Contents

Deadlock Analysis of Service-Oriented Systems with Recursion and Concurrency
*Mandy Weißbach, Supervisor: Wolf Zimmermann* 5

Predicting Quality of Service of Software Applications
*Ahmad Ibrahim, Supervisor: Antonio Brogi* 16

Ensuring quality-critical digital services in and for dynamically changing environment
*Anne Immonen, Supervisors: Eila Ovaska, Veikko Seppänen* 24

Impact-minimizing Runtime Adaptation in Cloud-based Data Stream Processing
*Cui Qin, Supervisors: Klaus Schmid, Holger Eichelberger* 32

Towards Unified Management of Applications on Heterogeneous Clouds
*Jose Carrasco, Supervisors: Francisco Duran, Ernesto Pimentel* 40
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 five contributions. Each submission was reviewed by at least two 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 Schahram Dustdar and his team for his organizatorial support. We thank the international programme committee consisting of

- Antonio Brogi, University of Pisa, Italy
- Friederike Klan, Friedrich Schiller University Jena, Germany
- Welf Löwe, Linnaeus University, Sweden
- Flavio de Paoli, University of Milano-Bicocca, Italy
- Alexander Pokahr, University of Hamburg, Germany
- Ernesto Pimentel, University of Malaga, Spain
- Emilio Tuosto, University of Leicester, UK
- Massimo Villari, University of Messina, Italy
- John Erik Wittern, IBM T.J. Watson Research Center, USA

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 1, 2016
Halle, September 1, 2016
Deadlock Analysis of Service-Oriented Systems with Recursion and Concurrency

Mandy Weißbach
Supervisor: Wolf Zimmermann

Martin Luther University Halle-Wittenberg, Institute of Computer Science,
Von-Seckendorff-Platz 1, 06120 Halle, Germany
mandy.weissbach@informatik.uni-halle.de

Abstract. In this paper, we show an abstraction-based approach towards analysis of Service-Oriented Systems with the help of Process Rewrite Systems. On the one hand the approach takes into account recursive procedures, i.e., internal recursion in service implementations as well as external recursion over service boundaries. On the other hand, also internal concurrency and concurrency over service boundaries are considered. The abstraction can automatically derived from the service implementations.

Keywords: Process Rewrite Systems, Deadlock, Recursion, Concurrency

1 Introduction and Motivation

Composition of services can be an error-prone task. In particular when for example web service descriptions only provide interface signatures such as WSDL-specifications. The lack of information might lead to unintended behaviour of services such as deadlocks, livelocks [26], unexpected abortions, unexpected functionalities. As interface signature do not provide enough information on the use of services, approaches such as semantic web technologies, tools for checking compatibility criterion etc. are used to reduce the risk of unintended behaviour due to composition incompatibilities [18], [3], [20].

The focus of this work is on deadlock analysis. Prominent approaches such as van der Aalst’s workflow nets [22] are Petri-Net based and use Petri-Net tools to analyse deadlocks [23]. Other approaches are based on process-algebras [15] and use tools from this field to analyse deadlocks [19]. These approaches are usually refinement-based, i.e., the behaviour of a service is defined as a workflow net or process-algebraic expression and then refined to the service implementation. The behaviours are composed corresponding to the architecture of the service-oriented system and then e.g., checked for absence of deadlocks. In contrast to these works, we use an abstraction-based approach, i.e., the behaviour is abstracted from the service implementations. The motivation for an abstraction-based approach is that there are many services not developed according to a
PRS-Rule refinement-based approach. Furthermore, even if they have been developed initially by a refinement-based approach, it is unlikely that programmers consistently maintain the implementation and its abstraction.

For abstraction-based approaches, it is necessary to automatically derive the abstract behaviour of a service from its implementation. Therefore, this abstraction is a classical translation task which can be implemented using compiler technology. An abstraction-based approach should deal with all kinds of source programs. Following this philosophy, concepts like procedure calls, forking asynchronous procedure calls, synchronization and exception handling have to be considered, c.f. Tab. 2. However, Petri-Nets allow only rather imprecise abstractions of procedure calls. The reason is that the behaviour of recursive procedures corresponds to the LIFO principle and requires therefore a stack. A solution for a more precise abstraction might be recursive Petri-Nets [11] or Mayr’s process rewrite systems [17]. Recursive Petri Nets combine properties of Petri Nets and context free grammars. In [9] it is shown that recursive Petri Net languages strictly include the union of Petri Net and Context Free languages. The Process Rewrite System (PRS) are an extension of Petri-Nets by stacks [16]. Therefore, for modelling recursive procedures, forking asynchronous procedure calls, and synchronizations, Process Rewrite Systems and recursive Petri Nets are sufficient. Indeed, it was shown in [10] that PANs (a subset of the more expressive PRSs) are a subset of recursive Petri-Nets, but is unknown whether they are equal.

It was also shown [17] that it is decidable to check PRSs for the absence of deadlocks (deadlock reachability problem). We are not aware of any work towards decidability of deadlock detection in recursive Petri Net. We therefore use PRSs as the basic model for specifying the abstract behaviour.

This paper is organized as follows: In Section 2 we introduce PRSs according to [16], service-oriented systems and we give a short overview of the abstraction process and discuss the deadlock issue. In Section 2.4 we present the problem by means of an example of a service-oriented system including procedure calls, forking asynchronous procedure calls, loop and conditionals. We show how a service-oriented system can be described with Process Rewrite Systems. Section 3 discusses related work and Section 5 concludes our work with a short overview of the gained results and gives a short outlook.

2 Foundation

2.1 Services and Service-Oriented Systems

We assume that a service \( S \) is an implementation with a provided interfaces \( I_s \), where an interface is a set of procedure signatures. It is possible that a service calls procedures/functions of other services. The required interface \( R_s \) of \( S \) is the set of procedures of other services called by \( S \), c.f. Figure 1.a. A service-oriented system is composed by two or more services which communicate only over a required (and provided) interface, c.f. Figure 1.b. Procedures of a interface can
be either called synchronous or asynchronous. Called synchronous procedures block their caller until the callee has completed the call, while asynchronous procedure calls are executed in parallel when they are called. Hence the caller can proceed without waiting for completion.

2.2 Process Rewrite System

Mayr presented a unified view of Petri Nets and several simple process algebras by representing them as subclasses of the general rewriting formalism Process Rewrite Systems [17].

It is based on rewrite rules on process-algebraic expressions. The set \( PEX(Q) \) of process-algebraic expressions over a finite set \( Q \) (atomic processes) is the smallest set satisfying:

(i) \( Q \subseteq PEX(Q) \)

(ii) If \( e, e' \in PEX(Q) \), then \( e.e' \in PEX(Q) \) and \( e \parallel e' \in PEX(Q) \) (sequential and parallel composition, respectively).

(iii) \( \varepsilon \in PEX(Q) \)

The empty process, denoted by \( \varepsilon \), is the identity w.r.t. sequential and parallel composition.

**Definition 1 (Process Rewrite Systems).** A Process Rewrite System (short: PRS) is a tuple \( \Pi \triangleq (\Sigma, Q, \rightarrow, q_0, F) \) where

(i) \( Q \) is a finite set (atomic processes),

(ii) \( \Sigma \) is a finite alphabet disjoint from \( Q \) (actions),

(iii) \( q_0 \in Q \) (the initial state),

(iv) \( \rightarrow \subseteq PEX(Q) \times \Sigma \times PEX(Q) \) is a set of process-rewrite rules,

(v) \( F \subseteq Q \cup \{\varepsilon\} \) (the set of final processes).

The PRS \( \Pi \) defines a derivation relation \( \Rightarrow \subseteq PEX(Q) \times \Sigma^* \times PEX(Q) \) (\( \Sigma^* \) is the set of all finite words over \( \Sigma \)) by the inference rules in Fig. 2.
Remark 1. For this paper, we use always $F \triangleq \{ \varepsilon \}$

Definition 2 (Normal Form). A process algebraic expression $\pi \in PEX(Q)$ of a Process Rewrite System $\Pi$ is a normal form iff there exists no process algebraic expression $\pi' \in PEX(Q)$ with $\pi \Rightarrow \pi'$.

For this work, the left-hand and the right-hand side of a process rewrite rule contains an atomic, parallel or sequential process. So both sides are process algebraic expressions without any restriction. If the right- and left-hand side only allows parallel processes, then there is a one-to-one correspondence to Petri-Nets (Place/Transition Nets) [16]. Table 1 sketches this correspondence. A parallel operator on the left-hand side of a process-rewrite rule corresponds to a synchronization, a parallel operator on the right-hand side of a process-rewrite rule to a fork of parallel processes (e.g. asynchronous service call), a sequential operator on the right-hand side corresponds to a procedure call or synchronous (blocking) service call, and a procedure return corresponds to a process-rewrite rule $q \rightarrow \varepsilon$ [13]. This work shows a one-to-one correspondence between general process-algebraic expressions and cactus stacks, i.e., an abstract program semantics can be viewed as transitions on cactus stacks. This is a well-known semantics of programs with parallel processes and were first introduced by Dahl and Nygaard for the runtime system of Simula (as tree of stacks) [5].

