## Contents

Towards a full-edged support for designing cloud applications .......................... 5  
*Jacopo Soldani, Supervisor: Antonio Brogi*

QoS prediction and QoS-aware service selection in service composition .......... 11  
*Ahmad Ibrahim, Supervisor: Antonio Brogi*

Artifact-centric Choreographies: Coordinating Flexible Cross-organizational Processes .......................... 17  
*Kristof Hamann, Supervisor: Winfried Lamersdorf*

Termination Analysis of Concurrent Service-Oriented Systems ......................... 23  
*Mandy Weißbach, Supervisor: Wolf Zimmermann*
Preface

The goal of the PhD-Symposium is to provide a forum for PhD students to present and to discuss their work with senior scientists and other PhD students working on related topics. As for the main conference, the topics focus on all aspects of Cloud Computing, Service Oriented Architectures, Web Services, and related fields. In contrast to the main conference, this work is usually unfinished or has just been started in the PhD projects. The programme committee carefully selected four contributions. Each submission was reviewed by three PC-members. In addition to the precise description of the problem to be solved, preliminary results, and first ideas for solving the main problem, the contributions also include a workplan. All these issues are discussed at the symposium with selected senior scientist and the PhD students.

We are grateful to the conference organizer Massimo Villari for his organisatorial support. We thank the international programme committee consisting of

- Einar Broch Johnsen (University of Oslo, Norway)
- Friederike Klan (Friedrich Schiller University Jena, Germany)
- Welf Löwe (Linnaeus University, Sweden)
- Ernesto Pimentel (University of Malaga, Spain)
- Alexander Pokahr (University Hamburg, Germany)
- Emilio Tuosto (University of Leicester, UK)

for their work. Finally we would like to thank all authors - no matter whether their contribution has been accepted.

Gianluigi Zavattaro
Wolf Zimmermann
PhD Symposium Co-Chairs

Bologna, September 10, 2015
Halle, September 10, 2015
Towards a full-fledged support for designing cloud applications

Jacopo Soldani
Dept. of Computer Science, University of Pisa, Italy

Abstract. In this paper we show how the development and maintenance of cloud applications can be eased by providing a full-fledged support for their design. We first motivate the need for a vendor-agnostic representation of cloud applications and their management. We also motivate the need for techniques for analysing cloud applications, and for reusing existing applications. We then show our first results in the field, and we discuss how we intend to proceed with our research.

1 Introduction

Cloud computing has revolutionized IT by permitting to run on-demand distributed applications at a fraction of the cost which was necessary just a few years ago. However, current cloud technologies suffer from a lack of standardization, with different providers offering similar resources in a different manner. As a result, cloud developers tend to remain locked in a specific platform environment because it is practically unfeasible for them, due to high complexity and cost, to migrate their applications to a different platform. According to [3], to enable the creation of portable cloud applications and to automate their management, application components, their relations and management should be modeled in a standardized, machine-readable format.

In this perspective, various metamodels are emerging. TOSCA [11] (Topology and Orchestration Specification for Cloud Applications) is an OASIS standard to enable the specification of portable cloud applications and the automation of their management. It indeed provides a modeling language which allows to formalize the structure of a cloud application as a typed topology graph, and the management tasks as workflow plans. The Aelous component model [9] permits specifying various characteristics of cloud application components, including the fact that their interfaces vary depending on the component state. It then employs this information to automatically provision cloud applications, and to help administrators with their management. Nevertheless, none of these approaches (nor others like CloudML [10], GENTL [1], etc.) has yet been widely accepted as the de-facto standard for cloud application design.

In short, to support the design of portable cloud applications, there is a need for identifying a vendor-agnostic model that enables the specification of

* Supervisor: Antonio Brogi (Dept. of Computer Science, University of Pisa, Italy)
both their structure and management. The availability of techniques for the
analysis and reuse of cloud applications is also crucial to support their design.
The former indeed permit verifying whether cloud applications are accurately
designed, while the latter avoids developers to design and/or develop multiple
times recurring components.

The objective of our work is precisely to identify a suitable representation
for cloud applications, and to develop analysis/reuse techniques based upon (but
not limited to) such representation. It this way, we aim at providing a full-fledged
support for designing complex cloud applications\footnote{Due to space limitations, we cannot further discuss related work. A more extended
discussion can be found in our previous work [4,5,6,7,8,14].}.

\section{Motivations and Problem Statement}

Consider a typical issue that arises when developing Web-based applications:
where and how to host them? Typically, multiple cloud providers are appropriate
as many focus on such kind of applications, and finding the appropriate solution
is a time-consuming, cumbersome challenge.

Suppose that we need to host a PHP application on a cloud environment,
along with a MySQL database containing application data. Currently, we are
required to select the appropriate providers and to explicitly describe the config-
uration of our application on those providers. Furthermore, if we decide to
migrate (part of) our application to another provider, we might have to redesign
its configuration, even from scratch. It would be much better to abstractly de-
scribe the desired hosting environment, and to provide such description as input
to a method which automatically derives a (verified) solution to host our appli-
cation over a single or multiple clouds.

