AUTOMATED PLANNING OF CONTEXT-AWARE PROCESS MODELS

Complete Research

Heinrich, Bernd, University of Regensburg, Regensburg, Germany, bernd.heinrich@wiwi.uni-regensburg.de

Schön, Dominik, University of Regensburg, Regensburg, Germany, dominik.schoen@wiwi.uni-regensburg.de

Abstract

Most real world processes are heavily influenced by environmental factors, which are referred to as the context of a process. Thus, the consideration of context is proposed within the research field of Business Process Modeling. Most existing context-aware modeling approaches consider context only in terms of static information like, for instance, the location where a process is performed. However, context information like the weather could change during the conduction of a process, which we will denote as non-static context. In order to increase the flexibility concerning environmental influences in general and especially context-related events of context-aware processes, we present an approach for the automated planning of context-aware process models that considers static and non-static context. We therefore propose an extended state transition system in order to represent context information in terms of context variables and consider process exogenous changes of these context variables through context signals and receive context actions. Further, to ensure a correct, complete and time efficient construction of context-aware process models, a planning approach is used to support modelers by means of an algorithm. To demonstrate the feasibility of our approach we mathematically evaluated the algorithm and applied it to real world processes.

Keywords: Context Awareness, Business Process Modeling, Automated Planning

1 Introduction

Most real world processes are heavily influenced by environmental factors such as context-specific information (Hu *et al.*, 2012; Soffer, 2005; Wang *et al.*, 2012; Zhou *et al.*, 2011). Hence, to appropriately react on such influences, processes have to be context-aware (Gottschalk *et al.*, 2007; Gottschalk and La Rosa, 2010; La Rosa *et al.*, 2011; Reichert and Weber, 2012; Swenson, 2010; van der Aalst *et al.*, 2006). As Abowd *et al.* (1999) and Dey (2001) define context as "any information that can be used to characterize the situation of an entity" and "an entity is a person, place, or object that is considered relevant" within a process, we call processes that are adaptive to environmental influences context-aware. Especially emerging modern technologies like smartphones, wearable sensors and mobile devices in general (Baldauf *et al.*, 2007; Zhang *et al.*, 2009) enable to gather detailed information about the current context of an entity and especially of the performer of a process. Context information like the current location, for instance, could easily be gathered by using sensor technology, which is present in almost every modern smartphone. Thus, the research field of context-aware processes of both, organizations and private persons, becomes even more relevant especially with respect to mobile processes and pervasive computing (Bettini *et al.*, 2010; Satyanarayanan, 2001; Ye *et al.*, 2012). As process models are an established method to represent processes, the consideration of context-specific information in process models in order to increase the flexibility of context-aware real world processes is also discussed (e.g. Schonenberg *et al.*, 2008).

To address context-aware process models, several approaches (cf. e.g., Bucchiarone *et al.*, 2013; Hallerbach *et al.*, 2010, 2008; Rosemann *et al.*, 2010; Rosemann and van der Aalst, 2007; Tealeb *et al.*, 2014; van der Aalst *et al.*, 2006) have been presented in the last years. All of those approaches consider context by means of appropriate configurations or variants of a so called master process model or segments of a process model. Each of these configurations or variants is created before starting the process execution (i.e., at design time) and is valid in a specific context. Then, these approaches select an appropriate configuration or variant from a process repository based on the current context during the execution of the process (i.e., at runtime). A few other authors follow a different approach (cf. e.g., Ayora *et al.*, 2013; Sakurai *et al.*, 2012) and consider context information only during runtime of a process. They do not consider any context information at design time and within the process model. The (partial) consideration of context during runtime usually requires the performer of the process to provide a broad knowledge and a high level of expertise, which leads to restrictions especially regarding complex processes. Moreover, it could result in an increased process execution time as the consideration of context information during runtime costs time (cf. e.g., Fujii and Suda, 2009).