<table>
<thead>
<tr>
<th>Synchronization</th>
<th>Fork</th>
</tr>
</thead>
<tbody>
<tr>
<td>Petri-Net</td>
<td>PRS-rule</td>
</tr>
<tr>
<td>$q_1 \rightarrow q_2$</td>
<td>$q_3$</td>
</tr>
<tr>
<td>$q_1 \parallel q_2$</td>
<td>$q_3$</td>
</tr>
<tr>
<td>$q_1 \parallel k$</td>
<td>$q_2$</td>
</tr>
<tr>
<td>$q_1 \parallel q_2$</td>
<td>$q_3$</td>
</tr>
<tr>
<td>$q_1 \rightarrow q_2 \parallel \varepsilon$</td>
<td>$q_2$</td>
</tr>
</tbody>
</table>

Table 1: Correspondence between Petri-Nets and PRS (only parallel processes on LHS and RHS of Rewrite Rules)
2.3 Deadlock Detection

A deadlock for Petri-Nets corresponds to a normal form (different from \(\varepsilon\)), i.e., a process-algebraic expression where no process-rewrite rule is applicable:

\[
\begin{align*}
& s_0 \xrightarrow{t_0} s_1\|s_1 \\
& s_1 \xrightarrow{t_1} s_2 \\
& s_1 \xrightarrow{t_2} s_3 \\
& s_1\|s_2 \xrightarrow{t_3} s_1\|s_1 \\
& s_1\|s_3 \xrightarrow{t_4} s_1\|s_1
\end{align*}
\]

Fig. 3: A Petri-Net and its Corresponding PRS

Example 1. Fig. 3 shows a Petri-Net that may lead to deadlock. This happens if the transitions fire in the order \(t_0, t_1, t_2\) (leaving one token in place \(s_2\) and one in place \(s_3\)). On the PRS, this corresponds to the derivation

\[
s_0 \xrightarrow{t_0} s_1\|s_1 \xrightarrow{t_1} s_2\|s_1 \xrightarrow{t_3} s_2\|s_3
\]

Now, no rule is applicable, i.e., the expression \(s_2\|s_3\) is a normal form. Note that there is an infinite firing sequence: \(t_0, t_1, t_3, t_1, t_3, \ldots\). This corresponds to the derivation

\[
s_0 \xrightarrow{t_0} s_1\|s_1 \xrightarrow{t_1} s_2\|s_1 \xrightarrow{t_3} s_1\|s_1 \xrightarrow{t_3} s_1\|s_3 \xrightarrow{t_3} s_1\|s_1 \cdots
\]

According to the above discussions, this class of PRS is a generalization of Petri-Nets as it allows on the right-hand side the sequential operator which corresponds to procedure calls including (mutually) recursive procedures. Let \(\Pi \equiv (\Sigma, Q, \rightarrow, q_0, \{\varepsilon\})\) a process-rewrite system and \(\pi \in \text{PEX}(Q)\) reachable from \(q_0\), i.e., \(q_0 \xrightarrow{\pi} \pi\). \(\pi\) is a deadlock for \(\Pi\) if \(\pi \neq \varepsilon\) and \(\pi\) is in normal form. [17] shows that deadlock detection is decidable for this class of PRS. Hence, it is possible to use PRSs for deadlock detection on services if recursion without restriction on the recursion depth and parallelism without restricting the degree of parallelism is allowed.

2.4 Abstraction

This Section shows how to abstract the behaviour of services to process rewrite systems.

The service oriented system in Figure 4 includes a synchronous and asynchronous (forking) procedure call, a synchronization and recursion. The used states correspond to the program points of the implementation of the services.

Table 2 shows how the different control structures of our example can be abstracted. By applying these rules to the service oriented system in Figure 4 the shown process rewrite rules \(\rightarrow\) in Table 3 can be abstracted. A derivation from \(q_0\) is shown in Table 2.
Assignments have no influence on deadlock behaviour. \( q' \) as the program point of the statement being executed after \( n := 1 \);

No influence on deadlock behaviour if the condition is being decided. \( q'' \) is the program point of the statement being executed after the last statement of the then- and else-part.

Call of a synchronous procedure \( p \): The program point \( q' \) of the statement to be executed after the call is pushed onto the stack. The execution continues with first program point \( q'' \) of \( p \).

Call of an asynchronous procedure \( p \): The execution can be continued concurrently with the statement at program point \( q'' \) after the call and the statement at the first program point \( q'' \) and \( q' \).

The statement after \( q \) (at program point \( q' \)) can only be executed when the previously called asynchronously procedure \( p \) returns.

<table>
<thead>
<tr>
<th>Control Structure</th>
<th>Abstraction</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q : n := 1; q' )</td>
<td>( q \rightarrow q' )</td>
<td>Assignments have no influence on deadlock behaviour. ( q' ) as the program point of the statement being executed after ( n := 1 );</td>
</tr>
<tr>
<td>( q : \text{if}(e)q' : \ldots \text{else}q'' : \ldots )</td>
<td>( q \rightarrow q' )</td>
<td>No influence on deadlock behaviour if the condition is being decided. ( q'' ) is the program point of the statement being executed after the last statement of the then- and else-part.</td>
</tr>
<tr>
<td>( q : p(\ldots); q' : \ldots )</td>
<td>( q \rightarrow q', q'' )</td>
<td>Call of a synchronous procedure ( p ): The program point ( q' ) of the statement to be executed after the call is pushed onto the stack. The execution continues with first program point ( q'' ) of ( p ).</td>
</tr>
<tr>
<td>( q : \text{async} p; q : \ldots )</td>
<td>( q</td>
<td></td>
</tr>
<tr>
<td>( q : \text{sync} p; q : \ldots )</td>
<td>( q</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Abstraction of Control Structures to Process Rewrite Rules [12]

Fig. 4: A Service-Oriented System with three services. Service S acts as a client.
By applying the rules 1, 3, 13, 6, 17, 19, 8, 14, 8 we maintain the process algebraic expression \( \pi = (q_3\|q_8\|q_{15}\|q_5\|q_8\|q_{12})\). It represents a normal form because no rules can be applied. So, the service oriented system reaches a deadlock when entering state \( \pi \).

### Abstracted PRS-rules

1. \( q_0 \rightarrow q_2\cdot q_1 \)
2. \( q_1 \rightarrow \epsilon \) (3) \( q_2 \rightarrow q_3 \| q_{10} \) (4) \( q_3 \| q_{12} \rightarrow q_3 \) (5) \( q_3 \cdot q_1 \rightarrow q_6 \) (6) \( q_4 \rightarrow q_{13} \cdot q_5 \)

### Table 3: Abstraction of the Example of Fig. 4 to Process Rewrite Rules \( \rightarrow \).

**Remark 2.** The abstraction is similar to control-flow graphs as used for program analysis [2]. Each standard statement has a unique entry point but possibly multiple exit points (e.g., break and continue-statements). For simplicity, Table 2 only considers statements with a single exit point. Multiple Exit points are future work. Decisions in conditionals and loops could be abstracted to non-deterministic choices and belongs to future work. For asynchronous procedure calls, the place after the call is considered as the unique exit point.

<table>
<thead>
<tr>
<th>Derivation of PRS-rules</th>
<th>Rule</th>
<th>Represents Method call</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_6 \Rightarrow q_2\cdot q_1 )</td>
<td>(1)</td>
<td>( p(); )</td>
</tr>
<tr>
<td>( q_2\cdot q_1 \Rightarrow (q_3|q_{10})\cdot q_1 )</td>
<td>(3)</td>
<td>( a(); )</td>
</tr>
<tr>
<td>( (q_3|q_{10})\cdot q_1 \Rightarrow (q_3|q_4|q_{11})\cdot q_1 )</td>
<td>(13)</td>
<td>( b(); )</td>
</tr>
<tr>
<td>( (q_3|q_{12})\cdot q_1 \Rightarrow (q_3|q_{14}\cdot q_5|q_{11})\cdot q_1 )</td>
<td>(6)</td>
<td>( r(); )</td>
</tr>
<tr>
<td>( (q_3|q_{14}\cdot q_5|q_{11})\cdot q_1 \Rightarrow (q_3|q_8\cdot q_5\cdot q_2|q_{11})\cdot q_1 )</td>
<td>(17)</td>
<td>( i(f(...)) )</td>
</tr>
<tr>
<td>( (q_3|q_8\cdot q_5\cdot q_2|q_{11})\cdot q_1 \Rightarrow (q_3|q_8\cdot q_5\cdot q_2|q_{11})\cdot q_1 )</td>
<td>(19)</td>
<td>( q(); )</td>
</tr>
<tr>
<td>( (q_3|q_8\cdot q_5\cdot q_2|q_{11})\cdot q_1 \Rightarrow (q_3|q_8\cdot q_5\cdot q_2|q_{11})\cdot q_1 )</td>
<td>(8)</td>
<td>( i(f(...)) )</td>
</tr>
<tr>
<td>( (q_3|q_8\cdot q_5\cdot q_2|q_{11})\cdot q_1 \Rightarrow (q_3|q_8\cdot q_5\cdot q_2|q_{11})\cdot q_1 )</td>
<td>(14)</td>
<td>( q(); )</td>
</tr>
<tr>
<td>( (q_3|q_8\cdot q_5\cdot q_2|q_{11})\cdot q_1 \Rightarrow (q_3|q_8\cdot q_5\cdot q_2|q_{11})\cdot q_1 )</td>
<td>(8)</td>
<td>( i(f(...)) )</td>
</tr>
<tr>
<td>( (q_3|q_8\cdot q_5\cdot q_2|q_{11})\cdot q_1 \Rightarrow )</td>
<td>( \rightarrow )</td>
<td>normal form</td>
</tr>
</tbody>
</table>

### Table 4: Derivation of Example of Fig. 4 with the Process Rewrite Rules \( \rightarrow \) from Table 3.

**Remark 3.** The abstraction is done for every service, following the rule of Table 2. Every service abstraction will be combined to a system abstraction, shown in 3.

### 3 Related Work

In [21] recursive Petri Nets (rPN) are used to model the planning of autonomous agents which transport goods form location A to location B and their coordinating problem. The model of recursive Petri Nets is used to model dynamic processes (e.g., agent’s request). Recursion in our sense is not considered. Deadlocks can only arise when interactions between agents (e.g., shared attributes) invalidates preconditions. For that reason a coordinating algorithm is introduced to prevent deadlocks (interactions between agents).
Another refinement based approach is described in [14]. This approach models healthcare processes and its called sub-processes with recursive Petri Nets.