A suitable \textit{modelling} for cloud applications would allow us to describe our
application in a way analogous to that sketched in Figure 1. The figure illustrates

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{motivating_scenario.png}
\caption{Motivating scenario.}
\end{figure}

our motivating scenario by explicitly drawing the components of our application
(viz., \texttt{PHPModule} and \texttt{MySQL Database}), the environments needed to host such
components (viz., \texttt{PHP Hosting Environment} and \texttt{MySQL Hosting Environment}),
and the dependencies occurring among the components and the environments.
The description should not be restricted to the basic information illustrated in
Figure 1. It should rather include the technological and QoS characteristics of
each application component, as well as the management operations they offer.
The modelled information can then be employed by matchmaking and adaptation techniques to *reuse* existing solutions and automatically produce one or more hosting solutions for the desired application. Figure 2 exemplifies a possible solution: The **PHP Hosting Environment** is obtained by reusing an **Apache WebServer** hosted on an **Azure** PaaS environment, while the **MySQL Hosting Environment** is implemented by reusing a **MySQL DBMS** installed on an **Ubuntu** operating system, which is in turn hosted on an **AmazonEC2** virtual machine.

Adapting existing solutions may be not enough to ensure that they will properly work when substituting the desired components or environments. For instance, we may expect the **PHP Hosting Environment** to provide a certain feature after performing a certain operation, but when invoking such operation on the **Apache WebServer** the desired feature may not be provided. Furthermore, we may plan to invoke the operation of the **MySQL Hosting Environment** in a certain order, but the same order may not be supported by the software stack implementing it. We thus need *analysis* techniques capable to verify the coherence of an application description, and to understand whether all the components’ functional and non-functional requirements are satisfied.

### 3 Research Challenges

As we already mentioned, in order to provide a full-fledged support for designing cloud applications we have to address three research challenges.

**(R1)** **Suitable modelling for cross-cloud applications.** We aim at identifying a vendor-agnostic, compositional model that permits specifying the structure of an application, and its management. More precisely, the model should permit describing the components of a cloud application (in terms of the technological and QoS characteristics they feature, the management operations they offer, the behaviour of such operations, etc.) and the relationships occurring among these components. It should also permit describing the management tasks of a whole application (e.g., by orchestrating the operations offered by its components).
(R₂) **Reuse of existing applications.** To ease the design and development of cloud applications, we also aim at fostering the reuse of existing solutions. This can be done from two different perspectives: (i) matchmaking and adaptation of existing applications with required components, and (ii) identification of common management patterns.

(R₃) **Analysis and verification of application specifications.** We aim at developing analysis techniques that inspect cloud application specifications and verify whether they are accurately designed. These techniques should permit verifying whether the functional and/or non-functional requirements of an application’s components are effectively satisfied, or whether their operations are orchestrated coherently with their behaviour.

4 First results and Future Work

As a first step of our research work, we focused on determining a suitable representation for cross-cloud applications, i.e. a vendor-agnostic metamodel offering a good compromise among many comparative dimensions (like expressive power, ease of use, non-ambiguity, etc.).

As we highlighted in [8], a promising candidate in this perspective is the TOSCA standard [11]. It indeed provides a modelling language to describe the structure of a cloud application (called ServiceTemplate) as a topology graph, whose typed nodes represent the application components, and whose typed relationships represent the relations occurring among the application components. TOSCA also permits specifying the management tasks of an application as Plans, i.e. workflows orchestrating the management operations offered by the components in an application topology.

TOSCA supports compositionality, by permitting to instantiate a NodeType by matching an existing ServiceTemplate [12]. While such matching is mentioned with reference to an example, no formal definition of matching is given either in [11] or in [12]. In [4], we thus formalized the notion of *exact matching* between a TOSCA ServiceTemplate and a NodeType, by requiring the former to offer the very same features as the latter to match it. We then defined three other types of matching (*plug-in, flexible* and *white-box*), each permitting to ignore larger sets of non-relevant syntactic differences when type-checking ServiceTemplates with respect to NodeTypes. In [7], we then described how ServiceTemplates that plug-in, flexibly or white-box match NodeTypes can be suitably adapted so as to exactly match them.

The approach in [4,7] permit instantiating desired NodeTypes by matching entire ServiceTemplates. In [14], we went a step further by allowing to reuse not only entire applications, but also fragments of such applications.

In both [7] and [14], we adapt a matched ServiceTemplate $S$ to exactly match a desired NodeType $N$. Thanks to the powerful way in which TOSCA supports the deployment of cloud applications, this does suffice to reuse any actual application modelled by $S$ to provision cloud applications that rely on $N$. TOSCA permits to pack in a CSAR (*Cloud Service ARchive*) file an application
specification together with the actual executable files to be deployed on a cloud platform. When a CSAR file is given in input to a TOSCA container, the latter takes care of deploying and executing the application specification contained in the CSAR file [12]. Therefore, in order to adapt a ServiceTemplate $S$ so as to deploy an application that relies on a NodeType $N$, it suffices to adapt $S$ into a new ServiceTemplate that exactly matches $N$ — without having to generate any implementation of the adaptation specified by $S$.

On the other hand, all the aforementioned approaches are either purely syntactical, or rely on ontologies to overcome non-relevant syntactic differences. This is because TOSCA does not yet provide a way to describe the behaviour of an application component. More precisely, TOSCA permits to specify the states, requirements, capabilities and management operations of an application component, but not how they interact each other. In [5,6], we proposed a simple extension of TOSCA that permits specifying the management protocol of an application component, i.e. the behaviour of its management operations and their relations with its states, requirements, and capabilities. We also showed how to compose the management protocols of the components building up a ServiceTemplate, and how such composition permits to automate various useful analyses, like determining the validity of a management Plan, which are its effects, or which Plans allow to reach certain application configurations. In [5], we also illustrated a proof-of-concept implementation allowing to edit the management protocol of a TOSCA NodeType, and to perform some basic analysis on the management of a TOSCA ServiceTemplate. We now intend to extend such implementation in a working prototype to support more complex analyses and to fully integrate it with the OpenTOSCA open source environment [2].

