these parts are combined using the MozSwing[3] component which integrates a Mozilla XulRunner[4] Engine into a Java applications. The Google Maps visualisation is working without any additional installation, whereas the Google Earth visualisation needs an installed Google Earth application and an additional Google Earth Mozilla browser plug-in. The advantage of a latter approach is the fundamental performance improvement, as Google Earth is a native application, whereas Google Maps is a JavaScript application running in a web browser. Currently the communication between the components is done via an XML file stored on the file system, which provides a one-way communication from the EvESim feeding the data to the Google Maps or Google Earth component. The next improvment is to enable a two-way-communcation based on an integrated Jetty Servlet Container[5].

[3] `http://sourceforge.net/projects/mozswing`
[4] `http://developer.mozilla.org/En/XULRunner`
[5] `http://www.mortbay.org/jetty/`

4.3 Semantic Search and Recommender

In parallel with the development of EvESim as a simulation framework, so-called simulation cases were implemented as proof-of-concept and for finding key factors of the configuration and applicability of Digital Ecosystems. Simulation cases are implemented simulations which reflect use-cases in reality and usually intend to emulate this use-case by letting the agents take over the actions of the actors in real life. One of the first simulation cases was the search for the most relevant factors of a critical mass of services and actors for digital ecosystems and self-organised clustering of SME networks [4].

The recent simulation case in EvESim is a combination of different semantic search and recommendation mechanisms. It is intended to combine best practice methods for searching with biologically inspired approaches to data distribution in digital ecosystems. Here again it is intended to replace the simple keyword search component of EvESim with later P2P-based semantic search components, currently being developed in OPAALS.

4.3.1 Network Exploration

For the search algorithm we are using two traditional approaches from P2P systems for node lookup and utilising them for search service. Firstly, the *breadth-first search* (BFS), used by Gnutella, is used as an example. The search begins at the requesting agent and explores all neighboring nodes. From there, it also explores the next level - neighbors of neighbors - until a given Time To Live (TTL) of the search request. For Gnutella the typical maximum TTL is 7. According to [36], about 95% of nodes can be reached with a TTL of 7 [37],[38].

And secondly, a modified *depth-first search* (DFS), which typically is implemented in Freenet [39], is compared to the BFS. In Freenet typically the network is explored node-by-node until a goal node is found. Iterative deepening is used in this simulation case in order to, first, search for services at all direct neighbors, second, assess the results, and third, if a service with fitness higher that a given threshold is found, end the search at this level. If the fitness threshold is not satisfied, the network is explored further in depth until a TTL boundary is reached again.

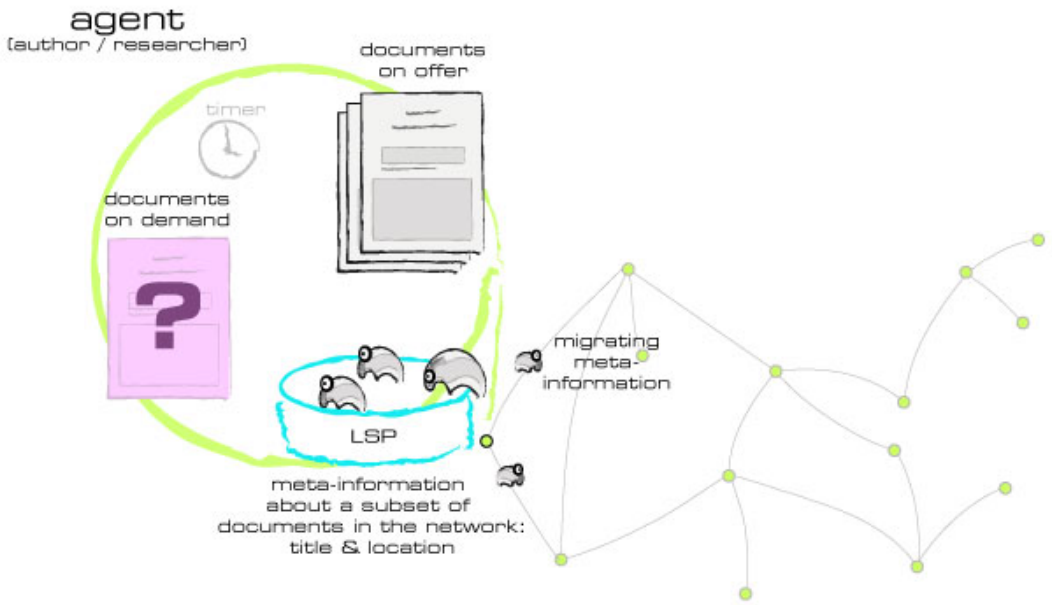

Figure 4.3: Agents using Local Service Pools.