Recursion in our sense is not considered. Both approaches use the ability of rPNs to prune subtrees from the state. In [14] and [10] recursion is used to allow the flexible extension of a certain workflow, e.g. health care workflow with a flexible healthcare process, called sub-processes.

[3] uses also Process Rewrite Systems to check the right behaviour of service-oriented Systems. Instead of considering deadlocks to ensure the right behaviour they check protocol conformance. Here, too, recursion and concurrency (internal and over service boundaries) of the services in a service-oriented Systems are allowed. Protocols ensure the right call sequence of all callable operations defined in an interface of a single service or instance. Like ours, this approach is abstraction based.

[1] uses workflow-nets to model business processes. This approach is based on Petri Nets. Hence only concurrency can be considered.

An abstraction based approach is done by Bouajjani et al. [4]. They discuss the analysis of recursive parallel programs based on recursive vector addition systems. They also explore the decidability of problems such as reachability. It seems that their model is slightly more general as there are situations where the reachability problem becomes undecidable. Neither deadlock analysis in services nor in composed systems is considered. Another abstraction based approaches are done by [7] and [6]. Both approaches are based on lam programs, a JAVA-like language introduced in [8]. [6] uses a points-to analysis and a may-happen-in-parallel-analysis to build up a dependency graph. The absence of cycles in the graph ensures deadlock freeness. The challenging example is how to handle object and task creation in loops.

The approach of [7] is based on a type system which is associated to the processes (or networks with nodes) of the lam program. The authors show that for a subset of lam programs including recursion of a non linear problem, deadlocks-freeness is decidable. E.g., the recurrence relation of Fibonacci is a non linear and the recurrence relation of the factorial function is a linear problem.

Neither deadlock analysis in services nor in composed systems is considered. So deadlocks that appear by recursive callbacks can possibly not being detected. This is subject for further research.

To our knowledge, abstraction-based deadlock analysis in service-oriented systems including synchronous and asynchronous procedure calls (forking), recursion and recursive callbacks was not investigated before.

4 Research Plan

We want to examine in detail whether all possible processes within a service come to an end or all called processes in a composition of services, a service-oriented system, come to an end. Services are allowed to communicate asynchronously and synchronously. Also, recursion and recursive callbacks are allowed. Furthermore,
our goal was to state an approach that supports the black-box behaviour of services to keep the business secret.

However, there are two reasons why a service or a service-oriented system can not come to an end: the service or service-oriented system can not terminate (recursion/sequential case) or deadlocks (concurrency).

So we used termination and deadlock analysis to decide if the called processes of or a service-oriented system comes to an end or not.

In summary the following points were or will be considered:

1. determination of areas of improvement towards termination analysis of service-oriented systems [26]
2. termination analysis of service-oriented systems with no asynchronous calls and with no recursive callbacks [26],[25]
3. termination analysis of service-oriented systems with recursive callbacks [27]
4. deadlock analysis of service-oriented systems with only concurrency including asynchronous calls [24], partly this paper.
5. termination and deadlock analysis of service-oriented systems including 3 and 4 (this paper).
6. implementation of an automatic verification tool and case study.

Literature research has shown that the common consideration of recursion and concurrency in termination and deadlock analysis in service-oriented systems is missing. In [26] we pointed out that deadlock analysis based on Petri Nets only abstract the concurrent behaviour. Recursion is ignored and fairness and soundness of the abstracted Petri Net is assumed. But under certain circumstances this can lead to a deadlock [26]. We proposed the introduction of a termination and size change function which can be provided over the WSDL-description of every service [26].

In [27] recurrence equations were introduced to handle recursion and recursive callbacks. With the help of a closed system of recurrences the termination can be proved.

There are still open points that need to be considered. For example under which circumstances, recursion can be ignored by doing the deadlock analysis.

5 Conclusions

An overview of the research topic deadlock analysis in service-oriented systems was given. We allow concurrency and recursion, internal and over service boundaries. Web Services can be described using Process Rewrite Systems. Process Rewrite Systems can model concurrency and recursion. The PRS description can be composed to describe the service behaviour and therefore enables the application of the deadlock detection algorithm for PRSs.

Recursive callbacks combined with concurrency, further implementation of a fully automatically tool to test the practical suitability and applicability are still open points to be investigated. Since we have an abstraction based approach, it seems to be interesting if the verification can be done by only providing the abstractions of the service to keep the business secret and the black-box view.
References

18. Parizék, P., Plasil, F.: Objects, Components, Models and Patterns: 46th International Conference, TOOLS EUROPE 2008, Zurich, Switzerland, June 30 - July


Predicting Quality of Service of Software Applications

Ahmad Ibrahim
Supervisor: Antonio Brogi

Department of Computer Science, University of Pisa, Italy
ahmad@di.unipi.it

1 Introduction

Quality of Service (QoS) refers to a set of non-functional attributes used to describe a system such as response time, reliability or cost.

The QoS of an application does depend on the QoS of the services it uses. For instance, for service based applications, the response time of a service orchestration obviously depends on the response times of the services it invokes - which may actually vary over time for different reasons (e.g., workload or network congestion).

A straightforward way to assess the QoS of an application would be to simply deploy it and then monitor the QoS parameters of interest over a sufficiently high number of runs. Unfortunately, such an approach can be time consuming, expensive (when non-free services are invoked) and may not be feasible when invocations have side-effects (e.g., as in the case of services enabling monetary transactions).

In this research we consider two classes of applications

1. Service based applications. Service based application are usually defined via workflows that implement business processes by orchestrating various (possibly third-party) services.
2. Parallel design patterns based applications. Parallel design patterns (also called skeletons) are customisable, composable design patterns that can be fruitfully exploited to define parallel applications.

The ability of a priori predicting the QoS of an application is very valuable for application designers. In service based applications, predicting the QoS of a service orchestration is valuable both during the design of the orchestration and for defining its Service Level Agreement (SLA). It helps answering questions, like:

– What is the QoS of a given orchestration?
– What is the effect on the QoS of an orchestration of replacing one or more of the invoked services with alternative services, (e.g., offered by different providers)?
– How modifying the workflow of an orchestration impacts on its overall QoS?
Similarly, for parallel design pattern based applications, QoS prediction is valuable for comparing different deployments as well as for assessing the scalability of their parallelization. It helps answering questions, like:

- What is the QoS of a given parallel design pattern based application?
- Would a restructuring improve the QoS of a parallel design pattern based application for a steady state?
- What type of parallelization would achieve the best QoS for a dynamic state?

**Challenges in predicting the QoS of an application**

A priori predicting the QoS of an application is not easy mainly because of four main challenges.

1. **Different possible results of service invocations.** In service based applications, each invoked service can return a successful reply, a fault notification, or even no reply at all [7]. If a fault is returned, a fault handling routine will be executed instead of the normal control flow. If no reply is received, the orchestrator will get stuck (unless some parallel branch throws a fault). In either case, the resulting QoS of the orchestration will differ from the case of successful invocation.

In the case of parallel design patterns based applications, the QoS of the activities executed depends for instance on the type of the inputs arriving in the input stream, as different inputs typically require different service times.

2. **Non-determinism in the orchestration workflow.** The control flow of a workflow defining a service based applications is in general nondeterministic. Besides the nondeterminism induced by the possible different results of service invocations, a workflow usually contains branching conditions that depend upon input data values. As a consequence, which branch will be executed in an alternative, or the number of iterations of a loop is not known a priori [8,9].

In the case of parallel design patterns based applications, the execution of conditional loops defined by feedback patterns has a similar nondeterministic behaviour, as some outputs may be routed back depending upon the evaluation of conditions, and the number of iterations is not known a priori.

3. **Complex dependencies among workflow activities.** Workflows can contain complex dependencies among activities, as for instance those defined by WS-BPEL synchronization links [10]. The control flow defined by synchronizations within parallel activities is more expressive than what is allowed by simple parallel execution with synchronization barriers at the end. This implies that workflows containing such complex synchronization structures cannot be always decomposed into parallel and sequential compositions, as shown in [11,12].

In the case of parallel design patterns based applications, the execution of a pipe pattern for instance exhibits a complex dependencies behaviour. An
activity in a pipe can proceed with processing new input and run in parallel along with other activities which can be still busy with previous inputs.

4. **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 [11]) can lead to incorrect results.

To the best of our knowledge, none of the existing techniques for QoS prediction for service based (e.g., [11–16]) and parallel design pattern based applications (e.g., [17–20]) fully addresses all the aforementioned challenges respectively.

2 **Research Contributions**

In this research we present a design time QoS prediction algorithm for service based and parallel design patterns based applications. The QoS prediction techniques rely on two main ideas:

(1) Expressing the control flow of applications in terms of two simple cost compositors (Both and Delay) to address complex dependency, and
(2) Exploiting Monte Carlo simulations [21] to deal with the non-determinism, different possible results of service invocations and correlations in workflow.

For both service based applications and parallel design patterns based applications, we present algorithms that suitably deals with all the challenges and that is capable of probabilistically predicting the QoS of an application.

3 **Proposed solution & obtained results**

To estimate the QoS of parallel design patterns based applications, we proposed structurally recursive functions that associate, in a compositional way, each activity with a cost structure. Such cost structure is general, and it can be instantiated to define different QoS attributes, e.g., the time needed to complete an activity etc. The cost compositors are:

- The first cost compositor is a parallel compositor. Both(A, B) defines the cost associated with executing independently an activity with cost A and an activity with cost B.
- The second cost compositor defines the delayed cost of an activity which must wait for the completion of another activity before starting. Delay(A, B) defines the cost associated with executing an activity of cost A which must wait for the completion of an activity with cost B before starting.
For example, let us denote with \(aTime\) and \(bTime\) the completion time of the two activities. The completion time for running both activities in parallel is given by the maximum between \(aTime\) and \(bTime\). Instead, the completion time for delaying one activity after the other is obtained by summing \(aTime\) and \(bTime\), as the delayed activity can start only after the first activity is completed.

\[
\text{let Both}(aTime, bTime) = \max(aTime, bTime) \\
\text{let Delay}(aTime, bTime) = aTime + bTime
\]

Other QoS properties (like energy consumption, monetary cost) can be similarly defined using \(\text{Both}\) and \(\text{Delay}\). Our algorithms respectively converts a given workflow into an expression of \(\text{Both}\) and \(\text{Delay}\) and then estimate the QoS value.

