Electronic Communications of the EASST
Volume 37 (2011)

Workshops der wissenschaftlichen Konferenz Kommunikation in Verteilten Systemen 2011 (WowKiVS 2011)

Towards NFC-Aware Process Execution for Dynamic Environments

Kristof Hamann, Sebastian Steenbuck, Sonja Zaplata

12 pages

Guest Editors: Horst Hellbrück, Norbert Luttenberger, Volker Turau
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Towards NFC-Aware Process Execution for Dynamic Environments

Kristof Hamann[1], **Sebastian Steenbuck**[2], **Sonja Zaplata**[1]

[1]Distributed Systems and Information Systems
Computer Science Department, University of Hamburg, Germany
{hamann | zaplata}@informatik.uni-hamburg.de

[2]Saarland University, Germany
s9sestee@stud.uni-saarland.de

Abstract: A flexible execution of business processes requires to deal with dynamic environments and to adapt to specific situations detected at runtime. However, flexibility needs to be restricted in a reasonable way which fits the requirements of the process initiator. An important aspect is the specification of user-defined constraints with respect to the non-functional characteristics (NFCs) of the execution. However, most existing approaches for such descriptions and corresponding service selection mechanisms are computationally complex and thus hinder a flexible runtime distribution of the process including resource-restricted clients such as, e.g., mobile devices.

This paper addresses the problem of NFC-aware process execution in such dynamic environments. Therefore, the paper presents an NFC meta-model and a corresponding language to support process modelers in expressing their non-functional requirements in a way which facilitates runtime decomposition and to dynamically derive local NFC specifications from remaining global requirements on process level by more simple heuristic approaches. Based on that, an initial algorithm is introduced in order to evaluate NFCs during runtime.

Keywords: Business Process Management, QoS, Flexibility

1 Introduction

One of the major challenges in execution and management of business processes is to cope with increasingly dynamic environments. To allow for the required flexibility, *Service-Oriented Architectures (SOA)* provide a paradigm to support the rapid, low-cost and non-complex composition of distributed applications [PTDL08]. In an SOA, a network of loosely coupled services provide the building blocks for composite services and finally leverage whole business processes. The approach fosters the vision of a dynamic binding in order to take advantage of the heterogeneity and variability of available services. Thus, service selection is often not only controlled by functional demands (i.e. the type of functionality and the service interface), but is also driven by non-functional requirements such as costs, availability, response time, or security issues. Non-functional *requirements* and *offers*, or – in general – *non-functional characteristics* (*NFCs*) of a service denote all aspects which can be used by clients in order to evaluate service quality

[DLS05]. In consequence, non-functional characteristics are an important criterion to differentiate between services which hold the same functionality, but which differ with respect to the user's needs in a current situation. For example, a business user accepts an advanced price for high availability and short response time of a service, whereas a home user is often willing to accept a time delay in order to reduce costs.

Considering business processes, non-functional characteristics can apply to single activities (*local statements*) as well as to the whole process (*global statements*). As an example, the process initiator may want to limit the costs for the whole process and additionally ensure that specific activities hold other conditions, such as security encryption for the execution of confidential tasks. During the execution, it has to be ensured that the activities of the process are executed by only invoking those services which comply with the corresponding local and global statements. In order to realize an NFC-aware execution of business processes, on the one hand, there has to be a formal description of the local and global statements to be matched to the characteristics of the available services [RCJ02]. On the other hand, an algorithm is needed in order to evaluate the statements and to find a suitable set of services which fulfill the requirements with respect to the business process model. In consequence, a large field of previous and ongoing research deals with the analysis and specification of such requirements, the detection of available services and the development of composition algorithms which, preferably, target at the identification of the overall optimal set of services in order to optimize all given non-functional characteristics of a process (cp. [YL05]). These approaches are well suited for relatively static environments where service properties do not change very often and process instances are executed in a similar way, so that the optimal configuration can be calculated in advance and the results of such complex selection algorithms can be re-used in several process instance executions.

