Modelling of Context-Adaptable Business Processes and their Implementation as Service-Oriented Architecture

Vojtěch Mates\textsuperscript{a}, Marek Rychlý\textsuperscript{a}, Tomáš Hruška\textsuperscript{a,b}\*  
\textsuperscript{a}Brno University of Technology, Czech Republic  
\textsuperscript{b}Brno University of Technology, IT4Innovations Centre of Excellence, Czech Republic

Abstract

The paper deals with a description of a system capable of automatic adaptation of managing logic. The logic is continuously adapting according to a user-driven and environmental context. The system observes the context and modifies implementations of business processes in order to keep optimal and fault-tolerant performance by utilising different services in service-oriented architecture.  
© 2014 The Authors. Published by Elsevier B.V.  
Selection and/or peer-review under responsibility of the Organizing Committee of ECE 2014.

Keywords: business process adaptation, SOA, Web service orchestration, context-aware system

1. Introduction

The context-adaptability is an important and desired feature especially for processes situated in dynamic environments Weber et al. (2004). The adaptability to the environmental changes allows rescheduling a process, i.e., to use its new realisation which is optimal for the environment and meets the same process goals and produces the same outputs as the original realisation. The rescheduling helps to keep fault tolerant processes with an optimal configuration.

Business processes automation and IT support is often implemented by means of service-oriented architecture (SOA) Lins et al. (2012). A business process can be managed via a business process management system which can use SOA as implementation architecture of the business process or its part. In SOA, a software system implementing the business process or its part is decomposed and distributed in the form of autonomous but cooperative

\*E-mail addresses: imates@fit.vutbr.cz (Vojtěch Mates), rychly@fit.vutbr.cz (Marek Rychlý), hruska@fit.vutbr.cz (Tomáš Hruška)
components known as services. This architecture has many advantages, such as better scalability and fault-tolerance. Moreover, SOA principles, such as loose coupling, statelessness, or reusability, allow easy runtime modifications of a composed system by changing its particular components [Karastoyanova et al. (2005)].

However, realisation of the context-adaptive processes in SOA brings several issues that need to be addressed. The rescheduling of a business process requires strong knowledge of its context at both business and implementation level, which includes an administration cost of the rescheduling; QoS requirements for the process, its sub-processes, and activities; performance indicator values for current and potential realisations of the process; etc.

The goal of this paper is to provide a conceptual approach to description and realisation of context-adaptable business processes by SOA. In the paper, we propose a model for both descriptions of the adaptable metrics. We also propose a method of automatic evaluation and assignment of process definitions to their best process realisations based on observation of the actual context and historical and current values of the performance indicators.

The rest of this paper is organised as follows. Section 2 introduces the context adaptability in more detail including related state-of-the-art work and an outline of our approach. The approach is described in-depth in Section 3 by conceptual models of context-adaptable business processes and the context adaptability itself. In Section 4, we describe a method to evaluate and assign process realisations by means of a product dependency tree. Section 5 continues with the description of implementation of context-adaptable business processes as SOA. Finally, Section 6 describes evaluation and discussion of our approach and Section 7 concludes the paper with an outline of future work.

2. Context-Adaptability in Business Processes

In business process modelling, there is a strong need for the context adaptability in order to build flexible processes. These processes should be able to adapt to changing conditions during their performance time, to keep their optimal performance despite of unreliable resources, or to provide secure fault-tolerant solutions. The adaptability can mean to reschedule a business process completely or to select one of many process variants how to achieve process goals, which may be very complicated and time consuming.

The context of a process has a significant influence on its performance. According to our previous research [Pospíšil et al. (2013a,b)], the context-dependency is very important for production processes as well as for project-oriented processes. In a factory production line implementing a production process, particular information is often available in runtime only or just before runtime of the process (e.g., an actual delay or availability of resources of which a specific product will be processed). The project-oriented processes, i.e., the processes controlling a project, are highly context-dependent on availability of resources, time, and cost, with significant influence upon process performance.

In both cases, potential rescheduling of the processes may be very expensive in time and resources. Therefore, it is useful to have a predefined set of variant solutions how to achieve specific process goals. In order to decide which variant is the best for achieving particular process goals, all available data about previous performance of a process has to be gathered continually and interpreted as fast as possible, to be able to represent the most actual state of the context, and has to be evaluated periodically according to many aspects. This data processing and evaluation can be done automatically, which will be described in this paper, however, data types, sources, and implementation methods as well as aspects for evaluation have to be defined manually.