### 3.1 Service based applications

For *service based applications*, the algorithm converts a given WS-BPEL workflow into an expression of \(\text{Both}\) and \(\text{Delay}\).

The algorithm \cite{22, 24} is implemented in a open source tool PASO (Probabilistic Analyser of Service Orchestrations) \footnote{The source code of PASO is available at 
\url{https://github.com/upi-bpel/paso}}. The inputs of the PASO (Figure 1) are a WS-BPEL workflow \footnote{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.} and probability distributions for the QoS properties of the invoked services as well as for the evaluation of the workflow branching conditions. The output of the algorithm is a probability distribution for the QoS properties (reliability, time and cost) of the orchestration.

A summary of the algorithm is given in Table 1.

---

**Fig. 1:** Bird-eye view of the input-output behaviour of PASO.
Table 1: Overview of prediction algorithm for service based applications.

<table>
<thead>
<tr>
<th>Workflow activity</th>
<th>Cost Compositor Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence(A,B)</td>
<td>Both(A, Delay(B,A))</td>
<td>If will executed after A.</td>
</tr>
<tr>
<td>Flow(activitySet,linkSet)</td>
<td>Both(activitySet)</td>
<td>All activities in activitySet will be executed in Parallel depending upon the links.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1. Sort activities using links to ensure that preceding activities are evaluated before current activity.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. For all activities in Flow:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.1 Identify all preceding activities of the current activity.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.2 Compute outcome of all preceding activities (Fault has precedence over stuck and success, Stuck has precedence over success).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.3 Compute cost of all preceding activities.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.4 Evaluate join condition for current activity.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.5 Evaluate transition condition for all outgoing links of current activity.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 Compute cost and outcome of Flow once all activities have been analyzed.</td>
</tr>
<tr>
<td>Scope(A,faultHandler)</td>
<td>Both(A, Delay(faultHandler,A)) or Both(A, Delay(Zero,A))</td>
<td>If evaluation of A yields a Fault, then faultHandler is evaluated. Otherwise cost of executing A is returned only.</td>
</tr>
<tr>
<td>Assign(name,expr)</td>
<td>Zero</td>
<td>We assume that Assign has Zero cost.</td>
</tr>
<tr>
<td>IfThenElse(guard,A,B)</td>
<td>Both(A, Delay(zero,A)) or Both(B, Delay(zero,B))</td>
<td>Evaluates guard condition using sampling function. If true, evaluate A else evaluate B.</td>
</tr>
<tr>
<td>While(guard,A)</td>
<td>Sequence (A,While(guard,A)) or Zero</td>
<td>Evaluates guard condition using sampling function. If false, the body of the loop is skipped. Otherwise evaluate the body of the loop and While again.</td>
</tr>
<tr>
<td>Invoke(partnerLink)</td>
<td>-</td>
<td>Use sampling function to sample a value for QoS properties and Success/Fault/Stuck outcome.</td>
</tr>
</tbody>
</table>
3.2 Parallel design patterns based applications

For parallel design patterns based applications, the algorithm converts a given parallel design pattern based application into an expression of Both and Delay. The algorithm is implemented in an open source tool PASA (Probabilistic Analyser of Skeleton-based Applications) \[25\] (Fig. 2).

PASA inputs (i) the description of a parallel application defined as a combination of basic activities (i.e., Nodes) and of the core stream parallel design patterns Comp, Pipe, Farm, Feedback, (ii) the size and the optional classification of the input stream (e.g., we can categorise the data items coming from the input stream to distinguish the energy consumption and completion time they require for being processed), (iii) the QoS required by each node to process each type of input, and (iv) the probabilities of a given input type to occur and of a given Feedback conditions to get satisfied. PASA also permits displaying the results (i.e., energy consumption and completion time) of the performed analysis in different formats (e.g., histograms). A summary of the algorithm is given in Table 2.

![Fig. 2: Bird-eye view of the input-output behaviour of the PASA analyser.](image)

Table 2: Overview of prediction algorithm for parallel design patterns based applications.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Cost Compositor Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipe ((A, B))</td>
<td>Both ((A, B))</td>
<td>A and B will be executed simultaneously. More precisely, B will wait for A to process an input (i) before executing it. While B will be busy processing (i), A could proceed with input (i+1).</td>
</tr>
<tr>
<td>Comp ((A, B))</td>
<td>Both ((A, \text{Delay}(B, A)))</td>
<td>B can start processing a given input only when A has completed processing such input.</td>
</tr>
<tr>
<td>Farm ((A, n))</td>
<td>Both ((\text{wQoS} \text{ of } n \text{ workers}))</td>
<td>We split the input stream among (n) workers. Execute (A) on (n) workers simultaneously and save cost in (\text{wQoS}).</td>
</tr>
<tr>
<td>Feedback((A, \text{Condition}))</td>
<td>Both((A, \text{Delay}(fQoS, A)))</td>
<td>Execute (A) &amp; save cost in (fQoS). Evaluate Condition using Sampling function. If True then item is routed back, (A) is executed again &amp; cost is saved in (fQoS).</td>
</tr>
</tbody>
</table>

[3] The source code of PASA is available at [https://github.com/ahmad1245/PASA](https://github.com/ahmad1245/PASA)
The algorithms for both PASO and PASA are implemented in F#.Net [20].

4 Future work

There are several possible directions to extend this work:

PASO and PASA can be extended to support new activities and languages. PASO is currently capable of analyzing a proper subset WS-BPEL. Support for other interesting WS-BPEL activities (e.g., Pick, Event Handlers) and workflow languages (e.g., YAWL, BPMN) could be included in PASO. Similarly, PASA currently models a simple set of parallel design patterns. Support for other interesting data parallel patterns (e.g., map, reduce) and frameworks (e.g., FastFlow) could be included in PASA.

Some forms of correlations could be introduced in the samplings. For instance, it would be interesting to consider some correlation among service invocations (e.g., if a service invocation returns a fault because it is “down for maintenance” it may be probable that the same result will be obtained in the next invocation) and in the input stream of parallel applications (e.g., bursts of data).

Last, but not least, the definition of the cost compositors Both and Delay can be extended to support new QoS properties (e.g., throughput, availability, and so on). An interesting extension in this perspective would be to provide users with a query language to specify their own QoS properties.

References


24. Leonardo Bartoloni, Antonio Brogi, Ahmad Ibrahim: Predicting the QoS of service orchestrations. Submitted for publication (Feb 2016)


Ensuring quality of digital services in and for dynamically changing environment

Anne Immonen
VTT Technical Research Centre of Finland, P. O. Box 1100, FIN-90571 Oulu
anne.immonen@vtt.fi

Abstract. The shifting to more open software development environments and the increased importance of data in software systems have caused the need of new software development models and methods. Dynamic, customer-controlled data-based software systems and services bring out new challenges, which new quality models and evaluation methods need to tackle. The main contribution of this work is to define how to ensure quality of digital services in the quality models/mechanisms of dynamic service architectures and how to develop these models and mechanisms for dynamic operation environments. The purpose of the research is to inspect how the service development and quality evaluation has changed when moving from the closed environments to the more open ecosystems, and to define the main phases and the key assets for digital service engineering in the ecosystem. Furthermore, the quality evaluation of software, services and data is surveyed and the key phases and elements for data quality management are defined. Finally, the concept of evolvable open data based digital service ecosystem is created, in which the ecosystem provides the assets for the quality evaluation for its members.

Keywords: architecture, service quality, data quality, digital service ecosystem

1 Introduction

Software development is evolving and shifting from isolated development environments to open innovation and co-development environments, called ecosystems. The ecosystems allow their members to act independently, strengthen their forces by cooperating, and create value in networks flexibly and dynamically under the common regulation. Openness breaks the boundaries around a company; the value networks are formed by several organizations to reach the solution. Furthermore, the growing extent of software systems are provided as services distributed across networks and fulfilling complex demands, even dynamically. Digital services can be anything that is entirely automated, and delivered digitally through an information infrastructure, such as web, mobile devices, communication network or any other forms of delivery. Therefore, a digital service is discovered on-line, accessed through a well-defined interface, and always controlled by the customer of the service [1]. People today have
free access to the Internet from all over, especially with the help of their mobile devices, causing new challenges for delivering services and information reliably. The role of data in today’s services has also changed; more and more services utilize freely available data instead of proprietary data. These changes have caused the emergence of new development models, methods and approaches for software systems and services. Also the quality evaluation models and methods have to be adapted to this shifting. The development approaches should introduce means for implementing the non-functional (i.e. the quality) properties, whereas the purpose of the quality evaluation is to verify that the software or service fulfills its non-functional responsibilities, such as reliability or availability requirements. In the case of data, the purpose of the quality evaluation is to ensure that the data fits for use by data consumers [2].

The research problem of this work is: “How to ensure quality of digital services considering frequently changing user requirements and capabilities of the dynamically changing ecosystem?” New quality challenges arise in the case of these new types of digital services, especially; how to verify the quality already in the development phase and evaluate it dynamically in the usage phase. The recent methods and approaches for quality prediction already at the architectural level are not adequate, applicable and rapidly adaptable for the challenges of quality evaluation of today’s services. This is emphasized, and also brings big challenges, especially in service ecosystems, where the quality problems affect several services and members in value networks. In digital ecosystems, the business and development environment is highly dynamic and the needs and demands of service customers are unclear and ever changing [3]. Support is required for detecting changes in the context, and for performing reactive adaptation according to the situation at hand. Furthermore, the ecosystem’s capability to perform actions should include the support services required to verify the quality of the services. However, the service engineering in digital service ecosystem has not been focused from the quality viewpoint in current research.