Summing up, current approaches have already strongly contributed to the current knowledge about context-aware processes. But most of them consider context in terms of different but static environments like different locations in which a process can be performed. However, occurring environmental events (i.e., context events) could change the context of a process *throughout its conduction*, which we will denote as "*non-static context*". A simple, yet common example is an upcoming thunderstorm during the conduction of an outdoor process. Here, a process that has already started to be conducted or a selected process variant may become inadequate due to a context event. For example, subsequent actions could no longer be performed (e.g., a flight could not depart due to a storm) or already running actions might be terminated (e.g., an emergency landing is required because of a technical issue). In the worst case, the conduction of the complete process must be terminated, as the desired process goal could no longer be reached. Therefore, it is inevitable to consider *non-static context* throughout a process model (Dobson *et al.*, 2006). This issue is addressed only by few authors at all, which, however, use rather complex process reconfiguration tasks that need to take place during runtime (cf. e.g., Hallerbach *et al.*, 2008) and therefore may delay the process execution. Thus, the consideration already during design time and hence the reduction of complexity during runtime is useful. However, to the best of our knowledge, there is no approach, which considers *non-static context* during design time within process models so that no further, complex reconfiguration tasks need to be performed during runtime to enable a consideration of context. We therefore want to address this research question and tend to construct a *comprehensive* process model, which does not have to be reconfigured or adapted during runtime in order to consider *non-static context*.

To allow us a correct, complete and fast construction of comprehensive context-aware process models the second research question arises of how this construction can be supported through a planning approach (e.g., algorithms). This question follows several approaches to support modelers and business analysts by means of automation (e.g., algorithms) in the last years. For instance, Process Mining (e.g. van der Aalst *et al.*, 2004; van der Aalst *et al.*, 2012), especially automated process discovery, assists business analysts in the *process analysis* phase of the Business Process Management (BPM) Lifecycle (cf., Wetzstein *et al.*, 2007). Automated service selection and composition (cf. e.g., Heinrich *et al.*, 2015; Khan *et al.*, 2010; Weber, 2007) increase the degree of automation within the phases *process implementation* and *process execution*. The construction of process models in an automated way is addressed by process planning algorithms (Heinrich *et al.*, 2012; Henneberger *et al.*, 2008; Hoffmann *et al.*, 2012, 2009) to support modelers in the phase of *process modeling*. To follow such approaches, we aim to present an approach for the *automated planning* of context-aware process models (second research question) that can cope with *non-static context* (first research question) in this paper. The main contributions are as follows:

❶ We aim to construct comprehensive context-aware process models by considering *non-static* context in terms of context variables. Therefore, we introduce a formal definition of a *planning domain* that can cope with *non-static context* throughout a process model. To construct complex process models, which are able to cope with occurring environmental events in terms of context

events (resulting in context signals), we take into account that the context of the process might change while it is performed later on.

❷ To address the issue of constructing context-aware process models in an automated way, we present a novel *planning algorithm*. In order to abstract from an individual process execution (cf., Ghallab *et al.*, 2004) and to construct entire process models, we consider belief states instead of world states. Further, to enable a widespread use and acceptance of our approach we present a nondeterministic state transition system that is independent from a specific process modeling language as the formal basis for our algorithm.

2 Background

In the following, we will discuss existing approaches on context-aware BPM and related topics like modeling exceptional flows and signals. Further, we want to summarize related work within the fields of context representation in general and well-known process modeling languages in particular.

An increasing amount of research has been conducted on context-aware BPM. Rosemann *et al.* (2008) state that context-awareness is highly relevant in the field of business process modeling. Rosemann *et al.* (2010) further state that the research has "increased attention to flexibility" which results from "the trend toward decreasing time-to-market" and "increasing frequency of product innovation" and therefore a demand for process flexibility with regard to their environment (Soffer, 2005) emerges. Rosemann *et al.* understand process flexibility as "the ability to change the process [during runtime] without completely replacing it" (Bider, 2005; Regev *et al.*, 2007). Hallerbach *et al.* (2010) follow a slightly different approach by defining so called process variants. By means of process variants, they assign different process models to different contexts. Further, during runtime, they reconfigure the process for the current context by applying rather complex reconfiguration tasks, defined alongside with the process variants. Basically, their approach could be understood as a selection of a specific process model from a large repository of regular (not context-aware) process models and a reconfiguration of the currently conducted process model in case of changing context information. As both approaches need to change parts of the process during runtime they do not consider a comprehensive process model that takes context into account as part of the regular control flow, which would be favorable with respect to an automated process execution (cf. e.g., Khan *et al.*, 2010; Weber, 2007).

Zhu *et al.* (2014a) and Zhu *et al.* (2014b) strive the problem of location dependency, which is a subproblem of context-awareness, by means of "location-dependent process model patterns" (e.g., a location dependent exclusive choice). However, these location-dependent process model patterns do not significantly differ from regular control flow patterns (van der Aalst *et al.*, 2003). For example, a "location-dependent exclusive split" (i.e., an exclusive choice) describes a decision based on location information and hence is basically a particular type of a regular exclusive choice. Further, they do not address other context information and the challenge of non-static context (Dobson *et al.*, 2006).