However, business process execution environments do not always provide such stability. The need for flexible business collaborations and exploitation of external resources often require an ad-hoc distribution of the process spanning several execution engines which also include resource-restricted systems such as mobile devices (cp. [HHGR06, KZL06]). In such heterogeneous and dynamic environments, non-functional characteristics are even more important: Available services are characterized by a high level of dynamism in type and quality, and well known stationary services can have completely different characteristics in mobile environments, e.g. costs and allowed size of e-mails in contrast to SMS. In addition, formerly negligible characteristics are now of essential importance, e.g. such as the location of process execution. Although non-functional characteristics are hence a very important part of process execution in dynamic environments, the necessity of evaluating non-functional characteristics at runtime and resource limitations of participating systems disqualify many current approaches.

This work therefore investigates NFC-aware business process execution for dynamic environments where not only the control flow of the process but also the non-functional characteristics of services and systems cannot be determined before the execution time of each single activity. The basic idea of this approach is to specify user-defined NFC statements in a way which – as far as possible – facilitates runtime evaluation of NFCs. Therefore, this paper is structured as follows: Section 2 examines existing approaches and related work in the research field of non-functional requirements for business processes. The main contribution is divided into two parts: Based on existing standards, Section 3 introduces a language to describe non-functional characteristics and corresponding statements with respect to their runtime evaluation. Subsequently, Section 4 intro-

duces an algorithm for selecting suitable services based on such pre-defined statements. Finally, the paper is concluded in Section 5 by summarizing first results and ongoing research.

2 Background and Related Work

According to the two main challenges of an NFC-aware execution of business processes, this section introduces corresponding fundamentals and evaluates related work for service selection algorithms and description of non-functional characteristics.

2.1 Algorithms for Service Selection

Business processes are usually described by defining the control flow over a set of activities, each representing one step of work to be done. In an SOA, every activity a_i can be implemented by a set of services S_i, each providing the required functionality. This set can be provided at design time or, in a dynamic environment, can be discovered at runtime. Based on the set of detected services, a selection algorithm has to pick one service $s_{i,k} \in S_i$ for each activity a_i to be executed. This is done with respect to the non-functional characteristics c_j of that service which can be evaluated with a function $Q(c_j, s_{i,k})$, if applicable. It is assumed, that services implementing the same activity exhibit the same types of characteristics.

In a service composition, the process of finding a sufficient or optimal solution has to be performed on two levels: On the local level, each activity is evaluated independently without taking into account global requirements, such as an overall cost limit. On the global level, the entire business process with its global requirements is examined. Finding solutions on the local level is comparatively trivial because required characteristics can be compared to the characteristics of available services. Furthermore, a weighting function which prioritizes the requirements can be used in order to select an optimal solution. Assuming that there are m activities each with n available services, this procedure has a complexity of $O(n)$ for each activity and $O(m \cdot n)$ for the whole process. In contrast, the naive approach for global optimization is to evaluate every possible execution, which is in $O(n^m)$. This is reflected by the fact that the problem can be modeled as the *Multiple-Choice Knapsack Problem* which is *NP*-hard [YL05].

In dynamic environments such as mobile ad-hoc networks, information about available services and thus their corresponding characteristics often cannot be determined in advance. Thus, non-functional characteristics have to be detected and evaluated at runtime based on the current situation. In the worst case, this results in a step-by-step late binding of services which is advanced each time an activity of a process has to be executed. In particular, this means that it is not possible to calculate an optimal configuration of service assignment for the entire process. In consequence, many existing optimization techniques cannot be applied and algorithms of a high complexity should be avoided in order to allow for an efficient service selection at runtime.

Based on these observations, the following assumptions help to reduce the problem: In a dynamic environment, only a temporary optimization is possible because services with a better quality can enter the system at any time. It should therefore be assumed that without waiting for an infinite time an *optimal solution* cannot be achieved. Instead, selecting a *suitable solution* which meets the range of declared requirements decreases time for service detection and thus allows for keeping the number of suitable services as small as possible. If, e.g., a user is willing

to accept services with a *response time* $\leq$ *1 second*, all compliant services are considered to be equal – even if there is a single service with a much better quality, e.g. $\leq$ *1 millisecond*. Furthermore, in many cases optimization is not useful at all: For example, it is sufficient if the stated requirement *number of supported encryption algorithms* $\geq$ *1* is fulfilled, but it is not required that the number is maximized. Thus, service detection can be finished immediately if at least one compliant service is found.