2.1. Related Work

The related work typically addresses different phases of a business process life-cycle, namely modelling, execution/management, and measurement [Mendling (2008)]. Our approach combines these phases in order to perform evaluation as soon as possible and to use results of the evaluation for adaptation of process definition at runtime. There are several approaches to modelling of adaptive business processes. The challenge is to provide flexibility and offer process support at the same time [Schonenberg et al. (2008)]. Most of the approaches (e.g., BPMN, XPDL, Petri Nets, etc.) are focusing on the modelling of processes in imperative way which is not as suitable for changing process structure in runtime as declarative approaches, see Hull et al. (1999); Pesic et al. (2007). There are also declarative approaches, e.g., Declare [Pospíšil et al. 2013a,b], which can provide a certain amount of freedom in performance of the process, because the rules can be easily added even during performance time. However, in Declare, the rules are not very suitable for processes which have to be controlled because of too many variations how to perform the process (e.g., in banks or insurance companies), while avoiding the rules may have a significant consequences.

Execution of business processes in the second phase of their life-cycle can be controlled by two types of business process management systems: rule-based and case-based. The rule-based systems manage business processes by predefined rules describing how to react in particular situations (e.g., in a case of a new production order). These
systems are based on best practices and supported by many well-established approaches, e.g., for monitoring of the rule-based managed processes in the measurement phase. However, the rule-based systems are not very suitable for dynamic modifications of the processes at runtime according to context changes which cannot be described by the predefined rules. Existing approaches, e.g., Ellis et al. (1995), focus mostly on updating of a process to its new version during its runtime, which could take days or even years without possibility to stop the process for the update.

The case-based systems, the second class of process management systems, manage business processes on the basis of previous experiences in the form of cases describing the process variants. These systems are very flexible, because the process definition is fully-specified in all its variants of individual cases and it is always possible to introduce a new case according to the current needs. However, there may be a problem with monitoring and controlling of the process performance in the measurement phase which may be very different in each case. This can lead to breaking business rules, problems with performance evaluation, and unpredictable quality because of too much freedom in the processes.

2.2. Outline of the Proposed Approach

In our approach, we propose the way to describe a business process including its adaptability, to control the process automatically taking into account its context information, and to adapt the process at runtime based on continuous measuring of the process performance. In comparison with the related work which has been described in the previous section, we propose a hybrid modelling approach to model business processes in both declarative and imperative way. The approach allows declarative modelling of business processes as high-level abstractions describing process goals, however, components representing realisations and implementations of the business processes can be modelled in imperative way (e.g., by work-flow or orchestrations). Our approach is also a combination of rule-based and case-based process management systems. The approach allows controlling variants (the cases) of possible realisations of the business processes by predefined business rules. The realisations corresponding to a given business process are found by goal-based approach when each realisation has to fulfill goals required by the business process.

The next sections describe business process modelling and management in two different levels for two different groups of users: managers or process analysts, and business process engineers, IT analysts, or IT architects. Managers or process analysts often looked at a business process in terms of what will be the result (a product) of the process and what is necessary to have in order to accomplish the process (resources). They may ask questions such as “Is it possible to make the certain product under these conditions?” or “Is it possible to make the product better in these specific external context conditions?” On the other hand, business process engineers are responsible for well-designed process definitions and are usually focused only on building solutions which solve particular problem, for example by a work-flow or an orchestration of IT services in the case of IT analysts or IT architects.

The basic idea can be represented by an automatic real-time decisions process which will compare the possible ways of reaching certain business process goals using the currently available processes realisations and context information. The goals, which represent business process high-level abstractions, should be defined by managers, and the available processes realisations should be described by business process engineers. Both goals as the high-level process abstractions and the processes realisations form a model of context-adaptive business processes and should be described at design time while context information is provided at runtime by an existing work-flow management system.