The transfer of the utilization from internal data to external, uncontrolled data causes new challenges for quality evaluation. These problems arise especially in the case of open data, which is based on the idea that certain data should be freely available to everyone to use and republish as they wish, without restrictions from copyright, patents or other mechanisms of control [4]. The poor quality of open data has been identified as one of the major problem that currently complicates the data utilization [5], [6]. Still, at the same time, the open data has been detected to provide several benefits both for service innovation and for business decision making [5], [7], [8]. Data quality evaluation has recently noted and some work has been done on data quality attributes and evaluation methods [9], [10], [11], [12], but still the common and standard practices are at the outset. The utilization of open data in business requires proper knowledge about data quality; how reliable, trustworthiness and valuable the data is for its intended use. The evaluation of data quality requires models, knowledge and supporting services. If these are provided by the digital service ecosystem, the open data can be certified for the usage of all the ecosystem members.

In this research, the earlier challenges in quality evaluation in service ecosystems and the quality evaluation of open data are brought together. The contribution of this research is to create the concept of evolvable open data based digital service ecosystem, in which the ecosystem provides the assets for the quality evaluation for its
members. The evolvability responds the capability of the ecosystem to adapt to changes. The quality evaluation is focused on open data, which in ecosystem context is considered as an open data service that encapsulates the open data for the usage of the digital service developers. The elements and activities of the open data quality evaluation are brought into the ecosystem context, and the main assets of the digital service ecosystem for the service engineering; activities, knowledge models and support services, are refined and extended to resolve the challenge of the open data quality certification.

2 Outline of objectives

This paper is based on the ongoing research on the quality evaluation of open data, the digital service ecosystems and the utilization of the open data in digital services.

The first objective of the research is to survey how the quality evaluation has evolved when moving from the evaluation of software and services to the data quality evaluation, and to define the key phases for quality evaluation of open data.

The second objective is to inspect how the service development and quality evaluation has changed when moving from the closed environment to the more open ecosystems, and to define the main elements and phases for digital service engineering in the ecosystem.

The third objective is to create a new model of a digital service ecosystem in which the varying elements; the open (big) data, the quality of data and the dynamic digital service development, are taken into account in the capability model, knowledge models and the service engineering model, and in a quality model that supports the open data certification.

3 State of the art

Quality attributes have been standardized strongly among software engineering [13], [14], [15]. Quality has also been taken into account systematically in many works in software architecture design [16], [17], [18]. When the faults and bad design decisions are detected in earlier development phases, such as at the architectural level, the modifications and corrections are easier and cheaper to implement. According to the surveys in [16], [19], a lot of methods for reliability prediction methods and approaches exists that are applicable at the architectural level, although they still have some shortcomings.

In the case of data, the quality issues are not brought commonly into use. The data quality challenges have recently arisen especially in the case of open data. These challenges include:

- There are no agreed definitions of data quality attributes.
- Data quality cannot be judged without considering the context at hand [9].
- The challenges caused by the growing amount of semi- and unstructured data, i.e. big data: volume, variety, velocity and veracity [20] of data.
• New ways of delivering data and information (digital services across Internet and mobile devices).
• User’s changed expectations and perceptions of data quality, “fitness for use” [21] in pervasive computing environments

One of the major challenges is to ensure whether the data fits for use of the certain context and situation; changing environment and user needs, and the dynamically changing situations in the markets (e.g. the data becomes suddenly unavailable) cause the need to continuously evaluate the data quality. Despite of the shortage of the quality evaluation practices of open data, the open data is seen as interesting to provide new business benefits, such as data-based content, ideas and basic functions, increased understanding about business opportunities, improved competitiveness, and potential new customers [6]. Several definitions for data quality attributes exist [9], [2], and some approaches have been introduced to achieve data quality [22], [23], [24]. However, the proper validation of these approaches in industry usage is far away. Furthermore, open data and data quality evaluation approaches have not been introduced in digital service ecosystem level.

Ecosystems have been interested researches increasingly, especially business ecosystems [25], [26] and software ecosystems [27], [28]. Recently, some research has been carried out on service ecosystems [29], [30], [31]. A service ecosystem is a socio-technical complex system that enables service providers to reach shared goals and gain added value by utilizing the services of other members in the ecosystem. The digital service ecosystems are part of service ecosystem, taking only the digital part into account [3]. Thus, the digital service ecosystem can be positioned between the business and software ecosystem, taking characteristics from both sides and filling the gap between the two. The main ecosystem elements include ecosystem members, i.e. service providers, service brokers, service consumers and infrastructure providers; capabilities that define the purpose of the ecosystem, its ability to perform actions and the rules of how to operate in the ecosystem; infrastructure that implements ecosystem capabilities; and digital services that are the products of the ecosystem.

The service engineering in the digital service ecosystem can be characterized under four features [3]: 1) service co-innovation, 2) service value co-creation, 3) enabling infrastructure, and 4) utilization of ecosystem’s asset. Several innovation methods exist for services that enable the open innovation in a value network [32], [33] as well as some models have been introduced to verify interoperability of services [34], [35]. Support is also provided for knowledge based service engineering that would enable to utilize the existing assets [36], [37]. However, the knowledge based service engineering have not been present in methods for ecosystem-based service engineering, neither the service innovation approaches take the knowledge bases into account. The identified methods and approaches for ecosystem-based service development are loose; they are only concentrating their own viewpoint and not working together. Clearly there still exists a lack of methods how to take the digital service ecosystem elements into account in service engineering. Furthermore, the elements should be extended and complemented to include the activities, models and supporting services to enable the quality evaluation.
4 Research method

This research will base on the current state of the art and knowledge, and the feedback achieved from industry, and will be carried out within real and laboratory environments. The research consists of five activities (see Figure 1). The first activity is performed purely based on the survey of the current literature about the software and service reliability and availability prediction and analysis methods. The second activity is performed in connection of an international project which is acting as an ecosystem, and the results are applied and validated in two different cases among project partners (i.e. the ecosystem members) involved in service requirements engineering. The third activity is performed based on the state of the art knowledge about open data ecosystems explored from literature and the state of the practice on data based business collected by interviews among industry. The fourth activity is performed within a national project based on state of the art literature of data quality evaluation and validation of the solution with an industrial case example. The fifth activity summarizes all the earlier work together, and complements the required elements for achieving the goal; the concept of the open data based digital service ecosystem.

Fig. 1. The activities of the research.

5 Outcomes and the stage of the research

The main outcomes of this research are the following:

1. Comparison frameworks and surveys for a) reliability and availability prediction methods at the architectural level [16] and b) methods and approaches for reliable dynamic service compositions [19], and a method for defining and representing the quality requirements and transforming them to architectural models [38].
2. The concept and service engineering model in digital service ecosystem.[3]
3. State of the art and state of the practice review of the open data based business ecosystem, and a concept of open data based business ecosystem [6].
4. A solution for evaluating the quality of open data [39].
5. A concept of open data based digital service ecosystem and open data certification process [40].

The main outcomes of this research have been published in recognized research forums. The first results of this research are published in [16], where a state-of-the art survey of the existing reliability and availability prediction and analysis methods is provided from the viewpoint of software architecture. The comparison framework defined in the paper describes the required characteristic of analysis methods from
context, user, method content and evaluation perspectives. The framework describes the required properties for reliability evaluation, and its usage enables the selection of the best suitable method for architectural analysis. The framework can be extended and adapted for future directions.

The development of new kind of modern software; services that are distributed across the network and adapted dynamically during run-time, requires new modelling and analysis methods and techniques to enable service reliability during run-time. In the work in [19], the required phases of the composite service design and execution are defined to achieve a reliable composite service. These phases are described in the form of a framework. A literature survey is performed of existing methods and approaches for reliable composite services to find out how they match with the criteria of the framework, and what are the shortages and development targets.

Quality issues are emphasized in the case of software product families, in which a faulty component or bad design decision can cause extensive problems concerning several software products. In product families, the products are built by exploiting shared architecture and common components. In [38], the QRF (Quality Requirements of a software Family) method is represented that focuses on how quality requirements have to be defined, represented and transformed to architectural models.

The work on digital service ecosystems were published in [3]. Since the concept of digital service ecosystem was new and not properly defined, a comparative definition of the properties of the business ecosystem, digital service ecosystem and software ecosystem is provided. Based on the state-of-the-art review, there exists a lack of methods of how to take the digital service ecosystem elements into account in service engineering. The work in [3] describes the main requirements and elements identified for ecosystem based digital service engineering and defines a requirements engineering (RE) method for digital service ecosystems. The work was validated in two industrial cases, where the ecosystem members use the RE method for specifying digital services and related support services.

The first research on open data is provided in [6], in which the first draft of open data ecosystem from the business viewpoint is defined. The requirements of such an ecosystem are collected with the help of interviews of the industry representatives and the motives and challenges of acting in the open data ecosystem are identified.

The unknown quality of open data was identified as one of the main challenges for open data utilization. The evaluation of the open data quality was the main concern in the work in [39] where the elements and phases of quality evaluation of open data in big data architecture are defined. The solution is validated with the help of an industrial case example, where the solution provides the verified data into company’s business decision making.

Finally, the work in [40] combines, adapts and extends all the earlier work, and introduces the concept of evolvable open data based digital service ecosystem. The work describes the main elements, and the required support services and knowledge models for open data certification and introduces an open data certification process according to which the open data is brought to the ecosystem and certified for the usage of the digital service ecosystem members.
The writing of the summary of the PhD thesis is planned to be completed during 2016. The supervisors of the PhD thesis are: Prof. Eila Ovaska (eila.ovaska@vtt.fi) as main supervisor, and Prof. Veikko Seppänen (veikko.seppanen@oulu.fi) as co-supervisor.