We also plan to exploit management protocols to extend the reuse techniques we proposed in [4,7,14]. More precisely, we first plan to define the notion of management protocol simulation [13], and then to introduce two novel types of behaviour-aware matching (exact and plug-in) which properly extend the corresponding syntactic notions. Then, by further relaxing the notion of simulation (e.g., by permitting to match a desired operation with sequences of available operations), we could further extend our matching notions. Subsequently, we will also extend our adaptation approach [7] to still be capable of adapting a plug-in matching ServiceTemplate and reusing it to instantiate a target NodeType.

Other interesting directions that we plan to investigate are the following. First, we intend to provide an integrated implementation of both the syntactic matching we proposed in [4] and the behaviour-aware matching we mentioned above. Furthermore, we also plan to extend the application fragment matchmaking we proposed in [14] by including the behaviour information of management protocols, and we intend to investigate other weaker notions of simulations to further relax our behaviour-aware matching notions.

Finally, it is worth noting that all aforementioned techniques are capable of determining whether a concrete application is type-compatible with a desired component, by also taking into account their management behaviour. An interesting direction for future work is thus to check not only whether an existing
application type-checks a desired component, but also whether the actual “values” of their features are coherent. This can be done by developing analysis techniques capable of determining the satisfaction of technological and/or QoS properties, and by including such techniques into our matchmaking approaches.

References

QoS prediction and QoS-aware service selection in service composition

Ahmad Ibrahim, Supervisor: Antonio Brogi
Department of Computer Science, University of Pisa, Italy
ahmad@di.unipi.it

1 Context & overview

Web services provide an easy way of managing businesses in general and in cloud computing. Due to complex business requirement, generally one service is not enough to satisfy all requirements and many services have to be integrated together to provide certain functionality. For example an Internet application can call different service’s, i.e., a stock-trading Web service can invoke a payment service which can then invoke an authentication service. The integration of services requires a (partial) order to join these services together (also called workflow) along with concrete services satisfying the goal. For this integration process (also called service orchestration) those services are selected, which can fulfill the functional requirements. Another aspect to consider during this orchestration is to fulfill the non-functional requirements (also called Quality of Service (QoS) [1] or Service Level Objectives (SLO)). Examples of QoS parameters include up time, average time between failures, average time to repair, data rate etc.

QoS of a service orchestration does depend on the QoS of the services it invokes. And the actual QoS featured by an invoked service depends on various conditions (e.g., network, server workload, and so on). Moreover, since complex applications can invoke multiple external services, a QoS change in one or more of those external services can seriously impact on the performance of the whole application.

One (obvious) way to estimate the QoS of a service composition is to deploy it on some infrastructure and to measure QoS parameters over a sufficiently high number of executions. Unfortunately such an approach may be expensive both in time and in monetary cost (if non-free services are invoked), and it may also be not effective if some service invocations have side-effects. On the other hand, predicting the QoS of service orchestration is challenging due to following four characteristics:

1. **Different results of service invocations.** Each invoked service can return a successful reply, a fault notification, or even no reply at all. If a fault is returned, the orchestration will execute a fault handling routine instead of the normal control flow. If no reply is received, the orchestration may get stuck waiting for a reply (unless some parallel branch throws a fault). In either case, the QoS of the orchestration will differ from the case of successful reply.
2. **Non-determinism in the orchestration workflow.** Different runs of the same orchestration can yield different QoS values since the control-flow of the orchestration workflow is non-deterministic. On the one hand, different runs of the orchestration can get different service invocation results (success/fault/no reply). On the other hand, some control-flow structures (alternatives and iterations) depend on input data which may differ in different runs.

3. **Correlations among workflow activities.** The above two characteristics suggest to employ a probabilistic approach. However, it is important to observe that the naive solution of assigning independent probabilities to workflow activities (e.g., as in [2]) can lead to incorrect results.

4. **Complex dependencies among workflow activities.** The control flow imposed by synchronizations on parallel activities (i.e., when a task needs to wait for another to complete before starting) is more expressive than what is allowed by parallel execution only (with synchronization barriers at the end of parallel tasks). This means that workflows which have complex synchronization structures (e.g., as those that can be specified with WS-BPEL synchronization links [3]) cannot be always decomposed in terms of parallel and sequential compositions [4].

## 2 Research Problem

The focus of this research is to predict the quality of service for the orchestrated service as well as how to select services based on QoS parameters. If c is a composition of (types of) services $S_1, S_2, \ldots, S_n$, then we want to investigate

1. What is QoS of c?
2. Is it good idea to replace $S_j$ with $S_j'$?
3. Let goal be the desired QoS of c. Does c achieve goal? If not, can we replace some $S_j$ so that QoS ($c$) satisfy goal?

The QoS attribute we want to focus are Response time, Cost and Reliability.

**Prediction of QoS of service orchestration c** Prediction involves determining QoS ($Q_c$) of a service orchestration c from QoS of given services ($Q_{S_1}, Q_{S_2}, \ldots, Q_{S_m}$). Given $Q_{S_1}, Q_{S_2}, \ldots, Q_{S_m}$ corresponding to respective services, can we predict the $Q_c$ of overall orchestration? If yes, then how, and if not then why not? The prediction of QoS for c can be defined by function f which takes workflow and QoS of given services and predict the $Q_c$.

$$f(\alpha(c), Q_{S_1}, Q_{S_2}, \ldots, Q_{S_m}) = Q_c$$

Where f is a function that predicts the $Q_c$. $Q_c$ is the QoS of the composed service. $Q_{S_1}, Q_{S_2}, \ldots, Q_{S_m}$ are the QoS of the individual services in the composition. $S_1, S_2, \ldots, S_m$ are the individual services part of the composition. A type of service $S_i$ can be bound to a specific endpoint $e_i$. $\alpha(c)$ is an abstract representation of the composition c, namely (i.e., a workflow modeling the control-flow
of the composition). It represents workflow composing types of services. The
motivation behind $\alpha(c)$ as one of the input of $f$ is that, without knowing the
workflows, prediction would be difficult. For instance, let $c$ be a composition of
$S_1$ and $S_2$ then different possible workflows could be

- C invokes both $S_1$ and $S_2$ in parallel and terminates as soon as it receives
  first answer,
- C invokes both $S_1$ and $S_2$ in parallel, combines the results obtained and then
  terminates, or
- C invokes first $S_1$ and then $S_2$.