Mattos *et al.* (2014) propose a metamodel for context-aware process models but do not address how to construct and represent (in terms of a notation) context-aware process models. Further, they state that well-known modeling languages like EPC, UML or BPMN "do not include the concept of context" but also leave this issue unsolved. Within their metamodel they define *Contextual Entities* (persons, places, objects, etc.) which are considered in terms of *Rules* that must be met in order to execute an action but they do not cope with the problem of integrating context into process models as they do not propose an approach to represent context and consider it in terms of a modeling language.

However, there exist well-known approaches striving related challenges like exception and signal handling in process models. Russell *et al.* (2006) present five different exception types that could be determined throughout a process. Amongst others, *External Triggers* represent "[t]riggers from sources external to a work item [that] are often used as a means of signaling the occurrence of an event that impacts on the work item and requires some form of handling. These triggers are typically initiated […] from processes in the operational environment, in which the PAIS [i.e., Process Aware Information System] resides" (Russell *et al.*, 2006). Thus, environmental events in terms of context signals

could be classified as *External Triggers*. Here, Russell *et al.* (2006) focus on recovery and termination strategies in order to enable a graceful continuation or termination of the process. For example, they consider rollback strategies and reoffering the interrupted action later. Thus, as we want to design process models that are adaptive to changing contexts, their findings are not applicable for our needs.

Besides the consideration of exception handling in terms of recovery approaches, UML and BPMN contain approaches to denote exceptional flows, which may also be used to represent context signals within process models. BPMN (since version 2.0) distinguishes between erroneous situations that lead to an interruption of an action (*error events*) and so called *escalation events* that may be non-interrupting (Object Management Group, 2014). Context signals may influence an action in a non-interrupting and an interrupting manner, too, and thus both cases should be considered here, as well. Furthermore, UML (cf., Object Management Group, 2013) contains so called *Exception Handlers* that, in case of an exception, gracefully handle the exception that occurred during a performed action (i.e., interrupting). Further, so-called *Interruptible Activity Regions* with *interruptible edges* denote areas in a process model (instead of single actions) that could be interrupted due to an exception. Besides, exogenous influences (i.e., non-interrupting) are denoted in terms of *signals*. Incoming signals (e.g., events initiated by another process) are handled by so called *Accept Event Actions* that initiate a further regular control flow within the process. "If [an] accept event action is executed and object detected event matching one of the triggers [i.e., signals] on the action, then the accept event action outputs a value describing the event. If the event does not match expected event, the action waits for the next event." In contrast to the representation of exceptions, events and signals, the representation of specific *context variables* and their consideration in process models receives only little discussion so far (cf., Zhu *et al.*, 2014a). Process modeling languages like BPMN, UML or EPC do not take context variables explicitly into account. In BPMN, for example, the context could be modeled by means of swimlanes, annotations or data (Object Management Group, 2014). Such a consideration in annotations or data attributes is limiting, as context then could not be considered as a variable during process execution (cf. e.g., Mulholland *et al.*, 2006) or by means of the control flow (e.g., decisions in terms of an exclusive choice are not feasible on annotations). Further, in languages like BPMN and UML (cf. e.g., Object Management Group, 2013, 2014) no modeling elements for context signals or context variables in particular exist, even though they strive related issues. To summarize, the discussed languages provide a basis to cope with contribution ❶ as they address related issues. However they do not strive contribution ❷, the automated construction of context-aware process models.

As modeling context information is the foundation for our work, we will review current approaches for *context representation* in the following as well. Strang and Linnhoff-Popien (2004) and Bettini *et al.* (2010) reviewed the most relevant approaches (according their appraisal) for modeling context and classified them in several groups. Especially the identified Ontology Based Models are closely related to the research strand of process planning (cf., Section 3). Further, ontology based context modeling has been proposed due to its advantages by means of normalization and formality (Öztürk and Aamodt, 1997). We will therefore use ontology based context modeling as a further basis to cope with contribution ❶ within our approach. Basically all approaches of this group (cf. e.g., Attard *et al.*, 2013; Chen *et al.*, 2003; Nadoveza and Kiritsis, 2014; Simons and Wirtz, 2007) have in common, that context is represented as a set of atomic or composite (consisting of atomic or composite variables again) context variables with a specific domain. Simons and Wirtz (2007), for example, introduce a UML profile for context modeling that could be used as a comprehensive metamodel for context modeling. Chen *et al.* (2003) and Wang *et al.* (2004) introduce context ontologies expressed in the Web Ontology Language (OWL), which is heavily used within the research fields of BPM and automated compositions of (web) services. However, none of these works addresses the issue of planning context-aware process models and thus contribution ❷, as they focus on the representation of context.