Finally, the process management system in our approach is driven by queries defined by its users at runtime. Each query describes goals of the main business process, which produces outputs required directly by the users, not by another business process as a part of its input resources. Moreover, the query describes particular properties of preferred realisations of the process (e.g., preferred cost of its product or time constraints for its performance). Our approach allows the process management system to monitor business processes at runtime and use their model, runtime context information, and the queries to adapt the processes in order to optimise their performance according to actual needs.


In this section, a conceptual model of context-adaptable business processes is introduced. It defines a collection of terms (e.g., “process”, “process realisation”, or “resource”) and their possible relationships (e.g., “the process is realized by…””) which can be used by a process analyst to describe the adaptable business processes. In this way, the model can be used as a meta-model for modelling of context-adaptable business processes. Moreover, it defines data supporting and operations implementing the adaptability of business processes in general, i.e., the dynamic modification of the business processes according to various aspects (e.g., current values of performance metrics).
This section serves as an introduction and a conceptual base for Sections 4 and 5 dealing with the context-adaptability and the implementation, respectively, in more detail.


Modelling of context-adaptable business processes has to address four aspects: process definition, process realisation, process implementation, and process performance. The **process definition** describes a process from an abstract point of view and according to the process goals as a set of output resources produced by the process. The **process realisation** follows up with a description of realisation of the process by a single activity or an orchestration of sub-processes transforming input resources to the previously described output resources. The **process implementation** describes a particular implementation of the process realisation, for example, as a Web service implementing the process activity or a BPEL process implementing the orchestration of sub-processes. Finally, the **process performance** describes definitions and values of metrics related to the process realisation.

The conceptual model is depicted by the UML class diagram in Figure 1 and it is based on the aspects described above. The process definition is represented by class `Process` and the “should produce” association with class `Resource` describing output resources produced by the process. The process definition can have assigned one process realisation. The process realisation with its input resources is represented by class `Realisation` with class `Resource` linked by the “consumes” association, respectively. Class `Realisation` has to be specialised to class `Activity` representing a single activity or to class `Orchestration` representing an orchestration of sub-processes (instances of class `Process`).

The process realisation also produces output resources, more specifically, the same output resources as the corresponding process definition describes. In the case of the activity (i.e., class `Activity`), those particular output resources are represented by the “produces” association with class `ProducedResource`. In the case of the process orchestration (i.e., class `Orchestration`), the output resources of the orchestration are the resources produced by its all sub-processes, that is instances of class `Resource` associated with the instance of class `Process` for each sub-process of the orchestration.

The process implementation is represented by class `Implementation` and its specialisations to classes `ActivityImpl` and `OrchestrationImpl` for implementations of activities and orchestrations, respectively. The
conceptual model takes into consideration orchestrations described in BPEL or Orc language and activities implemented as Web services or as human tasks. These implementations are represented by classes BPEL, Orc, WebService, and HumanTask, respectively. Finally, performance of the process realisations can be measured by class Measure is linked to information on a metric represented by an instance of classes RuntimeMetric (e.g., a supplier’s response time) or DesigntimeMetric (e.g., a set-up cost of the first production cycle) and a list of measured values represented by instances of class Value.

In the conceptual model from the previous section and Figure 1, the context adaptability is represented by separation of the (abstract) process definition and the (concrete) process realisation. A business process (class Process) can adapt to the context by selection of its appropriate realisation (class Realisation), so there can be different realisations for different contexts. Furthermore, the selection of a particular realisation appropriate to a given process definition is done on the basis of output resources. More specifically, output resources produced by the process realisation have to include output resources prescribed by the process definition. The produced resources can be obtained by method Realisation.getProducedResources(), while the prescribed resources are linked by the “should produce” association between classes Process and Resource. Method Realisation.getProducedResources() gives all instances of Resources which are produced by an activity (for a process realised by the activity) or by all sub-processes participating in an orchestration (for a process realised as the orchestration of its sub-processes). The mapping of a process definition to an appropriate process realisation is carried out for each single execution of the process. The appropriate process realisation has to produce prescribed output resources and to meet a user-defined query indicating preferred values of given metrics. The process realisation selection and execution is represented by method Process.findAndSetRealisation() and proceed as follows. At first, all convenient process realisations are found by method Process.findRealisations(). These realisations have to be active (attribute Realisation.enabled), must produce satisfactory output resources (method Realisation.getProducedResources()), and their measures have to match actual or grouped values stated in the query (method Realisation.evaluateQuery()). Finally, the convenient process realisations are ordered by their priority (Realisation.priority) and the first of them is selected and executed.