From the three cases above, it is clear that the way services are used in
a workflow, can affect the QoS of the resulting composition. This is why the
workflow $\alpha(c)$ is a necessary input for the function $f$.

Examples The following examples (Figure 1) show different scenarios in which
some services are bound to a specific endpoint $e_i$ while other are not bound and
where prediction of QoS of service orchestration $c$ can be applied.

So given $\alpha(c)$, $QS_m$ for services, function $f$ can calculate QoS ($Q_c$) of the
service orchestration $c$.

Ex 1.1 Both services are not bounded i.e. $f(\alpha(S_1, S_2), QS_1, QS_2)$ (Figure 1a).
Ex 1.2 One service is bounded with some endpoint ($e_{99}$) while other is not
bounded i.e. $f(\alpha(e_{99}, S_2), Qe_{99}, QS_2)$ (Figure 1b).
Ex 1.3 Both services are bounded with some endpoints ($e_{99}, e_{214}$) i.e. $f(\alpha(e_{99}, e_{214}),
Qe_{99}, Qe_{214})$ (Figure 1c).

3 Proposed Solution & results achieve so far

Our proposed algorithm to predict the QoS of service orchestration (function $f$)
has been implemented in F#.Net. For details about algorithm, the interested
readers can refer to [5,6]. As a proof of concept, we have built an open source
tool PASO (Probabilistic Analyser of Service Orchestrations)\(^1\). In terms of in-
put/output behaviour (Figure 2), PASO inputs:

\(^1\) The source code of PASO is available at https://github.com/upi-bpel/paso.
– A WS-BPEL [3] workflow² defining a service orchestration, and
– A file containing annotations of probabilities for outcomes and costs of service invocations, as well as for the truth of the guards of if and while activities.

and it can output histograms and pie charts summarizing the results of the performed analysis.

![Monte Carlo Simulation](Image)

Fig. 2: Bird-eye view of the input-output behaviour of PASO.

4 Related Work

QoS Prediction is not new and many approaches and tools (e.g., Palladio [7], Kieker [8]) have been proposed. Due to space limitations, we briefly name the approaches closely related to ours which include Cardoso [9], Mukherjee et al. [2,4], Zheng et al. [10], Ivanovic et al. [11]. To the best of our knowledge, none of the previous approaches completely address the problem of arbitrary dependency structures, fault handling, correlations among activities and unbounded loops(Table 1).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-determinism in the orchestration workflow</td>
<td>Weak</td>
<td>Weak</td>
<td>Good</td>
</tr>
<tr>
<td>Correlations among workflow activities</td>
<td>Weak</td>
<td>Weak</td>
<td>Good</td>
</tr>
<tr>
<td>Complex dependencies among workflow activities</td>
<td>Weak</td>
<td>Good</td>
<td>Good</td>
</tr>
</tbody>
</table>

The Table legend are: Good(Address the challenge), Fair(Partially address the challenge), Weak(Does not address the challenge)

² PASO is able to analyse a subset of WS-BPEL structural (sequence, flow, if, while, scope, and faultHandlers) and basic (invoke, assign, receive, reply) activities.
5 Possible direction for future work

We think that there are four possible directions to extend our work.

5.1. QoS aware selection of services  QoS aware selection of services involves selection of services given some goal. Given a QoS objective, which set of services is needed to meet the target? Can the target be achieved? If yes, then how? The QoS aware selection of services can be defined by function $f'$ which takes workflow, goal in the form of QoS objectives, $QS_i$ of the (types of) services, set of services and output endpoint(s) fulfilling the goal.

$$f'(goal, \alpha(c), QS_1, QS_2, ..., QS_m, SetofServices) \ni e_1/S_1, ..., e_m/S_m$$

Where $f'$ is a function. Goal is the QoS objective of the overall composition. $\alpha(c)$ is an abstract representation of the composition $c$, namely a workflow composition (with or without binding). $S_1, S_2, ..., S_m$ are the individual services part of the composition. $e_1, e_2, ..., e_m$ are different end points. $QS_1, QS_2, ..., QS_m$ are the QoS of the individual services that are orchestrated. $e_i/S_i$ means that service $S_i$ is bounded to the end point $e_i$.

Given input, function $f'$ will return set of endpoints $e_1, e_2, ..., e_m$ satisfying the goal. In case it cannot find such services, it may return best approximation results.

5.2. Integrate PASO with monitoring tool  A possible direction would be to integrate PASO with a monitoring framework. This would yield two possible benefits

- **Benefit for monitoring framework.** PASO can assist the monitoring framework by predicting in advance, which services are likely to violate the SLA.
- **Benefit for PASO.** Monitoring framework can assist PASO by providing the input probabilities distributions and other QoS information.

Although there are several monitoring frameworks available, but we are considering SLA monitoring framework (SALMon [12]).