3 Planning Domain and Running Example

As already stated, several approaches to support modelers and business analysts by means of process automation have been proposed in the last years. As part of this development, the construction of con-

text-aware process models in an automated way seems promising especially as context-aware process models are usually more complex than non-context-aware process models. The automated construction of process models can be understood as a planning problem (e.g. Heinrich *et al.*, 2009). More precisely, we have to abstract from an individual process execution and its world states in order to construct entire process models, valid for various process executions, resulting in a nondeterministic planning problem with belief states (Ghallab *et al.*, 2004). Here, a belief state represents possibly infinite sets of world states. Hence, to enable a widespread use and acceptance of our approach to construct context-aware process models, we use a general set-theoretic planning domain (cf., Ghallab *et al.*, 2004) as a foundation and starting point for our approach. This further guarantees a maximum of compatibility with existing approaches in the literature (e.g. Bertoli *et al.*, 2006; Bertoli *et al.*, 2001; Heinrich *et al.*, 2009; Meyer and Weske, 2006; Sycara *et al.*, 2003). Considering this planning domain, a *planning graph,* which basically consists of two types of nodes - representing *belief states* and *actions* - and edges is used. As a belief state $bs \subseteq BST$ could be seen as a set of information about the variables currently available in a process state (so called belief state variables) we denote a belief state as a set of *belief state tuples*, whereas each of them denotes one particular characteristic:

Definition 1 *(belief state). BS* is a finite set of belief states. A belief state $bs \in BS$ contains a set *BST* of belief state tuples. Here, every belief state tuple *p* exists one time at the most. A belief state tuple $p \in BST$ is a tuple of a belief state variable *v(p)* and a subset *r(p)* of its predefined domain *dom(p)*, which we will write as $p:=(v(p),r(p))$.

To represent *actions* conducted by a so called performer, a second type of node is defined. An action *a* is a triple $a=(name(a), pre(a), eff(a))$ whereas $pre(a) \subseteq BST$ denotes all conditions when *a* could be applied and $eff(a) \subseteq BST$ denotes all changes that result from the conduction of *a*.

Definition 2 *(action). A* is a finite set of actions. Each action $a \in A$ is a triple consisting of the action name and two sets, which we will write as $a:=(name(a), pre(a), eff(a))$. The set $pre(a) \subseteq BST$ are the preconditions of *a* and the set $eff(a) \subseteq BST$ are the effects of *a*.

In a next step, we define a planning graph as follows:

Definition 3 (*planning graph*). A planning graph is an acyclic, bipartite, directed graph $G=(N, E)$, with the set of nodes *N* and the set of edges *E*. Henceforth, the set of nodes *N* consists of two partitions: First, the set of action nodes $Part_A$ (set *A* of actions) and second the set of belief state nodes $Part_{BS}$ (set *BS* of belief states). Each node $bs \in Part_{BS}$ is representing one distinct belief state in the planning graph. The planning graph starts with one initial belief state and ends with one to probably many goal belief states (with $Init \in BS$ and $Goal_j \in BS$).

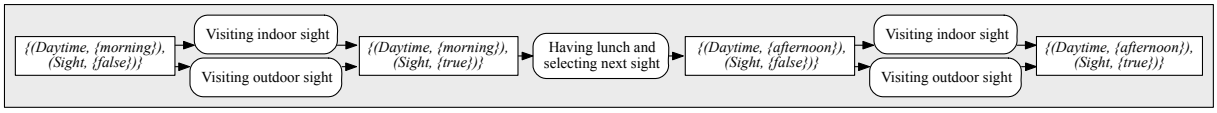


Figure 1. Initial planning graph of the running example.

To illustrate our approach and the planning domain, we will use a simplified excerpt of a real world tourism process, which guides a tourist during a day trip to a foreign city and is highly influenced by context information. The regular, non-context-aware process is represented by the planning graph shown in Figure 1. Here, the initial belief state (leftmost belief state; belief states denoted as rectangles with italic text) represents the start of a tourist's day trip in the morning. Thereby, in the morning the tourist could perform the actions *Visiting indoor sight* or *Visiting outdoor sight* (actions denoted as rounded rectangles with regular text). Afterwards s/he selects the next sight to visit while having lunch and visits an outdoor or indoor sight in the afternoon again. After performing one of those two actions, s/he finishes the process by reaching the goal belief state (rightmost belief state).