4. Process Realisations and a Product Dependency Tree

In the case of more variants how to achieve business process goals by its various process realisations, which is typical especially for project-oriented processes, one of the variants has to be selected according to desired conditions. However, it is often problem to evaluate which variant is better, because the decision could be very complex. In order to make the decision as good as possible, we need to evaluate possibility of mapping of the business process to all its possible realisations, and also to all possible realisations of all their sub-processes in the case of decomposition by orchestrations. This can take a lot time which could cause delay in the process performance.

In this section, we describe construction of a product dependency tree which maps business processes to their best realisations through the whole hierarchy of process decomposition. The tree is based on a query (see Sections 2.2 and 3.2) and on process realisations defined by business process engineers. The best realisation is selected by its produced resources (the desired products) which meet goals of a given business process and by preferences stated in the query.

4.1. A Product Dependency Tree

The product dependency tree is defined as a directed acyclic graph \( G = (N; E) \) where each node \( n \in N \) contains a set \( n(p) \) of business process abstractions and a set \( n(a) \) of process realisation that are activities (the realisations which cannot be decomposed, contrary to orchestrations).

The root node of the tree contains an empty set of the activities, \( n(a) = \emptyset \), and an elementary set with the abstraction of a main business process only, \( |n(p)| = 1 \), i.e., the process, which is producing output resources consumed directly by its user, not by another processes (it is the top process in the hierarchy of process decomposition). Each leaf node of the tree contains an empty set of business process abstractions, \( n(p) = \emptyset \), and a non-empty set of the activities, \( n(a) \neq \emptyset \).

\( ^{†} \) The conceptual model is not limited to the listed implementations of orchestrations and activities, i.e., another implementations are possible (e.g., an orchestration by SCXML description, a “business rule” activity, etc.).
Each edge \((n_p; n_c) \in E\) in the tree, which connects a parent node \(n_p\) to one of its children nodes \(n_c\), represents a set of mappings of all process abstractions from the parent node set \(n_p(p)\) to a particular combination of their realisations, which have business process abstractions of all their sub-processes in set \(n_c(a)\) of child node \(n_c\) in the case of realisations by orchestrations and all activities in set \(n_c(a)\) of child node \(n_c\) in the case of realisations by activities.

In other words and informally, each non-leaf node of the product dependency tree represent mappings of all process abstractions at a particular level of the hierarchy of process decomposition to a particular combination of their realisations. Then, each branch in the tree, i.e., a maximal path in the tree from its root to some of its leaf nodes, represents a particular and complete realisation of the main business process through its whole hierarchy of process decomposition, the desired result of a process management system. The mapping on each edge of the tree is done by matching, or a dependency of, goals of business process abstractions and produced output resources (the products) of process realisations. Because of the dependencies of the products, the tree is called “the product dependency tree”.

4.2. Building the Product Dependency Tree Based on a Query

The query declares the final products of a main business process whose realisation is required (e.g., a product shipping order), available input resources which can be consumed by the process, and preferences and constraints for the resulting realisation (e.g., shipping cost limits and a maximal delivery time). The resulting product dependency tree should provide, in one of its branches representing all possible ways to achieve the goals of the products, the best realisation of the main business process through its whole hierarchy of process decomposition. The realisation has to provide required products, consume input resources defined in the query only, and to meet desired preferences and constraints according to the query (e.g., an optimal shipping order at low cost and minimal delivery time). To build the dependency tree, each business process participating in hierarchy of decomposition of a main process, including the main process itself, has to be defined as a process abstraction with a set of produced output resources (products delivered by the process, at least information that the process is finished). Moreover, each process realisation which may participate in the dependency tree has to be described by its activity or orchestration, a set of produced output resources (products of the realisation), and a set of consumed input resources. If a process realisation needs some inputs which have to be provided by other processes, there is a dependency between the current process realisation and other process providing inputs as their products. Finally, each process realisation has assigned its performance profile with metrics (e.g., a performance time, costs calculated from fixed and variable costs depending on time, success rate, performance indicators of Web service implementations, etc.). The performance profiles are continuously monitored by a process management system and describe behavioural history of all process realisations in the system.