Second, as local characteristics can be evaluated with a low effort, global characteristics should be decomposed to single activities as far as possible. Remaining global requirements need to be evaluated by a low-complex service selection algorithm and require an appropriate handling of the unknown behaviour of the process's control flow, i.e. concerning the number of cycles and the selection of alternative branches. As a simple (but unsatisfactory) solution it can be assumed that a process has no cycles or branches. However, if this is not given, the number of passes through the cycles and the number of selected branches can be estimated based on prior experiences, e.g. by integrating application-specific knowledge about the probable execution of the process.

In case of mobile ad-hoc networks, it can furthermore be assumed that there is a relatively small number of available services, that processes are comparatively short and that services are not known until the process is executed.

In summary, this gives rise to the requirements for a suitable algorithm: low complexity, rapid detection of a first compatible solution which can be optionally optimized later, and the consideration of a user's goal during the optimization. In the following, three relevant existing approaches are investigated with respect to these requirements:

Discarding Subsets [JMG05] is a backtracking algorithm. It builds a tree containing the available services in a way that each level represents the decision for one activity. Hence, for each possible combination of services there is a path through the tree. In order to reduce the effort, subtrees are excluded if at least one non-functional requirement cannot be satisfied anymore, or the current solution cannot become better than an already known one. However, the algorithm will normally identify the optimal solution and thus needs the corresponding amount of time.

Berbner et al. [BSR+06] propose a two step algorithm: First, the non-functional characteristics are translated in order to be optimized using linear programming on a relaxed problem. The second step is to use a backtracking algorithm to find a valid solution for the original problem.

WS_HEU [YZL07] is a heuristic-based algorithm consisting of three steps: First, a valid solution is found using a heuristic which requires that all characteristics use the same range of values. Second, it tries to advance the detected solution by replacing single services until further replacements would violate the requirements. In order to further advance the solution, replacements are taken into account which violate the requirements, but which can be satisfied with an additional replacement of another service.

While the focus of these algorithms is the maximization of the user's value, the goal of this work is to find an acceptable solution which can be derived at runtime and which is also suitable for more resource-restricted devices. The approach of *Discarding Subsets* has the ability to efficiently provide a sufficient solution if it is executed without further computationally intensive optimization (cp. Section 4). However, to make use of such a heuristic approach, non-functional requirements have to be specified in a way which supports their ad-hoc evaluation. The next section therefore derives requirements for an appropriate meta-model and analyzes most relevant corresponding description languages.

2.2 Languages for Describing Non-Functional Requirements

As a foundation for a runtime selection of services as introduced in Section 2.1, an appropriate NFC description language should satisfy several requirements which are briefly motivated and summarized in the following. The requirements 1 to 4 arise from the basic goal of an NFC-aware process execution as described in Section 1, while requirements 5 to 7 result from the need for an ad-hoc evaluation. In addition, the requirements 8 to 10 represent general goals in order to support the development and modeling of non-functional characteristics.

1. In order to be applicable in the context of business processes, the language must be able to define the **composition behavior** of NFCs for composed services. In particular, local and global characteristics have to be specified unambiguously and in a way which allows assignment to process parts.
2. Respecting differing user preferences, requires the definition of multiple profiles in order to provide different **levels of quality**.
3. For reasons of interoperability, the technologies should be capable of being integrated into existing standards, i.e. web service standards (**compliance to standards**).
4. Statements about services have to be **fine-grained** in order to differentiate between multiple operations combined in one service.
5. The language must be usable for both service consumer and service provider in order to express **requirements resp. offers** of services and processes. Otherwise, requirements and offers have to be translated in a common language every time before comparison.
6. If two non-functional characteristics conflict, the language should provide a mechanism to nevertheless allow an optimization. For ease of use, this should be done by allowing the definition of **priorities**.
7. The functional and non-functional description of services should be **syntactical separated** in order to allow a flexible adjustment of NFCs in case of changing conditions.
8. In order to ease modeling and enhance reusability, it should be possible to define new characteristics by **refining** existing ones.
9. The description of characteristics should not be restricted to single application domains, hence the language must be **generically applicable**.
10. In order to describe characteristics which are not realizable with the integrated concepts, the language should be **extensible**.