References


31
Impact-minimizing Runtime Adaptation in Cloud-based Data Stream Processing

Cui Qin
Supervisors: Klaus Schmid, Holger Eichelberger

Software Systems Engineering, University of Hildesheim,
Universitätsplatz 1, D-31141 Hildesheim, Germany,
qin@sse.uni-hildesheim.de

Abstract. Recently, cloud-based data stream processing has been rapidly emerged. In particular, it has become competitive in high scalability for processing huge amounts of data. During such processing, the actual characteristics of data streams may vary, e.g., in terms of volume or velocity. To provide a steady quality of the analysis results, runtime adaptation of the data processing is desirable. While several techniques for changing data stream processing at runtime do exist, one specific challenge is to minimize the impact of runtime adaptation on the data processing, in particular for real-time data analytics.

In this research work, we aim at performing a runtime adaptation in cloud-based data stream processing, namely, dynamically switching alternative distributed algorithms which has similar functionality but operate at different characteristics. The goal of this work is to provide a generic approach which can automatically determine the algorithm switch with quality guarantees on the data processing. To minimize the impact on the data streams as well as optimize resource usage during adaptation, we combine stream re-routing with buffering and stream synchronization along with a support of dynamic deployment of alternative stream processing algorithms into the cloud.

Keywords: Cloud-based data stream processing; runtime adaptation; impact-minimizing adaptation; algorithm switching; dynamic deployment

1 Introduction

Big data applications aim at processing huge or complex data sets, which usually cannot be handled by traditional approaches. Distributed stream processing [1], i.e., continuous processing of conceptually endless streams of data items, is a popular approach to realize such applications. Particularly, cloud-based data stream processing, driven by the trend towards cloud computing, has been rapidly emerged and become competitive in high scalability for processing huge amounts of data [9]. Well-known systems supporting cloud-based data stream processing include Apache Storm1 and Spark2. Such stream processing systems

1 http://storm.apache.org/
2 http://spark.apache.org/
aim at real-time processing of complex and distributed stream analysis tasks. During processing, the stream characteristics such as volume or volatility can vary over time. For example, in the financial domain hectic markets can cause bursty streams leading to changes of the stream characteristics of several orders of magnitude. In such situation, the current stream processing algorithm may not withstand the occurred bursty streams due to the heavy workload. To cope with those problems, adaptation of the data processing at runtime is desirable.

Approaches such as Borealis [3] or RTSTREAM [15] provide adaptation capabilities for continuous stream queries, such as changing the query program, however, they concentrate on a fixed set of database-like stream operators. In contrast, recent frameworks such as Apache Storm or Spark provides possibilities for developers to implement complex analysis algorithms, but currently do not provide much support for runtime adaptation. Cloud-based data stream processing supports high scalability to adapt for various workload while processing. One of examples is the Elastic Stream Computing Platform (Esc) proposed by Satzger et al. [13], which can dynamically scale up and down to adjust the computational capacities to handle peaks and off-peaks. With the availability of the public cloud infrastructure, it enables possibilities to provide virtually unlimited resources for running stream processing tasks with a new dimension of computational complexity. However, the utilization of the resources from the public cloud is directly visible to the monetary cost [8].

In this research work, we support adapting stream processing in cloud-based environments by switching among different processing algorithms, which provide similar functionality but operate at different runtime characteristics. This enables us to opportunistically utilize a better algorithm, e.g., at high load a faster, but more expensive algorithm such as a hardware co-processor, which can be utilized in other more urgent analyses at low load. The goal of this research work is to minimize the impact on the data streams, e.g., the disturbance on output results.

2 Problem Statement

In this section, we start with a naive stream redirection to figure out the potential problems in the runtime switching among distributed stream processing algorithms. Finally, we discuss the main goals that we want to achieve in this research work.

Figure 1 depicts the naive switch in terms of a data flow diagram, i.e., nodes represent (distributable) data processors and edges the data flow. The distributed stream processing algorithms are presented with several processor node. As shown in Figure 1, the processors $P_{1,1}$ to $P_{1,n}$ constitute Algorithm$_1$ ($P_{2,1}$ to $P_{2,m}$ for Algorithm$_2$). In addition, two guarding processors control input (Switching Element) and output (Join Element) streams. Assume that Algorithm$_1$ is the currently active algorithm. A signal, i.e., an asynchronous event sent to a processor, indicates the need to switch the active algorithm to another target algorithm $\uparrow$. In the naive stream redirection, the switch signal causes an
immediate re-routing of the data streams ②, i.e., the stream to Algorithm 1 is disabled and the alternative stream to Algorithm 2 is enabled at the same time.

In this naive stream direction, we can potentially see two problems. One is the overlapping results that could be caused by the queuing effects in the processors of the Algorithm 1 if there is no additional care at the connection between the Algorithm 1 and the Join Element after the switch. Another potential impact is the gap on the processing results that could occur if we immediately remove the resources on the Algorithm 1 after switching to Algorithm 2. Arising from such problems, we consider the following goals for our research work:

**G1. Transparency.** Our main goal for switching data processing algorithms is to minimize the impact on data streams, i.e., the switch shall happen transparently without disturbance on the processing results. More specifically, we focus on two sub-goals:

(a) **No missing or duplicated data.** Switching an algorithm at runtime must not cause data loss or duplication of data items.

(b) **Minimizing effects on output stream characteristics.** In addition to G1a, also the effect on further (application-) relevant stream characteristics such as latency or throughput shall be minimized.

**G2. Minimizing the switching time.** This is our secondary goal aiming at switching algorithms at runtime as fast as possible to reduce the time for causing disturbances to the streams.

### 3 Related work

Approaches such as Borealis [3] or RTSTREAM [15] provide adaptation capabilities for continuous stream queries, such as changing the query program. However, they concentrate on a fixed set of database-like stream operators and also do not support adaptation in cloud-based environment. For adapting stream processing in cloud, Satzger et al. [13] present an elastic stream computing platform...
dynamically attaching and releasing machines to adjust the computational capacities to the current needs. Similarly, Heinze et al. [9] support auto-scaling of data stream processing to address varying workloads and system characteristics problems. However, they do not address the adaptation on the stream processing algorithms.

There exist mechanisms to adapt stream processing algorithms in literature. The typical mechanism is the parameter adaptation of the stream processing algorithm, e.g, dynamically adapting the batch size to improve the performance of the streaming processing [7]. Furthermore, to cope with varying workload, Madsen et al. [11] present a dynamic load balancing approach parallelizing operators to scale up the processing. Similar approaches, such as [14] and [4], automatically extract data parallelism of distributed stream processing. Moreover, Chatzistergiou et al. [5] present a task reallocation approach reconfiguring stream executing jobs to minimize the transfer latency over cluster-based stream processing systems. Those mechanisms do support adaptation on stream processing algorithms but none of them exchanges processing algorithms at runtime.

In summary, to our best knowledge, except for our previous published work in [12] there are no approaches on dynamically exchanging alternative stream processing algorithms in cloud-based environment, which explicitly aim at minimizing the impact on the data streams.

4 Research Challenges

In order to fulfill the requirements discussed in Section 2, we identified the following research challenges as part of this research work.

1. **Determination of a "safe" switch.** To avoid the disturbance on the output result of the processing algorithm, especially its accuracy, we introduce the concept of a safe (transparent, gap-free) switch, which takes the characteristics of alternative algorithms into account. We aim at determining a feasible, if not even optimal point in time for switching. This requires us to classify different stream processing algorithms based on their characteristics and to provide different switch mechanisms for different groups of algorithms. The goal of the safe switch is to provide quality guarantees for each group of algorithms while minimizing the impact on the runtime algorithm adaptation.

2. **Theoretical formalization of the algorithm switch.** To provide the full concept of a generic approach for the algorithm switch, we would need to formalize the algorithm switch based on the "safe" determination as the theoretical part of this research work. This would focus on the theoretical categories of algorithms with different characteristics as well as the theoretical solution of switching among different combination of algorithms. To perform a "safe" switch, we also need to define the prerequisite for each category of algorithms.

3. **Data tracing.** In order to determine whether the data item is fully processed or still pending for the completion of its processing, we need to trace data
items for obtaining such information along with processing nodes of the
algorithm. This requires the acknowledgement of each data item to inform
its completion. Although recent data stream processing frameworks, such as
Apache Storm, support acknowledgement mechanism, we do not have access
to the detailed tracing information.

4. **Data synchronization.** To support achieving the main goal, i.e., trans-
parency mentioned in the requirement G1, data synchronization on both
algorithms is needed to ensure no data loss or duplicated data items before
switching to the target algorithm. Depending on the enqueued data items
as well as the synchronization mechanism, it may require the transferring of
data items to the target algorithm.

5. **Queuing control.** As discussed in Section 2, queuing effects in the origi-
nal algorithm could cause inconsistent results while performing an algorithm
switch. To cope with such effects, we would need queuing control over al-
gorithms to observe the input data feeding into the respective algorithms.
However, the amount of enqueued data items could also have a significant
impact on the switching time as we may need to transfer them to the tar-
get algorithm during data synchronization. For this case, we would adopt
Backpressure [6] or admission control [2].

6. **Dynamic algorithm deployment.** To optimize the resource usage, we
must support dynamic deployment of alternative algorithms, i.e., deploy-
only the selected processing algorithm, avoiding unnecessary processing
nodes in the cloud. For achieving such flexibility, the alternative algorithms
must support dynamical connection to the current processing.

## 5 Preliminary Solution and Result

In this section, we present our initial idea (more details can be found in [12])
as well as the preliminary results we achieved so far regarding the research
challenges listed in Section 4.