The dependency tree contains all possible ways of getting the desired output resource produced by a main business process. The tree \(G = (N; E)\) is built from a query by top-down approach as it follows:

1. The root node is created and its \(n(p)\) contains an abstraction of the main business process as declared in the query.
2. Each leaf node \(n\) with non-empty set of process abstractions, \(n(p) \neq \emptyset\), is analysed. For each possible combination of process realisations which can be mapped to the process abstractions in set \(n(p)\) and meets constraints and conditions in the query, a new child node \(n_c\) is created. A process realisation can be mapped to a process abstraction if output resources which should be produced by the process abstraction are a subset of output resources which are produced by the process realisation (i.e., the process realisation provides products defined by the process abstraction). For each process realisation which is an orchestration, process abstractions of its all sub-processes are added into set \(n_c(a)\). Moreover, each process realisation which is activity is added into set \(n_c(a)\).
3. The second step is repeated until each leaf node \(n\) in the tree has \(n(p) = \emptyset\).

4.3. Evaluating the Query

To find the best realisation of a main business process through its whole hierarchy of process decomposition, we need to evaluate the previously build product dependency tree and the query together. In the tree, each branch, i.e., a maximal path in the tree from its root to one of its leaf node represents a particular and complete realisation of the main business process through its whole hierarchy of process decomposition. Then, all branches in the tree are evaluated according to criteria set in the query (the branch which fulfils the criteria best gets the highest value). Finally, the evaluated branches are sorted according their value and a branch with the highest value represents the best realisation of the main business process according to the query. In the case of several branches with the highest value, the choice is done by user-defined priority of the realisations or randomly.
The above mentioned evaluation criteria are related mostly to the process performance and to available context information. The decision tree may be constant as long as there is no modification of the query and no changes in process abstractions and available process realisations. However, the query has to be re-evaluated when a decision which variant is currently the most suitable is needed, based on the actual process performance and context information. Nevertheless, it is not necessary to re-evaluate neither all branches nor every single process realisation in a branch. We need to re-evaluate only such branches which are affected by changes in the process performance or actual context information. Moreover, the re-evaluation needs to be performed only for those nodes in the affected branches which are directly related to the change or precedes in the path another nodes related to the change. Therefore, continuous re-evaluation of a query, which is necessary for business process context adaptability, may be considerably optimised.

4.4. An Example of the Query

A general business process usually contains a human task and Web services (WSs). A user who needs a result of the process need not know the process realisation in detail and to understand the complete process structure. The realization often depends on particular user requirements, e.g., on a response time, a cost, and a quality of the process product.

In our approach, the requirements of a user for a process realisation are described by a query, which is written in a SQL-like language. By the query, the user describes what business process is he looking for in terms of its origin (there can be several “databases” of processes), its products, and performance indicators of its eventual realisation. The result of processing of the query is the best realisation of the business process according to the query.

To demonstrate our approach, let us consider the following query resulting from a simple use case. A user would like to run a high demand computational application, e.g., a large-scale real-time network simulation which can be implemented in several different ways, by different distributed algorithms orchestrating many services of different providers. For example, a simulation can be performed by a group of commodity low-cost computers, however, the progress will be slow and the results may be inaccurate. On the other hand, the same simulation can be performed by a super-computer which is quite expensive; however, the user will get fast and precise response.

Therefore, based on particular requirements and available possibilities, the user can make the following query “select the least expensive implementation of the simulation as an orchestration of available WSs that produces the final product where the response time is less than 1 second, maximum cost is 2 Euro, and the precision level is 90”.

Formally, the query can be described in the SQL-like language as follows:

```
SELECT p FROM processDatabase /* a process variable name, the source for the query */ /* final product definition (a process abstraction) */ WHERE p.finalProducts = "the large-scale real-time network simulation" /* constraints of context variables based on performance history of p */ AND (p.responseTime < 1 second) AND (p.cost <= 2 Euro) ORDER BY p.cost LIMIT 1; /* order the results according to their costs and return the first */
```

To accomplish the query and execute the resulting process (its best realisation), these steps have to be followed:

- Focus on the main process which is derived from the product “the large-scale real-time network simulation” and build the dependency tree, as it is described in Section 4.2.
- Select a branch of the tree such that it satisfies the constraints of the response time shorter than 1 second and maximum cost is lower or equal than 2 Euro.
- Order all suitable branches by costs (the branches represent hierarchies of suitable process realisations).
- Choose the first branch from the list by the order above.
- Assign the first process realisation in the branch to the main process and the rest of process realisations in the branch to its sub-processes according to their level in the hierarchy, as it is described in Section 4.3.
- Launch the main process (its realisation).