4.3.2 Simulation Model

Figure 4.3 shows the basic model of the semantic search simulation case. Each agent in the network has a timer which triggers the searches. Additionally, each agent owns a list of keywords on demand, which are requested by the agent when the timer triggers the search. Both the on-demand list and the on-offer list of documents are abstracted service descriptions, which are represented by the title of the document. Consequently, if a request for a document is triggered, the agent searches the network by calling the previously described keyword search component of EvESim. This component takes as parameters the title of the requested document and a list of candidate document titles. Which candidates are passed on for search will be explained in the next paragraphs. The return value is a ranked list of documents based on the matching keywords. Currently, the agent picks the highest ranked document and remembers it together with the request. In order to enhance future searches, a new connection is established between the requesting agent and the owner of the document. This behavior leads to a dynamic change of the network infrastructure based on the documents found in the network. Connections are set

up between nodes which have common interests and future searches speed up.

The candidates for the search in this simple scenario are the documents in the on-offer list of an agent. That means that every actor / agent in the network has a list of documents which were written by this actor. This list is searched for the requested document until the TTL of the request is reached. The results of the searches are returned to the originally requesting agent. We refer to this search as *pull-approach*, because the documents are distributed by pulling for a requested document.

Contrary to the pull approach, we introduced a biologically inspired *push approach*. This work is inspired by one of the principles of the Evolutionary Environment (EvE) in the DBE project [28]. The underlying concept is one of migrating individuals in nature. In the simulation case at hand, each agent has a local service pool (LSP) (see Fig. 4.3), represents the environment in which documents can 'live'. Living in this case means that the documents, i.e. document titles, which are most probably interesting for that agent, concentrate in the LSP of an agent. The interest, and therefore the search requests of an agent, define in that case the environment of documents. The idea is to have a browsable LSP which acts as a dynamic recommender for possibly interesting readings and in parallel as the starting point for searches.

The name pull in this case is chosen because if an agent searches for a document and finds it in the network, this agent copies the document description in its own LSP and pushes it to all its neighbours LSPs as well. This is based on the assumption that the neighbours of a node also have similar reading interests. As can be seen in the first results section, we introduced to this migration/propagation of document descriptions of neighbors the additional propagation of a number of random nodes in the network, which is needed in order to connect island networks.

The following simulation results are initialized within a social network of coauthorship by researchers from all over the world.. The data was collected by the Instituto de Pesquisas em Tecnologia e Inovação[6] in Brazil. We restricted the area of those physics-related publications which lead to a network of 476 researchers. The first author of a paper as the owner and the respective agent gets an entry managing the paper title in its on-offer list. The on-demand lists of the agents are then initialised with on-offer lists of other agents in the network. The LSPs are filled randomly with document descriptions in the network with a total redundancy of 5.