5.3. Use proposed algorithm to predict QoS of cloud based application  A possible direction would be to explore how we can define a QoS-oriented analysis of topologies of cloud-based applications (e.g., SeaClouds [13]). The topology of an application specifies various (QoS) properties of the application components as well as the relationships among them. The goal of the analysis would be to generate some predictions on the QoS of an application by analysing the description of its topology. The challenge is to determine what can we deduce from the analysis of such relations, and possibly how to extend those relations to get better results of the analysis.
5.4. Use PASO to support SLA generation Another direction would be to extend PASO to support Service Level Agreement (SLA) generation. Specifying and verifying properties of the SLA of a service composition are two expensive and error-prone activities if performed entirely by hand. The availability of a well-defined, automated support to help performing those activities (in the form of PASO) could considerably reduce their cost and reduce the possible presence of incoherencies in the generated SLAs.

References

Artifact-centric Choreographies: Coordinating Flexible Cross-organizational Processes

Kristof Hamann

Supervisor: Winfried Lamersdorf

Distributed Systems Group, University of Hamburg, Germany

Abstract. In the context of service-oriented architectures, choreographies constitute an agreement between participating organizations in cross-organizational processes. However, prevalent approaches for modeling choreographies use the activity-centric or message-centric paradigm. Hence, they are rather inflexible and cannot cope with increasing demands for runtime adaptation to changing requirements. Therefore, the goal of this work is to develop an end-to-end approach for supporting flexible, cross-organizational processes. The main building block is a choreography meta-model using a more flexible modeling paradigm, i.e., the artifact-centric paradigm. Furthermore, a realization concept has to be developed, which proposes a distributed system-architecture for enacting and monitoring such artifact-centric choreographies.

1 Introduction

Today’s organizations, such as enterprises or administrative bodies, have to be flexible in order to be able to adapt to changing requirements. Especially the rapid advancements of communication technologies have resulted in a highly competitive market. Therefore, flexible business processes have to cope with adaptations, e.g., on the operational, behavioral or organizational level.

At the same time, decreasing costs and increasing quality of communication – i.e., availability and speed of Internet connections – facilitates new shapes of cooperation between organizations. This enables the dynamic integration of varying partners in order to realize complex cross-organizational processes and introduces another dimension of flexibility.

On the technical level, Service-Oriented Architectures (SOA) are the preferred paradigm used for building process-oriented applications [1]. Here, a business process is executed using a dynamic composition of loosely coupled services. In general, two approaches for realizing such service compositions can be distinguished: orchestrations and choreographies.

Orchestrations are directed by a central execution instance and are therefore typically used in intra-organizational processes with internal or external service calls. Respective processes are typically modeled using activity-centric languages
such as BPMN or BPEL. Here, a process model consists of activities, such as service calls, which are structured using control flow elements in order to describe the sequence in which the activities have to be executed. Although alternative control flow paths can be modeled and dynamically selected at runtime, flexibility is limited since it is not possible to model all exceptional behavior and special cases of a real world process in advance.

However, due to increasing needs for flexible processes, several approaches have been proposed in order to provide runtime adaptation for activity-centric processes. Examples are the ADEPT project which targets at control flow modification of process instances [3] or the process instance migration for a flexible distribution of processes at runtime [17].

Due to the complexity of such attempts, other approaches dismiss the activity-centric structure of processes and introduce new modeling paradigms which allow for a more flexible process description. A notable example are artifact-centric processes which focus on business objects: here, an artifact is a modeling construct which combines an information model containing all relevant data of this object with a life cycle model defining artifact states and, state-dependently, the allowed operations on the data [12]. Unlike activity-centric models, the artifact does not dictate the exact order of operations and, therefore, helps to overcome the inherent inflexibility of a strict procedural view in activity-centric processes.

In contrast to orchestrations, a choreography views a service composition from the global perspective without a central execution instance. Its purpose is to describe a multi-party collaboration found in cross-organizational processes. Therefore, choreographies can be seen as an agreement between participating organizations. Hence, it cannot be executed actively, but it have to be enacted by executing the respective orchestrations by the involved participants.

There are two dominating paradigms for modeling choreographies [4]: interconnected interface behavior models (or interconnection models) are activity-centric and connect the participants' public activities, such as sending or receiving messages, in order to create the global view (e.g. BPEL4Chor, BPMN collaboration diagrams). Interaction models are message-centric, i.e., the order of messages exchanged between the participants is modeled from the global perspective (e.g. WS-CDL, Let’s Dance).

Since interconnected interface behavior choreography models are a kind of extension of activity-centric orchestrations, they likewise exhibit the discussed disadvantage, namely inflexibility. Apart from that, interaction models are located on a rather technical level. They deal with messages to be exchanged between cooperating services and therefore are not suited very well for modeling cross-organizational processes on a business level. Furthermore, the flexibility of the message order is still limited by the procedural nature of defining the flow.

2 Statement of the Problem

Since a choreography constitutes an agreement for a cross-organizational process, every participant has to comply with it. As a consequence, due to their inflexibil-
ity, prevalent choreography paradigms hamper the use of flexible orchestrations (cp. Section 1) as part of a cross-organizational process. This is problematic, e.g., if some cross-organizational activities’ order have to be changed while the choreography does not provide a respective mechanism for coordination between the participants. Therefore, this work addresses the problem of utilizing choreographies for implementing flexible cross-organizational processes.

As shown for orchestrations, another modeling paradigm can help to overcome inherent flexibility restrictions. In particular, the artifact-centric paradigm is promising, since its more declarative approach provides an inherent higher level of flexibility than the activity-centric paradigm [11], it is a more natural approach than the message-centric paradigm [16], its focus on business objects fosters comprehensibility and modularity [8], and its applicability for choreographies was already shown theoretically [9, 16, 14].