Various context variables can be used in a query. These can be defined by a user or a process specialisation (e.g., parameters of a desired product). However, the most important are response time, cost, location, and quality of service.

5. Context-Adaptive Business Processes and Service-Oriented Architecture

In the previous sections, the context adaptability of business processes has been described as a dynamic assignment of a particular process realisation to a given process definition driven by user-defined criteria and by
features and performance indicators of available process realisations. So far, we have thought in terms of business processes and their decomposition and performance. In this section, we will take into account implementation details of business process realisations and discuss mapping of context-adaptive business processes to SOA services.

5.1. Implementation of Context-Adaptive Business Processes

The context-adaptive business processes are implemented by process realisations which “realise” process definitions as it is described in Section 3.1. In the other words, each process realisation can have assigned a process implementation. The process realisation and the process implementation are different views of exactly the same business process. The first one takes into account business perspectives (e.g., process decomposition and input/output resources), while the second one focuses on implementation techniques (e.g., service and human-task implementations). Both views are interconnected as each process realisation specifies the purpose of the corresponding process implementation. Moreover, the type of a process realisation, which can be an orchestration or an activity, affects its implementation, which can be a BPEL/Orc orchestration or a Web service/human task, respectively. Analogously, the implementation determines the resources required by the process realisation, e.g., a human task may require qualified persons.

Based on the two views, we can distinguish two types of the adaptability, namely (business) context adaptability and (technical) implementation adaptability, as it is depicted in Figure 2 (for the context, see also Figure 1). Generally, the context adaptability is managed by a business analyst, while the implementation adaptability is solved by a business process engineer or an IT analyst/architect. In our approach, we focus on the context adaptability to allow optimization at the business level by automatic mapping of process definitions to process realisations. Moreover, at the business level, each realisation represents also its implementation which affects the its resources and measured performance. Therefore, for multiple realisations of the same process with different implementations, the context adaptability decides also which of the implementations will be utilised in the process because each realisation has assigned at the most one implementation. In this way, the mapping of process realisations to their implementations needs not to be considered.

The implementation of context-adaptive business processes proceed as follows. In the beginning, a business analyst defines a set of business processes and their realisations including output (produced) and input (consumed) resources. For each process realisation, a set of the output resources produced by the realisation have to include (to be a superset of) the output resources that should be produced by the corresponding business process according to its definition (also see Section 3.2). The business process realisations can be found by decomposition of a main business process (a top-down approach) or by description of all possible realisations of partial business processes and their composition (a bottom-up approach). Each realisation should include definitions and values of its design-time metrics (e.g., set-up costs) and also definitions and initial, current, or historical values of run-time metrics (e.g., a response time) if they are available. With this information, current mappings of process definitions and their realisations can be automatically re-evaluated to find an optimal initial decomposition of the main business process. After this step, each business process is realised by a hierarchically decomposed and orchestrating sub-processes to the level of individual activities.

In the second step, business process engineers and IT analysts/architects implement the previously defined process realisations. Each realisation is mapped to its (one) implementation, which can extend or affect the previously described input resources and metrics. Then, the system is defined as SOA including implementation details of orchestrations, individual services, and human tasks, and it is ready to deployment, testing, and trial run.

Finally, the implemented business process realisations are continuously re-evaluated at runtime, based on actual values of relevant metrics. Before each single run of a business process in the system, its process realisation can be automatically re-assigned according to the actual needs and values of performance indicators (measured metrics).
At each time, the resulting hierarchy of mappings of business processes to their realisations (for example, see Figure 3) is correct because output resources produced by each process realisation always include all output resources that should be produced by the corresponding (mapped) business process by its definition. Moreover, the hierarchy is always optimal as each process realisation in the mappings was selected from all possible realisations (producing the required output resources) according to their performance and user-defined priorities. The performance of a process realisation is determined by performance of its current implementation, in the case of an activity, but also by performance of all sub-processes (their particular realisations), in the case of an orchestration.