[6] `http://ipti.org.br`

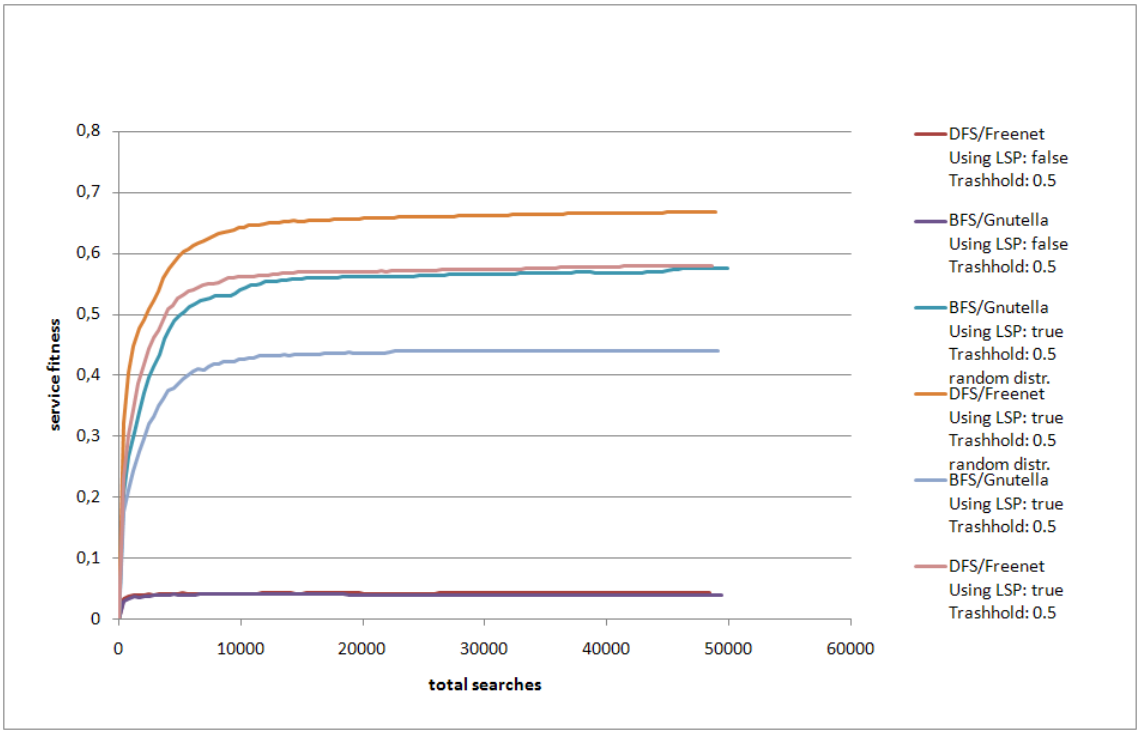

Figure 4.4: Fitness development of the simulated network.

This means that every document on offer in the network is copied randomly into 5 LSPs in the network.

For all simulation runs a threshold of 50% fitness is assumed, which means that an agent accepts only search results if the fittest service has at least a 50% match with all keywords within the result. If the threshold is not reached, the search will be started again later in the simulation.

As can be seen in Figure 4.4, only searches which use the LSPs for service-discovering reach fitness values exceeding 0.4, whereas search methods, without LSPs do not even achieve a value of 0.1. Because matched services are migrated only to other agents' LSPs, better services located further away than seven hops cannot be found. This leads to a low average service fitness for searches without using LSPs. The best results are reached with the depth-first traversal search method, due to the frequent propagation of services. In addition, it can be observed in the simulations that the average number of hops for successful network searches decreases with the number of searches executed. However, this is only true for searches also using local service pools.

5 Conclusion and Outlook

As the work at hand tries to find cross linkages or mixing points between social science, computer science and ecological and biological phenomena, we want to point out here that this linkage is not trivial, and needs better learning and understanding of all the disciplines involved. Such a better understanding by the authors from the other discipline is already one significant outcome of this work.

This report has also shown theoretical work on computer science as well as social science and presents a partial implementation of these findings in the current EvESim framework.

We have described a formal framework for the coordination of concurrent and distributed service invocations involved in a long-running transaction. In particular, we have used the transaction trees (to capture the structure of a transaction) proposed in Deliverable D3.2 and UML sequence diagrams (to describe the interactions involved) in the specification of a transaction. We then showed how a formal description of this initial transaction context can be used to reason about the corresponding behavioural scenarios and identify implicit scenarios of interaction which may indicate emergent behaviour. We have seen that our formal framework can determine the complete set of behaviours and make all possible scenarios explicit in the corresponding scenario-based specification.

We have also provided schemas for deriving XML representations of both a transaction context and the corresponding transaction scripts (reflecting transaction vectors) which is what is needed in order to refine the behavioural scenarios of the initial transaction context. The source files, together with some example scripts, can be found online following [1]. The material presented in Chapter 2 of this deliverable report draws upon work that has been published in ETAPS 2008 [2] and IEEE-DEST 2007 and 2008 [13], [3].