However, existing approaches in this area exhibit drawbacks which prevent them from being applied for cross-organizational processes in practice. Some publications examine the artifact-centric paradigm from the theoretical perspective only without considering implementation requirements [9, 16, 5, 14]. Other approaches miss an end-to-end realization on artifacts [10]. However, such a realization is desired in order to take advantage of using the same process model for all aspects of a collaboration, i.e., specification and communication, implementation and monitoring. Approaches considering an end-to-end realization of artifact-centric processes, however, propose a centralistic infrastructure which is only suitable for intra-organizational processes [2, 13, 11].

3 Goal of the Work

The goal of the dissertation is to develop concepts in order to provide an end-to-end approach for supporting flexible, cross-organizational processes in the context of service-oriented architectures. End-to-end means here, that the main paradigm – in this work, this will be the artifact-centric paradigm – is used for all steps of a process’ life cycle, i.e., from the modeling over implementation to execution and monitoring.

Therefore, based on existing approaches for artifact-centric processes, a meta-model has to be developed, which is flexible enough to cope with the requirements of cross-organizational processes.

Furthermore, a realization concept has to be developed, which proposes a distributed system-architecture for enacting and monitoring such cross-organizational processes. This practical realization will be used in order to evaluate the approach of artifact-centric choreographies.

4 Meta-Model for Artifact-centric Choreographies

The meta-model is the main building block for cross-organizational processes in this work. It is used as agreement or contract between the participants of a collaboration in order to achieve a common goal. Therefore, its design is directly
influenced by three major requirements for cross-organizational processes [16]:
compliance (all parties must provide the services as they have agreed in the contract), autonomy (each party has the freedom to realize its own part in the collaboration autonomously), and flexibility (each party has the freedom to change its own part in the collaboration). Here, a balance has to be found in order to respect contrary demands: compliance with a clear definition of who is when allowed to do what on the one hand, and flexibility resp. autonomy on the other hand, which takes advantage of a preferably open definition.

Therefore, the general approach in this work is to adopt the artifact-centric paradigm to cross-organizational processes involving multiple participants. The resulting concept of choreography artifacts also introduces the migratability of artifacts (cp. [9]) in order to enable both autonomy and flexibility by allowing participants to act autonomously on an artifact while having the flexibility to transfer the control to another participant.

However, beside the root element choreography, the main element of the meta-model is artifact, which can have multiple attributes which form the information model. The life cycle model of an artifact is based on UML protocol state machines and has to be extended with a role model, which allows for the definition of participants and roles as well as the flexible assignment to artifact’s operations.

Furthermore, non-functional requirements often play an important role in cross-organizational processes. Respective service-level agreements have to deal with, e.g., time limits or quality levels for specific activities or participants. Moreover, the meta-model has to include a suitable error handling for situations, when functional or non-functional agreements cannot be hold. Therefore, it has to consider contractual penalties or compensation subprocesses for these cases.

The possibility to transfer a choreography artifact between participants allow for flexible changes of responsibilities, but introduces also security issues. Therefore, the meta-model has to provide constructs in order to restrict the migration of or the access to (parts of) an artifact using encryption.

5 Distributed System Architecture

Unlike orchestrations, choreographies cannot be executed actively, but they are enacted by executing the respective orchestrations by the involved participants. Therefore, an important challenge is to provide an infrastructure for monitoring the process’ progress and trigger appropriate actions in case of violating the choreography, e.g., replace a participant with a more reliable one and compensate the erroneous work. In general, due to the lack of a central instance and the autonomy of the participants, such an infrastructure requires a distributed architecture.

Figure 1 sketches the proposed architecture, which addresses these challenges. The general idea is to execute an artifact as a mobile process (cp. [17]). With this approach, an artifact can be transferred between participants and each participant can perform operations on it – either locally or remote using the management infrastructure [6].
This approach influences the participating organizations, since they have to integrate artifacts into their process in order to collaborate with the other participants in the cross-organizational process. However, in order to be able to integrate existing, heterogeneous systems, the approach allows for different integration scenarios. The first scenario uses a BPMN extension (e.g., [9]) in order to integrate the artifacts into the (local) orchestration (organization 1). This approach benefits from an end-to-end realization of the artifact paradigm. Nevertheless, traditional orchestration such as BPEL processes can be integrated by using an artifact engine on top of the orchestration engine (organization 2). Here, the orchestration can access the artifact through a web service interface. Furthermore, this infrastructure component can be outsourced to a cloud provider which offers the artifact as a service (organization 3).

On the top, a monitoring and management component supervises the choreography progress using a distributed complex event processing based on monitoring agreements proposed by [15]. Here, the participants are under an obligation to exchange events concerning the choreography. In case a participant’s monitoring and management component detects missing event notifications, a delayed participant or other problematic situations, the observing participant can initiate respective corrective actions, e.g., force a participant to substitute.

6 Summary and Outlook

This work aims at the flexibilization of cross-organizational processes. Therefore, a flexible choreography meta-model based on the artifact-centric paradigm as well as a distributed system architecture are being developed. The approach will be evaluated using an analysis on how exemplary processes can profit from applying an artifact-centric choreography model. It is expected that such a model provides more flexibility than traditional choreography models.
References

Abstract. The popularity of service-oriented software systems is shown by the extended use in industry and the interest of new research results. To ensure the right behaviour of service-oriented software systems a wide range of analyses exist. The aim of this project is to find an appropriate approach to verify existing service-oriented software systems including concurrency, recursion and recursive callbacks, to ensure the right behaviour of the services and the service-oriented software system with respect to termination, e.g., whether each called process implemented by a service reaches a final state. In this paper we are focusing on concurrency and its impact of termination in service-oriented software systems. We present an example that shows the presence of a deadlock which can be hidden during testing and we propose an approach to detect the presence of possible deadlocks by code abstraction.