Based on these requirements, several languages which formalize non-functional characteristics have been studied. A short overview is presented in the following.

The *Web Service Level Agreement (WSLA)* language specification [LKD+03] allows the definition of Service-Level Agreements (SLA) with a focus on monitoring these SLAs. Therefore, the language contains many constructs that are not needed for the pure description of non-functional characteristics. The underlying model complies widely to the model used in this work, but it does not support levels of quality.

WS-Policy [VOH+06] is a framework for providing policies on service quality and corresponding policy requirements. Non-functional requirements can be integrated in the policies, but the intension is rather to use external XML namespaces for domain-specific extensions. WS-Policy integrates very well into other WS-* standards.

	WSLA	WS-Policy	WSOL	CQML	HP Labs
Composition	–	–	–	–	–
Levels of quality	–	+	+	×	○
Compliance to standards	+	+	○	–	+
Fine-grained	+	+	+	×	+
Requirements vs. offers	+	○	○	+	–
Priorities	–	–	–	–	–
Syntactical separation	+	+	+	×	+
Refinement	+	○	○	+	○
Generic applicability	×	×	×	+	–
Extensibility	+	+	+	–	–
good: +	average: ○	poor: –	not applicable: ×		

Table 1: Evaluation of existing description languages

The *Web Service Offering Language (WSOL)* [TPP02] allows the definition of non-functional characteristics on different levels of quality. However, the required characteristics should be defined in a separate language. Though, following the suggestion to use ontologies could be too complex for resource-restricted devices.

Although the *Component QoS Modeling Language (CQML)* [Aag01] targets software components, the approach can also be applied to services, because both a component and a service describe functionalities with a well-defined interface. CQML describes non-functional characteristics comprehensively and in a generic way. However, process flow is modeled directly in CQML, whereas this work assumes the existence of an independent process description, e.g. a WS-BPEL process. Furthermore, CQML is based on an EBNF syntax instead of XML.

Sahai et al. (HP Labs) examine [SDM01] when an SLA should be checked, which data should be involved, where the monitoring should happen, what is assured and how it is assured. To answer these questions, the work assumes that non-functional characteristics and requirements are known at design-time. However, as shown, this is is not true for dynamic environments.

Table 1 summarizes the results of the analysis. Although none of the existing standards and approaches fulfill all requirements, a combination of WS-Policy and CQML would satisfy 8 of 10. Therefore, Section 3 introduces an approach using WS-Policy and an enhanced approach based on CQML in order to describe non-functional characteristics. With this enhanced combination all 10 requirements can be met.

3 Description of Non-Functional Requirements and Offers

In order to model non-functional characteristics in a generic way, the underlying assumption is that many characteristics have common properties and behavior. For example, both requirements *response time* > 30 and *price* < 20 are measured numerically and exhibit a total order. In this section, a language is introduced which can be used to define such properties of non-functional characteristics in a generic way. With this approach, service provider and service consumer only need to agree on a common semantic sense of the characteristics. However, this part is not within the scope of this work. Rather, it is assumed that the participants have a common perception of

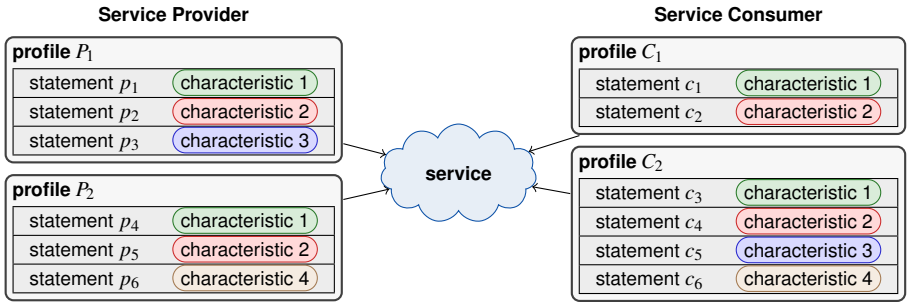

Figure 1: Model for describing non-functional characteristics (based on [Aag01])

identifiers such as *price*.