A stochastic perspective was introduced for simulation, using co-evolution models. This method is more explored as an analysis tool in D10.8, and further details

can be found there. The perspective to understand network autocorrelation as two separated, but dependent processes, is of great value for social sciences and new results about social network are now giving a new understanding of how collaboration and interaction evolves to a rich network in terms of social-capital, or its closure. The analogy for the P2P and transactions may not be straightforward, given the different kinds of nodes with despair complexity, but can be a good exercise to integrate research fields from different domains, like Statistics, Computer Science and Social Science.

Regarding EvESim, in this report the emulation potential for multi-agent simulations is demonstrated in the context of Digital Ecosystems by utilizing a SOA-based approach. The architecture of the system is discussed in terms of the challenges of distributed simulations, and the relevant infrastructural components are described in detail. Two variants of the visualisation component are introduced together with the functionality of XML-based data import for network topologies. The mechanism of enhanced service lookup and discovery is detailed and its benefits are demonstrated in simulation results. Future work will address the substitution of Soapod by the general-purpose P2P framework JXTA, the technology of choice in the OPAALS project community.

To cover the huge state space more efficiently in the simulations, it is intended to utilise genetic algorithms to automatically run simulations with varying key parameters for adaptively tuning these parameters to specific social network simulations. Also planned is to extend the simulations of social networks to investigate processes like wiki-based cooperation, online conferences, and joint-paper authoring work flows. These activities will support our ongoing work regarding the concept of '*knowledge services*', intended to foster creation, usage, and exchange of knowledge in dynamic digital societies.

Bibliography

[1] http://personal.cs.surrey.ac.uk/personal/pg/A.Razavi/ppna/.

[2] S. Moschoyiannis, A. Razavi, and P. Krause. Transaction Scripts: making implicit scenarios explicit. In *ETAPS 2008 - Formal Engineering approaches to Components and Architectures (FESCA'08)*, ENTCS. Elsevier, 2008. To appear.

[3] S. Moschoyiannis, A. Razavi, Y. Zheng, and P. Krause. Long-Running Transactions: semantics, schemas, implementation. In *IEEE Digital Ecosystems and Technologies (DEST 2008)*. IEEE Computer Society, 2008.

[4] T. Kurz and T. Heistracher. Simulation of a self-optimising digital ecosystem. In *1st IEEE/IES Conference on Digital Ecosystems and Technologies*, 2007.

[5] A. Razavi, S. Moschoyiannis, and P. Krause. Concurrency Control and Recovery Management for Open e-Business Transactions. In *Communicating Process Architectures (CPA 2007)*, pages 267–285. IOS Press, 2007.

[6] P. Furnis, S. Dalal, T. Fletcher, A. Green, A. Ceponkus, and B. Pope. *Business Transaction Protocol, version 1.1.0.* available from www.oasis-open.org/committees/download.php/9836, November 2004.

[7] F. L. Cabrera, G. Copeland, J. Johnson, and D. Langworthy. *Coordinating Web Services Activities with WS-Coordination, WS-AtomicTransaction, and WS-BusinessActivity.* http://msdn.micorsoft.com/webservices/default.aspx, January 2004.

[8] M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, and B. J. Kramer. Service-Oriented Computung Research Roadmap. In *Dagstuhl Seminar Proc. 05462, Service-Oriented Computing (SOC)*, pages 1–29, 2006.

[9] S. Moschoyiannis, P. J. Krause, and M. W. Shields. A True-Concurrent Interpretation of Behavioural Scenarios. In *ETAPS 2007 - FESCA'07*, ENTCS. Elsevier, 2007. To appear.

[10] S. Moschoyiannis, M. W. Shields, and P. J. Krause. Modelling Component Behaviour Using Concurrent Automata. In *ETAPS 2005 - FESCA'05*, volume 141 of *ENTCS*, pages 199–220. Elsevier, 2005.

[11] M. W. Shields. *Semantics of Parallelism*. Springer-Verlag London, 1997.

[12] OMG. *Unified Modeling Language: Superstructure, version 2.0*. OMG document formal/05-07-04, available from http://www.omg.org, August 2005.

[13] A. Razavi, S. Moschoyiannis, and P. Krause. A Coordination Model for Distributed Transactions in Digital Business Ecosystems. In *IEEE Digital Ecosystems and Technologies (DEST 2007)*. IEEE Computer Society, 2007.

[14] S. Moschoyiannis. *Specification and Analysis of Component-Based Software in a Concurrent Setting*. PhD thesis, University of Surrey, 2005.