To handle the queuing effects explained in Section 2, we introduced an **En-
trance Queue** to explicitly control the input data feeding into the respective
algorithms. As illustrated in Figure 2, we represent the entrance queues as indi-
vidual processors (\textit{Intermediary}_1 and \textit{Intermediary}_2). The queues in the inter-
mediary processors remove an item only when it is fully processed by the last
processing node of the respective algorithm. This is indicated by an acknowl-
edgement signal (ack in Figure 2) sent by \textit{P}_{1,n} or \textit{P}_{2,m}, i.e., the last node of
the respective algorithm. Thus, the items in the queues are either pending to
be processed or emitted to the respective algorithm but not fully processed yet.
Moreover, we utilize a further queue in the **Switching Element** to control the
overall stream ingestion.

To maintain the state for the target algorithm, we currently apply a parallel
track strategy [10] to run the original and target algorithm for a time \(\Delta t_m\) in
parallel to enable the target algorithm for creating and stabilizing its state. \(\Delta t_m\)
the so-called warm-up time depends on the involved algorithms and can, e.g.,
be the time frame of a sliding window.
Let $Algorithm_1$ be the currently active algorithm. When $Switching Element$ receives the switch signal $\text{switch}$, we first warm-up the target algorithm in parallel to build its state, i.e., activating the stream to $Algorithm_2$ by duplicating the input items in $Switching Element$. At the end, $Join Element$ passes only the items of the active algorithm ($Algorithm_1$), i.e., it discards the output of the passive algorithms, in particular the output of $Algorithm_2$ in the warm-up phase. When the warm-up time is over, the actual switch happens by performing the data synchronization between both entrance queues, i.e., $Intermediary_1$ negotiates with $Intermediary_2$ the last processed item in $Algorithm_1$ as a basis for the queue transfer. As soon as synchronized items are passed to $Algorithm_2$, $Intermediary_2$ starts emitting data to $Algorithm_2$ and the output of processed data to $Join Element$ will be enabled. To minimize the switching time, $Algorithm_2$ processes synchronized items in parallel to the queue transfer, i.e., it starts processing real data already during queue transfer. Finally, $Intermediary_2$ notifies $Switching Element$ about the end of the synchronization as well as that $Algorithm_2$ took over the processing and $Algorithm_1$ is discarded.

**Fig. 2.** The preliminary solution of the algorithm switch.

We evaluated the initial solution on a cloud-based Storm cluster. As expected, plain stream re-routing suffers from queuing effects, item duplication and requires an overall switching time of more than 15 s. In contrast, the initial improved approach reduces the switching time to 60 ms (algorithms with no latency and no item transferring) or less then 110 ms (at 30 ms latency with transferring 2889 items) without impacting on the output accuracy.

### 6 Research Plan

To fulfill our listed requirements as well as address the mentioned challenges, we present the research plan of this PhD work below by listing the steps that we intend to approach.
Step 1. Conducting a systematic literature review on runtime stream processing adaptation. This step is used to obtain an overview on the runtime adaptation of data stream processing. It aims at gaining insights into different adaptation approaches in order to acquire the state of the art.

Step 2. Formalization of the ”safe” algorithm switch. This step aims at formalizing the algorithm switch considering the ”safe” concept as the theoretical part of this research work. It mainly contains two substeps. 1) Classification of the stream processing algorithms with different characteristics to be addressed in our algorithm switch approach. 2) Determining the prerequisite of each group of algorithms for performing a ”safe” switch.

Step 3. Design of a generic approach for algorithm switch. We aim at designing a generic switch approach which fulfills the requirements for the groups of algorithms formalized in the Step 2. The generic switch approach is intended to apply respective strategies for the algorithms with different characteristics to achieve different levels of quality guarantees.

Step 4. Evaluation on the feasibility of the proposed approach. Before starting to implement the solution, we plan to make an early evaluation on the feasibility of the proposed approach. In particular, we will focus on the programmability to prevent from the potential development problem.

Step 5. Design and Implementation of the switch toolbox. The aim of this step is to build a flexible switch toolbox which can easily integrate different switch strategies. This could support the implementation of the proposed switch approach and also benefit for comparing switch techniques with different achievements.

Step 6. Evaluation of the generic switch approach. As the last step, we evaluate the implemented switch approach. We mainly focus on two types of evaluation: a) Technical evaluation, e.g, the characteristics such as latency, throughput, etc. b) Developer study, e.g., the usability of our approach for developers.

Acknowledgments

This work is partially supported by the European Commission in the 7th framework programme through the QualiMaster project (grant 619525).

References


Towards a Unified Management of Applications on Heterogeneous Clouds*

Jose Carrasco

Supervisors: Francisco Durán and Ernesto Pimentel

Dept. Lenguajes y Ciencias de la Computación, Universidad de Málaga, Málaga, Spain
josec@lcc.uma.es

Abstract. The diversity in the way cloud providers offer their services, give their SLAs, present their QoS, or support different technologies, makes very difficult the portability and interoperability of cloud applications, and favours the well-known vendor lock-in problem. We propose a model to describe cloud applications and the required resources in an agnostic, and providers- and resources-independent way, in which individual application modules, and entire applications, may be re-deployed using different services without modification. To support this model, and after the proposal of a variety of cross-cloud application management tools by different authors, we propose going one step further in the unification of cloud services with a management approach in which IaaS and PaaS services are integrated into a unified interface. We provide support for deploying applications whose components are distributed on different cloud providers, indistinctly using IaaS and PaaS services.

Keywords: Cloud applications, multi-deployment, TOSCA, Brooklyn

1 Introduction

In recent years, Cloud Computing has experienced a growth in the demand of its services. The Cloud promotes on-demand access to a large number of resources throughout three service models, namely Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) [1], which allow cloud providers to offer services for current IT requirements, with scalability and elasticity as the most relevant ones, and allow users to tailor the used resources to their needs.

Vendors such as Google, Amazon, Cloud Foundry, etc., have implemented their solutions to this model by developing their own cloud service layers, with custom APIs that expose their resources. Most of these providers offer a set of similar services as regards functionality, but developed according to their own specifications. E.g., each supplier specifies its own Service Level Agreement (SLA) or Quality of Service (QoS), supports a concrete set of technologies, etc. The proliferation of these solutions has also increased the number of issues to be

* This work has been partially supported by Spanish MINECO/FEDER projects TIN2014-52034-R and TIN2015-67083-R; Andalusian Gov project P11-TIC-7659; and Univ. Málaga, Campus de Excelencia Internacional Andalucía Tech.
addressed in cloud computing, mainly related to the diversity of providers and their solutions, giving place to the vendor lock-in problem [2], and hampering the portability and interoperability in the definition and usage of services.

Due to this lack of standardisation, developers are often locked-in to concrete cloud environments, since they have to adapt their developments according to the specifics of the vendors that will be used to run their applications. This heterogeneity affects the entire lifecycle of systems, from design time to release/deployment, which complicates the development of portable applications and the integration of services of different providers to achieve cross-deployments. In this context, migrating components between different platforms seems impossible.

Given the current state of Cloud Computing, it looks reasonable to offer to developers mechanisms to deal with the restrictions to the portability and interoperability of applications. From the developers’ point of view, we believe it would be very useful to have an environment in which we could build full detailed application descriptions in an agnostic way, supporting the use of services of different offerings to deploy our applications, and abstracting from the constraints of concrete providers. Furthermore, it would make sense to distribute the different modules of an application over services of different providers. This would allow us to optimise the usage of cloud resources, since we could select, for their deployment, and given the requirements of each of the modules of an application, and requirements of the application itself as a whole, the services with best features for each of the modules of our application. Moreover, we plan to go one step further, and analyse the portability between abstraction levels, initially focusing on IaaS and PaaS.

Once applications have been deployed and are running, using services of specific providers, developers may need to modify the cloud environment where the application is being executed due to many reasons, such as an application updating or different cloud events. For example, the performance of services could be altered, e.g., by a modification in the QoS by the provider, affecting the application performance or its cost. Developers could also modify the cloud resources used by their applications, for example, by adding new cloud services to provide new application features. It may be useful counting with mechanisms supporting the management and reconfiguration of cloud applications.

Additionally, it may be useful to users to have facilities for the migration of application modules between different cloud levels in order to maintain the performance and optimise the resources usage and minimise the cost. For instance, given an increase in the workload of an application, it could be beneficial to migrate some of its modules to PaaS, in order to take advantage of the automatic scalability facilities of this kind of services.

The rest of the paper is structured as follows. Section 2 describes our research challenges. The research plan and the current state of our research are explained in Sections 3 and 4, resp. Section 5 exposes our conclusions and future work.
2 Research Challenges
The main goal of this thesis is to develop an environment that offers an homogeneous management of IaaS and PaaS services, and enables a methodology to describe applications and the required target cloud resources, providing developers with mechanisms to improve the portability and interoperability of applications. Moreover, it will allow users to choose the cloud resources whose features best adapt to their applications’ requirements, with support for the deployment of each of the modules of applications using the PaaS or IaaS services that better fit their needs. In the following, we elaborate on the descriptions of our goals.

Unification of IaaS and PaaS cloud services. We plan to develop a common API that will unify cloud services independently of their abstraction level, for IaaS and PaaS. To achieve this, we will analyse the different service features and restrictions in order to find common patterns and abstract them under a unified interface. This unified level will offer to users a transparent and simple usage of different cloud services, allowing them to focus on the functionality of these services, while the complexity of using and integrating their interfaces is hidden by the unified API. We plan to build this API by homogenising services with different properties in order to build a normalised upper layer. Given the existing diversity, trying to homogenise all the functionalities of each provider will most probably not be possible. To minimise this problem, we will try to maintain these functionalities by using lower layers, with the goal of providing as many services as possible.