The presented language for describing non-functional characteristics and requirements is based on a meta-model which has its foundation in the work of Aagedal [Aag01] as well as of Ardagna and Pernici [AP06]. As shown in Figure 1, the main concepts are *(non-functional) characteristics*, *statements* and *profiles*. The definition of *characteristics* include the above mentioned properties and behavior of a specific non-functional characteristic, such as price. A *statement* restricts the possible range of values for a characteristic. Both service provider and service consumer can define multiple *profiles* in order to provide different service levels, each consisting of multiple statements. Service providers can specify which values they offer, whereas service consumers can provide their requirements.

In order to achieve the requirements presented in Section 2.2, this work uses a combination of WS-Policy and an enhanced approach based on CQML. Profiles are realized with the capabilities of WS-Policy. Non-functional characteristics and statements are expressed in an XML-based language, which uses concepts and ideas of the non-XML language CQML replenished with the missing constructs for composition and priorities. In the following, the elements of the meta-model are described in more detail.

Characteristics Non-functional characteristics can be specified with the element *characteristic* and are uniquely identified with a *name*. Characteristics can be either *numeric* or non-numeric (*set*). Numeric characteristics have a *type*, e.g. *integer* or *double*. The *direction* (*increasing* or *decreasing*) specifies if smaller or larger values are preferred. Restrictions in the range of values can be added with an *invariant* element.

If the characteristic is used to describe the requirements of the whole process or parts, the composition behavior needs to be clarified. For most non-numeric and some numeric characteristics the statement can be processed locally (element *local*). This is e.g. true if a particular encryption algorithm is demanded. Otherwise, a function has to be specified which aggregates the local values of singular services into global values of a whole process (*aggregate-function*). Depending on the characteristic, this can e.g. be sum, product, minimum, maximum or arithmetic mean [AP06]. For parallel execution it is possible to incorporate all services or just the critical path (element *parallel*). The critical path is the path with the worst characteristic values, e.g. with the highest execution time [AP06].

Non-numeric characteristics are specified by its *elements*, i.e. a set of strings. Optionally, an *order* of the elements and the *direction* towards better values can be specified if applicable. Con-

sidering service composition, the element *local* can be used if the characteristic can be checked for each service independently. Otherwise, if an order exists, the minimum or alternatively the union set is used.

Complex characteristics can be defined by declaring multiple characteristics using the element *composite*. With the element *specialized*, an existing characteristic can be limited to a certain range of values.

Statements Describing both NFC requirements and offers is done by constructing constraints on characteristics. Likewise, statements can be defined by limiting the range of values of a simple characteristic as well as of a complex characteristic, or by further limiting the range of values of another statement. A *priority* helps if two statements are contradictory.

Profiles WS-Policy provides constructs to combine multiple statements (*all*) as well as to specify alternatives (*ExactlyOnce*), which can be used to define different service levels [VOH+06]. The resulting definitions can be integrated into existing WSDL files on behalf of the according WS-Policy extension.

The introduced language can be used to specify both requirements of a service consumer and NFC offers of a service provider. Both is done by using the *statement* element. To evaluate if the NFC offers match the requirements, it has to be checked, if the offered values are a subset of the required values: $o_i \subseteq r_i$. An offer's profile O satisfies a requirement's profile R if this is true for every requirement statement: $O \text{ satisfies } R \Leftrightarrow \forall r \in R : \exists o \in O : o \subseteq r$. This can be used to find appropriate services on the local level (cp. Section 2.1). A corresponding algorithm which is able to check statements on the local and global level is introduced in the next section.