Keywords: termination analysis; concurrency; deadlock.

1 Introduction

The correct behaviour of services and of service-oriented software systems is important to ensure the expected functionality. Within program analysis the termination analysis is an important research area of computer science. If a service is not terminating, the right behaviour of the service-oriented software system cannot be guaranteed. The consideration of termination of service-oriented software systems, including concurrency, seems important, because the termination often depends on the unpredictable execution of the threads or the processes.

However, deadlocks and livelocks caused by threads or processes contained in a service are almost impossible to detect during testing although they can occur in a running system. Termination analysis including concurrency is done by [1], [2]. Termination analysis tools for service-oriented software systems, to the best of our knowledge, does not exist.

To fill this gap, existing methods for termination analysis are used to examine service-oriented software systems towards termination. Recursion, concurrency,
Mandy Weißbach

recursion associated with internal and external concurrency, recursion including recursive callbacks, synchronous and asynchronous messaging will be considered in this project. Our goal is to integrate the information to state termination and deadlock-freeness into the services interface description to keep the blackbox view of services. Our termination analysis is conservative and should work fully automatically.

In this work the goals, the existing problems and the main ideas are shown in Section 2 followed by the discussion of the results in Section 3. The related work can be found in Section 4. At the end, a conclusion and a short outlook is given in Section 5.

2 Problem Statement and Example

In our work we are focusing on termination analysis of service-oriented software systems. We want to consider occurring dead- and livelocks that arise within service-oriented software systems, if concurrency and recursion with no restrictions (e.g., depth of recursion and degree of concurrency) is present. We are also considering recursive callbacks and synchronously and asynchronously service communication [3]. The following points need to be considered:

1. determining areas of improvement by means of example [3],
2. termination analysis (TA) of service-oriented systems with no asynchronous calls and no recursive callbacks [4],
3. TA of service-oriented systems with internal recursion [4],
4. TA of service-oriented systems with recursive callbacks [5],
5. TA of service-oriented systems with only concurrency including synchronous and asynchronous calls (this paper),
6. TA of service-oriented systems including 3 and 4 and
7. implementation of an automatic verification tool.

Literature research has shown that these considerations towards termination analysis of service-oriented software system are missing. A discussion about related work can be found in section 4.

In [3] we focused on Petri-Net based termination tools that assume, under the fairness assumption, loop termination. This can lead if a non-terminating loop exist to a livelock situation and if the service is called synchronously by another service it causes a deadlock of the service-oriented software system. Furthermore, we showed in [5] and [4] how to handle recursion and recursive callbacks. We used size change function to prove termination of recursion and to prove the termination of recursive callbacks we used recurrences which can be computed and solved automatically. Both approaches are conservative which means if termination can be proven, then the service-oriented system indeed terminates. Thus, with a conservative abstraction false positives can not occur (termination is predicted but the service-oriented software system does not terminate). However, false negatives, can appear (termination is not predicted but the service-oriented systems terminates).
The next step is to consider concurrency within a service and concurrency between services in a service-oriented software system. For that reason, we want to introduce the example code of Listing 1.1. It is written in Java and produces under certain circumstances which can not be predicted into a deadlock. However, the Deadlock is not detected by testing although it can occur in a running system.

Listing 1.1. Deadlock.java

```java
public class Deadlock {
    static class A {
        private final String name;
        public A(String name) {
            this.name = name;
        }
        public String getName() {
            return this.name;
        }
        public synchronized void x(A obj) {
            System.out.format("Object %s : Object %s \n", this.name, obj.getName());
            obj.y(this);
        }
        public synchronized void y(A obj) {
            // do something
        }
    }

    public static void main(String[] args) {
        final A object1 = new A("no_1");
        final A object2 = new A("no_2");

        // thread 1
        new Thread(new Runnable() {
            public void run() { object1.x(object2); }
        }).start();

        // thread 2
        new Thread(new Runnable() {
            public void run() { object2.x(object1); }
        }).start();
    }
}
```

The code is based on the example code at the Oracle Site [6]. In the main method two objects of A will be created and two threads will be started. Let thread 1 be the first thread that is executed. Thread 1 will call method x on object 1, because the method x is synchronized the object 1 will be blocked. Before thread 1 calls method y thread 2 calls method x on object 2 and object 2 is now blocked. Because object 1 and object 2 are now blocked neither of the two threads can call method y and the program deadlocks. Sometimes it happens that thread 1 is executed completely before thread 2 is started. However, if each thread is executed at one time the program terminates (this happens during debugging).

We also want to discuss what happens to a service-oriented software system if an deadlock occurs. The deadlock can occur in a service, an internal deadlock. A service can be called synchronously (the caller waits until the message
is passed to the callee and it gets an response) or asynchronously (the caller
does not wait until the message is passed to the callee). If the deadlock occurs
in the synchronously called service a typical communication deadlock occurs.
The system changes into a livelock, compare with [3]. If the deadlock occurs in
a asynchronously called service, the behaviour of the service-oriented software
system will be undefined.