5.2. Quality of Service and Implementation of Performance Metrics

The quality of service (QoS) and the performance metrics play key roles in context-adaptive business processes. The adaptability of business processes does not change the process goals; however it affects their QoS by maintenance or improvement of process performance. In our approach, we utilise runtime metrics and design-time metrics. The first are used to measure performance of process realisations at runtime and to evaluate their suitability as final realisations of defined processes, while the second allows including hidden costs into the evaluation, such as resource utilization or set-up costs (i.e., changeover costs associated with switching from current process realisation to another). Based on the metrics values, each process can adapt to a process realisation with the best QoS and affordable costs. Performance metrics of process realisations are defined at design-time. Together with definitions of business processes, a business analyst defines process performance metrics based on business goals. These process performance metrics are then translated by a business process engineer and an IT analyst/architect into performance metrics of process realisations. Moreover, in the case of runtime metrics, business process monitoring tools have to be implemented to support runtime metrics by continuous or event-driven/triggered measurements of process realisations. Finally, the process performance metrics are used as evaluation criteria in business process adaptability, i.e., during mapping of business processes to their optimal realisations. For example, a process performance metric states that “average duration for order processing is less than 60 minutes” restricting mapping to “automatic order processing” realisations as a manual order processing cannot meet 60 minutes limit for working hours or holidays.

In SOA, the runtime metrics of process realisations should measure performance and QoS of individual Web services and their orchestrations. The following Web service performance metrics have been adopted from Kalepu et al. (2003): availability and reliability of the service, price, throughput, response time, latency, performance, security, accessibility, regulatory, robustness/flexibility, accuracy, servability, integrity, and reputation.

6. Evaluation and Discussion

Potential applications providing fault tolerant and adaptive solution in environment of distributed systems should address many issues, such as the lack of possibility to have some influence in order to control every part of a system, which is caused by the adaptability, or fault-tolerance of service providers, which can be solved by redundancy in providers or alternative service realisations. However, the process adaptability and resource sharing are limited mainly by user requirements. For example, one group of users’ needs fast responses at any cost (e.g., for life-critical systems) while other users may prefer lower cost. Therefore, a resulting system has to implement both process
realisations, different for each user group according to their needs, even if both realisations provide the same products. Our approach can automatically adapt to a changing environment based on user requirements.

The concept of adaptation at business process level has been previously analysed in Mates and Hruška (2012) as rescheduling of projects with focus on causes of the rescheduling. Each sub-process has been modelled as an object with its own dynamic profiles which describe its possible behaviour. Our approach extends the concept of dynamic profiles from Mates and Hruška (2012) to dynamic process realisations. For example, in realisation of a business process by means of SOA, the process would be realised as a service orchestration where orchestrated services would be realisations of individual sub-processes. Continuous observation of performance of these services and other user-defined metrics representing a context of the process allows to change possible process realisations, i.e., to switch service orchestrations and implementations. A resulting system would be very flexible (adaptation to the context, it easy to add new services at runtime, etc.) and would be able immediately find how to keep optimal performance based on a user-defined query which describes priorities for desired optimal behaviour.

7. Conclusion

Main contribution of our approach is an automatically controlled and optimal dynamic orchestration of processes by context dependencies. By means of our approach, the processes can adapt to changing environmental conditions by selection of particular process realisations while keeping performance requirements. The approach deals with both business and implementation level of the processes. At the implementation level, we utilised service-oriented architecture, so the processes can adapt, e.g., to changing response time, cost, or availability of orchestrated services.

The current approach can be improved. The future work includes an extension of the approach with the ability to analyse combinations of service performance properties and process input resources, because we expect that a service may have different performance according to its different input configurations (the current approach evaluates suitability of a service as a process representation by its performance in producing of desired outputs, i.e., without any knowledge of the actual services inputs).

Acknowledgements

This work was supported by the BUT FIT grant FIT-S-14-2299, MPO Czech Republic MPO TIP FR-TI3 039, and by the European Regional Development Fund in the project CZ.1.05/1.1.00/02.0070 (IT4Innovations Centre of Excellence).

References