4 Dynamic Service Selection for NFC-Aware Processes

Service selection denotes the operation of picking one service from a set of available services. The proposed algorithm for selecting services for process execution consists of three steps: In the first phase, services not satisfying the local requirements are removed. The second phase is a backtracking algorithm using a heuristic to find an efficient solution with respect to the global requirements. In a final step, the detected solution is optimized rudimentarily. These three steps are described in more detail in the following.

Evaluating local requirements Since evaluating local requirements needs much less efforts, the algorithm begins the first step with attempting to transform global requirements into local ones. This is only relevant for characteristics that are not already marked as *local* (cp. Section 3). Numeric requirements can be transformed, if the aggregation function is *minimum* resp. *maximum* and the specified requirement is the minimum resp. maximum. Consider, e.g., a process with the global requirement that the number of (redundant) network connections has to be at least 2 in order to strengthen reliability. As the aggregation function is *minimum* and the requirement is minimum, it can be replaced by local requirements for each activity demanding at least two network connections. Ordered non-numeric characteristics can be treated in the same way.

Unordered non-numeric characteristics can be used only for requirements with the structure *characteristic* = *value* or its negation. When specified for a process, this means that the re-

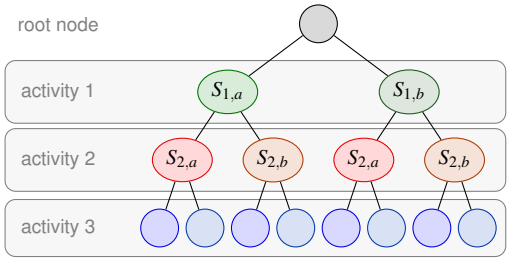

Figure 2: Tree structure used for a heuristic-driven search

quirement has to be fulfilled permanently. Therefore, such a global requirement can directly be evaluated on the local level for each activity.

In this step, each service's NFC offers have to be compared with the local requirements and the service can finally be removed from the set of suitable services if it is not adequate. Accordingly, the complexity of this step is in $O(n)$, where n is the number of services.

Evaluating global requirements The second step analyzes the remaining global requirements. As shown in Section 2.1, this problem has a complexity in $O(n^m)$. Therefore, we propose to use a heuristic which can find a suitable solution more efficiently. However, the basic data structure is a tree – equal to the one used by [JMG05] (cp. Section 2.1) – which expands every possible combination of services (cp. Figure 2). To avoid traversing the whole tree, parts of it are purged if they are detected not to be able to fulfill the requirements anymore. In addition, a heuristic is used to firstly select paths which are estimated to offer better results.

Beginning at the first level of the tree, the best suited node is selected consecutively on every level. The decision is taken with means of a heuristic, which is described in detail later. Thus, for every activity one service is chosen successively. A check at every level ensures that all non-functional requirements are satisfied so far. If not, the part of the tree beginning at the current node is purged and the algorithm proceeds with the next node.

Conditional branches can be realized in three variants: Firstly, for every possible path, a distinct solution is computed. However, this results in a higher effort for processes with many branches. Secondly, the different paths are integrated into the search tree and only one valid solution is found. This variant does not increase the effort significantly, but it fails when the process engine chooses a path which does not correspond to the solution. The algorithm has then to be started again. Thirdly, the introduced algorithm is paused when a conditional branch is reached until the condition can finally be evaluated. In order to evaluate global requirements, estimated values are used for characteristics on the basis of the average of the possible paths respectively of the critical path.