Also, deadlock can occur between services in a service-oriented software sys-
tem, we call them external deadlocks. In figure 1 a service-oriented software
system with three different services is shown. Service A calls service B and ser-
vice C concurrently (at the same time). We assume that we have single instance
services and every service is locked for the called service. Once a service re-
sponded to the called service, the lock is dropped. In figure 1 component B tries
to call component C but C is locked by service A. At the same time service C
tries to call service B but service B is also locked by service A, hence a deadlock
occurs. This situation could happen with services that provide database access.

To detect internal deadlocks and external deadlocks we propose two differ-
ent approaches. The approach of the internal deadlock detection can be done as
follows:

**Step 1:** do code analysis to find all classes implementing synchronized methods
(synchronized classes)
**Step 2:** for every thread do code analysis to find objects of the synchronized
class if an object is found store the location for that synchronized class
**Step 3:** if more than one location is stored for a synchronized class a could
deadlock occurs

If more than one location is stored for a synchronized class a deadlock might be
possible. A deadlock will never occur if there is only one or none location stored
for a synchronized class.

To prove the absence of an external deadlock we refer to our work in [7]. For
every service the required functions that are called by calling a provided function
of the service need to be computed. Another service besides the Registry could compute the possible call graph of the services dynamically. This call graph has to be acyclic to guarantee deadlock absence.

3 Results

The presented approach to find internal deadlocks is abstraction based, while many other research in that area is refinement based.

We do a code analysis that does an over-approximation to detect all path where a deadlock could occur. With this kind of approach we will have false negatives, we detect possible deadlock situation which cannot be present during runtime and the service could terminate. The number of false negatives need to be reduced in further considerations. False negatives can occur because we consider every thread within a program as it is running at the same time. If we know that a thread is stopped before other threads are started the amount of false negatives could be reduced. False positives cannot be occur because it is an conservative approach.

Remark (False Positives, False Negatives). The terms of false negatives and false positives are chosen regarding the problem: does the Program terminate.

An approach to state absence of external deadlocks in service-oriented software systems is also shown. We propose to use a static call graph to compute the required function by invoking a provided function of a service. This can be done before publishing the service. The information could be stored in the WSDL interface description and with the help of this information the call graph of the called services can be computed. If the call graph is acyclic, termination cannot be guaranteed.

4 Related work

In this project we are focusing on termination analysis in service-oriented software systems. Termination analysis has its origin in 1967. Floyd published his work [8] concerning partial and total correctness. Until now, a lot of research was done towards termination analysis of programs. A good summary for proving program termination is given in [9]. Termination analysis for Java programs including recursion is done by [10]. To our knowledge termination analysis of service-oriented software systems including internal and external deadlocks does not exist. The proof of termination in combination with the occurrence of recursion, recursive callbacks, concurrency in service-oriented software system are not investigated by other researcher, except [3].

In [12] an abstract behavioural specification language is used to model distributed concurrent systems. COSTABS a COST and Termination analyzer, based on COSta [10], is presented, which is able to prove termination. In contrast to our work they focus on loop termination. The above work [12] is extended in [11].
Other approaches that focus on termination analysis is van der Aalst et al. [13, p. 36(58)]. They consider concurrency and the absence of deadlocks. The Petri models are refined into business processes (refinement based approaches). Petri nets and their analysis tools requires the fairness assumption, e.g., every loop (recursion) terminates.

5 Conclusion

In this paper we focused on termination analysis in services and service-oriented software systems. We showed an example which illustrates the occurrence of a deadlock in a Java program and we showed what it implies if a deadlock occurs in a service or in a service-oriented software system. We proposed for the prediction of deadlocks two approaches. To predict deadlocks in services, internal deadlocks, we suggest using an abstraction mechanism using well known compiler methods.

For external deadlock detection in service-oriented software systems, we propose to provide a certain kind of protocol which points out the services that need to be called by the called service. The protocol could be provided over the WSDL interface description.

Currently we are concerning the implementation of a termination tool for service-oriented software systems including our previous research results. Another aspect that need to be considered is the reduction of false negatives during deadlock detection of services.

Acknowledgment

The authors would like to thank the anonymous reviewers for their helpful comments.

References

6. Oracle.
   https://docs.oracle.com/javase/tutorial/essential/concurrency/
   last accessed August 2015.
7. Weißbach, M., Zimmermann, W.: Controlling data-flow in the cloud. CLOUD
   COMPUTING 2012: The Third International Conference on Cloud Computing,
   GRIDs, and Virtualization (2012) 24–29
   science 19(19-32) (1967) 1
    implementation of a cost and termination analyzer for java bytecode. In de Boer, F.,
    and Objects. Volume 5382 of Lecture Notes in Computer Science. Springer Berlin
    concurrent objects: Theory and practice. In Johnsen, E., Petre, L., eds.: Integrated
    Formal Methods. Volume 7940 of Lecture Notes in Computer Science. Springer
    Berlin Heidelberg (2013) 394–411
    A cost and termination analyzer for abs. In: Proceedings of the ACM SIGPLAN
    2012 Workshop on Partial Evaluation and Program Manipulation. PEPM ’12, New
    York, NY, USA, ACM (2012) 151–154
13. Van der Aalst, W.M.: The application of petri nets to workflow management.
Information about the Author(s)

Gianluigi Zavattaro (Editor)
Dipartimento di Informatica
Università de Bologna
40127 Bologna, Italy
E-Mail: gianluigi.zavattaro@unibo.it
WWW: https://www.unibo.it/sitoweb/gianluigi.zavattaro

Wolf Zimmermann (Editor)
Institut für Informatik
Universität Halle
06099 Halle, Germany
E-Mail: wolf.zimmermann@informatik.uni-halle.de
WWW: http://swt.informatik.uni-halle.de/mitarbeiter/zimmermann/