Description of applications and cloud services. We believe that the way to address the portability and interoperability issues is by developing an agnostic modeling framework to describe applications and the used cloud (IaaS and PaaS) services and resources. With this framework, users will be able to build full-detailed descriptions of their applications, including all the knowledge about the capabilities, requirements, kinds of services to run the application, etc., regardless of the concrete providers over which the application will be finally deployed. We plan to build on current standards, such as CAMP and TOSCA, in order to propose a standardised, powerful and flexible application-modeling environment.

Integration of the modeling and the unified API. A unified API will offer a homogeneous management of different services. An application model will allow us to detail all the knowledge about an application. Then, we believe that by joining both elements, API and agnostic modeling, we will be able to provide an environment which will allow portable applications to be modeled and deployed using the unified API features in a standardised manner, providing a complete application lifecycle management. Then, any services supported by the unified API will be available for users to deploy modeled applications without requiring any knowledge about the concrete provider interfaces.

Development of a functional prototype. We will develop a functional prototype in which we will experiment with the accomplishments related to the previous goals, and to show its viability and to evaluate its advantages and disadvantages.
Post-deploy management of applications. Although not one of the core goals of the work being described, we will also study the implications of our proposal on the management of applications once they have been deployed and are running. Specifically, we will consider aspects such as the monitoring of cloud applications whose modules are deployed using services of different providers, possibly at different levels, and how SLA policies may be specified (e.g., auto-scaling policies).

Hot reconfiguration of applications. Given agnostic application descriptions, it seems natural to consider the possibility of moving application modules from the services they are deployed on to other ones with better features, or for a better adjustment of the application needs. We may even think on performing such reconfiguration operations at runtime.

3 Research Plan
In this section we structure the work of this thesis on the following phases, detailing the tasks to develop for each of them.

Analysis of the related work
• Exhaustive analysis of the state of the art on homogenisation and cloud management. We will review current practical and theoretical proposals and related standards. We will also analyse their implementation plan.
• Systematic analysis of the features and restrictions of the different cloud offerings in order to determine the key aspects to consider when carrying out the proposed homogenisation. This will be made by defining different deployment use cases involving different service levels.
• Study of deployment-related concepts using services of multiple clouds (multi-clouds).
• Review of related open projects, with special emphasis on those using standards, including an evaluation of their capabilities and limitations.

API composition and unification of IaaS and PaaS services
• Classification of different cloud services in terms of their functionalities and the services of the cloud offerings that will be supported by our approach, establishing a preliminary approach of the unified API.
• A first prototypical development of the unified API. We will most probably first develop independent versions for IaaS and PaaS, which will later be unified under a common interface.
• Our implementation efforts will be integrated inside an existent open project supported by an active community. We will pay special attention to Apache Brooklyn an open project that offers a flexible and robust management of IaaS services of a large number of providers.

Application Modeling
• Analysis of the different concepts related to the management of applications and cloud services that will be supported by our modeling facilities to provide flexible and extensible mechanisms to describe systems.

1 Apache Brooklyn: https://brooklyn.apache.org/
• Development of a modeling proposal, supporting the definition of applications according to the results of the previous step, addressing the significant management and capabilities differences between the different providers. We will also study the use of current standards, initiatives and open projects focused on the normalisation of applications and the description cloud services.
• Development of a generic nomenclature to identify and reach the target providers that will be used to deploy applications, making sure that the nomenclature is flexible enough to support any provider properties, and enabling the distribution of the different application modules over different providers (cross-deployment).

Validation of the proposal
• Revise the diversity of use cases proposed on the first phase focusing on different characteristics in order to check the supported providers under diverse restrictions.
• Application of the use cases to specific deployment scenarios which will be composed by different providers according to real situations.

Post-deployment strategies
• As possible extensions, we will consider the monitoring concepts and mechanisms to add them to the common API and the application modeling.
• We will research on management policies, such as auto-scaling, which will be based on the previous monitoring experiments.
• We will study migration techniques, determining how application modules can be moved between services of different providers and its abstraction levels.

4 Current State
We present in this section some of the goals we have already achieved.

4.1 Trans-cloud management
Independent tools and frameworks have emerged with the goal of integrating, under a single interface, the services of multiple public and private providers (see, e.g., [3], and [4]). In a very short time, these platforms have evolved according to the mode in which developers can take advantage of integrated cloud services to expose and run their systems. Terms such as multi-cloud [5], cross-cloud [6], federated clouds [7], or inter-clouds [8] have been used for deployment platforms with the ability of distributing modules of an application using services from different providers.

The main differences between these approaches lie on the different ways of handling the connections between modules deployed on different platforms. However, in all these attempts, platforms allow operating simultaneously with a single level of service to deploy applications, i.e., all the components of an application are deployed either at the IaaS level or all at the PaaS level (see, e.g., [9], [10] and [11]). From this, with the goal of unifying cloud services, we propose in this thesis a second dimension in which deployment tools integrate IaaS and PaaS levels under a single interface. Then, this will allow developers
to deploy their applications combining services offered by providers at any of these levels. Following the evolution in terminology, multi-/cross-/inter-cloud, we envision trans-cloud management tools without the limitations we currently have. Trans-cloud mechanisms enable one of most important goals of this work, the unification of IaaS and PaaS cloud services (see Section 1). The idea behind trans-cloud is to be able to build our applications by using available services and resources offered by different providers, at IaaS, PaaS or SaaS level, using virtual machines or containers, according to our needs and preferences. We will focus on IaaS and PaaS in this thesis.

4.2 Application Modeling

There is a lot of work on methodology descriptions in the literature, including many projects, standards and initiatives, as Cloud4Soa [10], CAMP [2], Roboconf [3] and mOSAIC [4]. After analysing the most relevant related work, we consider TOSCA (Topology and Orchestration Specification for Cloud Applications) [5] as a standard that provides a useful framework on which basing our application modeling because it defines a very flexible model for the description of cloud applications, the corresponding services, allowing their relations to be specify explicitly by using a fully service topology, containing all the knowledge about the applications. Furthermore, it allows the description of procedures to manage services using orchestration processes by using plans.

Currently, we only take advantage of the topology specification of TOSCA, what allows us to describe the knowledge about applications independently of any cloud resource restrictions, and integrate the different features and requirements of the different provider abstraction levels in the same model, the application portability.

4.3 Toward a Unified API

We propose the development of a common API to unify the management of IaaS and PaaS services. After analysing the mechanisms to manage the cloud of different alternatives, such as OpenTOSCA [6], Alien4Cloud [7], Cloud4Soa and, we decided to base our work on Apache Brooklyn, an open project with an active community behind. Brooklyn can manage the provisioning and deployment of cloud applications, can monitor applications’ health and metrics, and handle the dependencies between components. It enables cross-computing features through a unified API to manage IaaS services offered by various providers.

Brooklyn provides an API for the management of IaaS cloud services for a great number of providers and establishes a lifecycle for the management of services and applications. We have extended this API with facilities for the...
management of PaaS services of platforms based on Cloud Foundry, providing an homogeneous access to IaaS and PaaS services. We have integrated the PaaS management in all the Brooklyn levels but without modifying its API. Then, we have obtained a prototype with a common API that manages IaaS and some PaaS services (currently, Cloud Foundry-based platforms) in an unified manner. We have tested this API by building portable applications, and deploying them using different IaaS and PaaS providers. Indeed, we have obtained in this way a first implementation of the proposed trans-cloud mechanisms. However, the current model only supports IaaS and Cloud Foundry-based platforms, so it will be necessary elaborate on PaaS levels of different providers in order to understand their capabilities and requirements and analyze how they would be integrated in our approach by means of extending the current model.

Although we mentioned in Section 2 that post-management is not one of the main goals of this work, but we think the knowledge about this issues would be useful to enhance our IaaS-PaaS integrated model, e.g., migration of applications shares the concern of the management of different providers in order to move the application. Hence, we have developed proofs of concept of some scaling policies both for IaaS and PaaS services by taking advantage of Brooklyn’s capabilities.

5 Conclusions and Future Work

We propose the development and use of a common API to unify the management of IaaS and PaaS cloud services, making their use completely uniform. We allocate this proposal inside what we call trans-clouds, which extends cross-cloud application deployment and management by supporting the portability and interoperability of application modules from different providers and at different levels. We propose a TOSCA-based agnostic modeling of applications and cloud services, which allows us to specify the characteristics and requirements of any system to be deployed in the cloud. The standardised description of applications and cloud resources and the homogenous service API significantly reduce the portability and interoperability issues related to vendor lock-in, facilitating the reusability of cloud services. By having an agnostic model of our system may greatly simplify migration, or simply decision change. Indeed, with our approach, each component may be deployed at one level or the other just by changing its location. It is worth noting that the proposed thesis project is not an implementation exercise on an existing deployment tool, but an innovative general approach to ease the cloud deployment of applications, enforcing the independence of both cloud providers and cloud models.

We have developed an operational prototype built on the well-established Apache Brooklyn tool in order to test our trans-cloud ideas. Brooklyn provides support for a large number of IaaS providers. Thanks to our efforts in integrating Cloud Foundry into Brooklyn, it now also provides access to PaaS Cloud Foundry-based providers such as Pivotal Web Services or Bluemix.

Part of the research in this thesis was developed in the context of the Seaclouds project [11], and some preliminary results related to the thesis plan described here have already been published in [12][13][14].
Much work remains ahead. We plan to analyse new providers in order to extend the supported PaaS services and technologies. Thus, current model will be extended in order to integrate PaaS levels of new providers, such as Heroku or OpenShift. Due to providers heterogeneity, the new providets has to be carefully analyzed in order to elaborate on how they should be added to our approach. Furthermore, we plan to study the possibility of using the flexibility and scalability mechanisms available for PaaS to develop management policies to react to applications’ events.

References

Information about the Author(s)

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

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