However, while traversing the tree, a heuristic selects one of the available services at the current level of the tree. To achieve good results, a heuristic needs knowledge about the application context. Yu et al. assume for WS_HEU [YZL07] that NFCs are stochastically independent and uniformly distributed. However, real scenarios do not show these properties, e.g. availability, throughput and quality are often closely tied to the price and typical values for availability are above 90 %. In this work, it is assumed that services providing comparable functionality exhibit akin ranges of values for their non-functional characteristics. Here, similarity is justified due to

```
ServiceSelection(REQ, A)
    LocalSelection(REQ, A)
    solution = GlobalSelection(REQ, A)
    if(solution!=FAIL)
        solution = Optimization(REQ, A, solution)
    return solution

LocalSelection(REQ, A)
    for each (act ∈ A)
        for each (s ∈ act.getServices())
            for each (qos ∈ s.getQoS())
                if(! isFulfilled(REQ, qos, s))
                    A.removeService(s)

GlobalSelection(REQ, A)
    global REQ, P, S
    tree=initialize(tree)
    return AnalyzeNode(tree.root)

AnalyzeNode(node)
    if(isLeaf(node))
        if(checkFullConst(REQ,node))
            return node
        else
            return Backtracking()
    else
        if(checkConst(REQ,node))
            return DescentTree(node)
        else
            return Backtracking()
```

```
DescentTree(node)
    Boolean goodNodeFound = false;
    for each(ws ∈ node.childs())
        Boolean fulfilled = true;
        Double score = 0
        for each(qos ∈ ws.getQoS)
            if(isFulfilled(REQ, qos, ws))
                score=1∗qos.weight
            else
                fulfilled=false;
        if(fulfilled)
            P.push(ws)
            goodNodeFound=true;
        else
            insertList(ws,score,S)

    if(goodNodeFound)
        return AnalyzeNode(P.pop())
    else
        return AnalyzeNode(S.first())



Backtracking()
    if(P.size>0)
        return AnalyzeNode(P.pop())
    else if(S.size>0)
        return AnalyzeNode(S.first())
    else
        return FAIL
```

Figure 3: Pseudocode of the algorithm for an NFC-aware service selection

the fact, that functional properties and local requirements for non-functional characteristics have been already detected to be suitable according to the user's demands.

Hence, to estimate the characteristic's value for an activity that has not been analyzed so far, the average of values for the according services can be used. This is uniquely done for each activity. Now, it can be checked if the path going down from the currently selected node fulfills the requirements. If so, the node is put on a stack of promising nodes P. However, if at least one requirement is not satisfied, there might be nevertheless a valid path starting from this node. The approximation of characteristics causes an uncertainty which will be inspected if none of the promising nodes leads to a valid solution. Therefore, an ordered list of second-choice nodes S collects nodes not fulfilling all the requirements, ordered by the weighted number of satisfied requirements. The weight is defined by the user in order to achieve a uniform representation of the characteristics for the case when a characteristic depends on multiple other characteristics, e.g. the price is dependend on availability and throughput.

With this approach, promising paths are investigated as long as a requirement is violated. The algorithm now uses backtracking to start again from the next promising node in the stack P or – if the stack is empty – the best node from the list of second-choices S.

The complexity of this step is in $O(m \cdot n)$ to calculate the average values for each service, where m is the number of activities and n the number of services per activity. The backtracking algorithm is in the worst case indeed in $O(n^m)$, but in real scenarios, a valid solution could be found very early due to the depth-first search. In the best case, one path is used to traverse the tree without reverting to backtracking. Moreover, in mobile environments, n and m are typically not very high, which leads to a good performance of the algorithm.

Optimizing the solution The second step of the algorithm is able to find a valid solution very fast. Allthough this solution satisfies all the user's requirements, it is not likely to be the best. In the third step, the found solution can optionally be optimized with respect to a user-defined weight function $w_j(x)$ which weights the values of non-functional characteristics c_j. Therefore, the *benefit* of a service $s_{i,k}$ implementing an activity a_i is measured as $b_{i,k} = \sum_{x=1}^{n} w_x(Q(c_x, s_{i,k}))$. The optimization tries successively for each acitivity to find another service with a higher benefit. If found, the selected one is replaced by the new service.

The complexity for this step is in $O(m \cdot n)$ since every service has to be investigated exactly once. The weight function has to be supplied by the user and should implement a normalization of the values as well. With this approach, a rudimentary optimization is achieved which guarantees a fast execution time.

Figure 3 shows a simplified variant of the introduced algorithm for selecting services for a set of activities A with respect to a list of requirements REQ. The main procedure *ServiceSelection* initiates the three parts *LocalSelection*, *GlobalSelection* and *Optimization* (not shown). *Analyze-Node* checks if the current solution is valid or initiates the *Backtracking* otherwise. Considering the two procedures for checking statements, *checkConst* omits the evaluation of characteristics using aggregation functions which cannot be evaluated on a subset. Therefore, *checkFullConst* finally evaluates all requirements. *DescentTree* uses the heuristic approach to decide which path has to be evaluated firstly.