[15] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge Mathematical Textbooks, Cambridge University Press, 1990.

[16] M. W. Shields. Concurrent Machines. *Computer Journal*, 28:449–465, 1985.

[17] A. Mazurkiewicz. Basic Notions of Trace Theory. In *Linear Time, Branching Time and Partial Orders in Logics and Models for Concurrency*, volume 354 of *LNCS*, pages 285–363. Springer Verlag, 1988.

[18] J. Küster-Filipe. Modelling Concurrent Interactions. *Theoretical Computer Science*, 351(2):203–220, 2006.

[19] C. A. Petri. Introduction to General Net Theory. In *Proceedings of Advanced Course on General Net Theory of Processes and Systems: Net Theory and Applications*, volume 84 of *LNCS*, pages 1–21. Springer-Verlag, 1979.

[20] S. Moschoyiannis and M. W. Shields. A Set-Theoretic Framework for Component Composition. *Fundamenta Informaticae*, 59(4):373–396, 2004.

[21] T.A.B. Snijders. *Stochastic actor-oriented models for network change.* 21: 149 : 172. Journal of Mathematical Sociology, 1996.

[22] T.A.B. Snijders. *The Statistical Evaluation of Social Network Dynamics.* Sociological Methodology. Boston and London: Basil Blackwell, 2001.

[23] T.A.B Snijders. *Models for Longitudinal Network Data.* Models and methods in social network analysis. Cambridge University Press, 2005.

[24] Snijders T Pearson M. Steglich, C. Dynamic networks and behavior: separating selection from influence. Technical report, under review. Available at:http://www.nuffield.ox.ac.uk/nnnr/papers.htm.

[25] Steglich C. Schweinberger M. Snijders, Tom A.B. Modeling the co-evolution of networks and behavior. chapter 3. In *Longitudinal models in the behavioral and related sciences*, pages 41–71. Mahwah NJ, Lawrence Erlbaum, 2007.

[26] UniS. D3.2 - Report on formal analysis of autopoietic P2P network, together with predictions of performance. OPAALS Project, August 2007.

[27] T. Kurz, J. Noguera, R. Eder, and T. Heistracher. Peripheral simulation framework for digital ecosystems. In *1st international OPAALS Conference on Digital Ecosystems*, 2007.

[28] G. Briscoe. Digital ecosystems: Evolving service-oriented architectures. In *IEEE First International Conference on Bio Inspired mOdels of NETwork, Information and Computing Systems*, 2006.

[29] Soapod Central. Soapod - P2P Service-oriented Application Server. `http://www.soapod.org`, Last Accessed: 15/04/2009.

[30] The DBE Consortium. Servent - DBE Execution Environment. `http://swallow.sourceforge.net`, Last Accessed: 15/04/2009.

[31] T. Kurz R. Eder C. Adelberger and T. Heistracher. Evaluation of two p2p-approaches for a distributed simulation framework. In *2nd international OPAALS Conference on Digital Ecosystems*, 2008.

[32] The Apache Software Foundation. Apache Tomcat. `http://tomcat.apache.org`, Last Accessed: 15/04/2009.

[33] Hari Balakrishnan, M. Frans Kaashoek, David R. Karger, Robert Morris, and Ion Stoica. Looking up data in P2P systems. *Commun. ACM*, 46(2):43–48, 2003.

[34] Codehaus XFire Team. XFire. `http://xfire.codehaus.org`, Last Accessed: 15/04/2009.

[35] Salzburg University of Applied Sciences. EvESimulator. `http://www.evesim.org`, Last Accessed: 15/04/2009.

[36] Schahram Dustdar, Harald Gall, and Manfred Hauswirth. *Software-Architekturen für verteilte Systeme*. Springer, 2003.

[37] Beverly Yang and Hector Garcia-Molina. Improving search in peer-to-peer networks. In *ICDCS*, pages 5–14, 2002.

[38] Clip2 and The Gnutella Developer Forum. The gnutella protocol specification v0.4.

[39] I. Clark. *A Distributed Decentralised Information Storage and Retrival System*. University of Edinburgh, 1999. `http://freenetproject.org/papers/ddisrs.pdf`, Last Accessed: 15/04/2009.