5 Conclusion and Future Work

Complementing existing approaches on the specification of quality contraints and non-functional aspects as well as on service selection, this paper presents an advanced description language to specifiy NFCs in a way which supports the usage of strategies in order to descrease complexity as well as a respective low-complex service selection algorithm. Both the description language and the algorithm are currently implemented within a prototype of the *DEMAC process execution engine* for mobile workflows (cp. [KZL06]) which was formerly using an insufficient key-value approach to specify NFC offers and requirements. First tests have shown the applicability and effectiveness of the approach. Ongoing work includes research on further description mechanisms which enable the integration of application-specific knowledge about the probable execution of the process in order to estimate the control flow with respect to branches and cycles.

Acknowledgements: The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

Bibliography

[Aag01] J. O. Aagedal. *Quality of Service Support in Development of Distributed Systems*. PhD thesis, University of Oslo, 2001.

[AP06] D. Ardagna, B. Pernici. Global and Local QoS Guarantee in Web Service Selection. In *Business Process Management Workshops*. Pp. 32–46. Springer, 2006.

[BSR+06] R. Berbner, M. Spahn, N. Repp, O. Heckmann, R. Steinmetz. Heuristics for QoS-aware Web Service Composition. In *Int. Conf. on Web Services*. Pp. 72 –82. 2006.

[DLS05] G. Dobson, R. Lock, I. Sommerville. QoSOnt: A QoS Ontology for Service-centric Systems. In *31st EUROMICRO Conference on Software Engineering and Advanced Applications*. Pp. 80–87. 2005.

[HHGR06] G. Hackmann, M. Haitjema, C. D. Gill, G.-C. Roman. Sliver: A BPEL Workflow Process Execution Engine for Mobile Devices. In *Int. Conf. on Service-Oriented Computing (ICSOC 2006)*. Pp. 503–508. Springer, 2006.

[JMG05] M. C. Jaeger, G. Mhl, S. Golze. QoS-Aware Composition of Web Services: An Evaluation of Selection Algorithms. In *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*. 2005.

[KZL06] C. P. Kunze, S. Zaplata, W. Lamersdorf. Mobile Process Description and Execution. In *Proc. of the 6th Int. Conf. on Distributed Applications and Interoperable Systems (DAIS 2006)*. Pp. 32–47. Springer, 2006.

[LKD+03] H. Ludwig, A. Keller, A. Dan, R. P. King, R. Franck. Web Service Level Agreement (WSLA) Language Specification. Technical report, IBM Corporation, 2003.

[PTDL08] M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann. Service-Oriented Computing: a Research Roadmap. *Int. J. Cooperative Inf. Syst.* 17(2):223–255, 2008.

[RCJ02] N. Rosa, P. Cunha, G. Justo. ProcessNFL: A Language for Describing Non-functional Properties. In *35th Annual Hawaii International Conference on System Sciences (HICSS'02)*. Volume 9. IEEE Computer Society, 2002.

[SDM01] A. Sahai, A. Durante, V. Machiraju. Towards Automated SLA Management for Web Services. Technical report, HP Laboratories, 2001.

[TPP02] V. Tosic, B. Pagurek, K. Patel. WSOL – A Language for the Formal Specification of Various Constraints and Classes of Service for Web Services. Technical report, Carleton University, Ottawa, Canada, 2002.

[VOH+06] A. S. Vedamuthu, D. Orchard, M. Hondo, T. Boubez, P. Yendluri. Web Services Policy 1.5 – Primer. Technical report, W3C, 2006.

[YL05] T. Yu, K.-J. Lin. Service Selection Algorithms for Web Services with End-to-End QoS Constraints. *Inf Syst E-Bus Manage* 3(2):103–126, 2005.

[YZL07] T. Yu, Y. Zhang, K.-J. Lin. Efficient algorithms for Web services selection with end-to-end QoS constraints. *ACM Trans. Web* 1, May 2007.