4 Approach to Plan Context-Aware Process Models

To enable the automated planning of context-aware process models (cf. contributions ❶ and ❷ discussed in the introduction) we divide this overall goal into the following sub goals:

- *Set theoretic representation of context.* To consider context information in our planning domain, we need to adapt the above definitions. In particular, we have to extend the definition of belief states to denote context variables in terms of a set theoretic representation.
- *Consider context as non-static.* As context should be considered as non-static throughout a process (Dobson *et al.*, 2006), it is required to take occurring process exogenous changes of context into account. Therefore, it is needed to extend the set theoretic state transition system by means of context signals that accord to those occurring process exogenous changes. In detail, to address contribution ❷, we need to automatically construct receive context actions (i.e., by means of an algorithm) that represent process exogenous changes of context (i.e., the transition) taking place during the conduction of a (regular) action in the process models.
- *Align notation to well-known modeling languages.* To ensure a maximum of compatibility with existing process modeling approaches and to increase the acceptance of the constructed process models among modelers, we need to consider existing notations of well-known process modeling languages like UML. Thus, in order to represent process exogenous context changes we need to use well-known modeling notation elements for asynchronous events or signals, for instance.

4.1 Consider context within the planning domain

As a first step, we need to represent context variables in terms of our planning domain. As said above, context is supposed to be represented by means of ontology based models (Bettini *et al.*, 2010; Öztürk and Aamodt, 1997; Strang and Linnhoff-Popien, 2004). Thus, we represent the context of a process as a set of so called context variables, a subset of belief state variables that allow us to denote context information and their predefined domains. As context variables represent process exogenous information and regular belief state variables represent process endogenous information, we define those two set as disjoint. A belief state tuple $p \in BST$ therefore is either contained in the set $CT \subseteq BST$ of so called context tuples or in the set $BST \setminus CT$ of regular belief state tuples.

Therefore, we adapt the previous definitions of belief states and actions as follows:

Definition 1' *(belief state).* BS is a finite set of belief states. A belief state $bs \in BS$ contains a set BST of belief state tuples. A belief state tuple $p \in BST$ is either assigned to the set of context tuples $CT \subseteq BST$ or to the set of regular belief state tuples $BST \setminus CT$ and is, in any case, a tuple of a belief state variable $v(p)$ and a subset $r(p)$ of its predefined domain $dom(p)$, which we will write as $p:=(v(p),r(p))$.

Definition 2' *(action).* A is a finite set of actions. Each action $a \in A$ is a triple consisting of the action name and two sets, which we will write as $a:=(name(a), pre(a), eff(a))$. The set $pre(a) \subseteq BST$ are the preconditions that must be met to apply action a. The set $eff(a) \subseteq BST \setminus CT$ are the effects that result from the conduction of a.

By using context tuples in the preconditions of actions, we create a basis for considering context information throughout a process model. However as Definition 2' also outlines, the conduction of (regular) actions a cannot change context variables due to their process exogenous character.

Both definitions are a first step that allows us to take context into account in a static manner. For instance, within our running example, it may be favorable to plan the actions *Visiting indoor sight* and *Visiting outdoor sight* depending on the weather, as a performer may prefer an indoor activity like a museum to an outdoor activity in case of bad weather. Such user preferences can be denoted and considered by means of belief state tuples within the initial state of the process model. To denote the fact that visiting an outdoor sight depends on the weather, we include the according context tuple *(Rain, {false})* in the preconditions of the corresponding action. By denoting context information in terms of context variables we are now able to consider static context information throughout the process.

In order to take different possible contexts in which a planned process model can be performed into account, we need to identify relevant context variables that could influence the process execution. As each influencing context variable is part of the preconditions of at least one action, the identification is quite straightforward as we need to consider all context variables that are contained in any action's preconditions. Thus, in order to be able to determine if an action can be planned (i.e., is applicable) within a particular context, that is, if all context-related preconditions of that action hold, context variables must be included from the beginning to the end of the regular process, making this process context-aware in a static manner. Moreover, it is needed to include all relevant context variables (i.e., all context variables present in any actions preconditions) within the initial belief state of the process, as these context variables cannot be changed by regular actions (cf., Definition 2').

A comprehensive coverage of possible contexts, in which a process can be performed, requires the consideration of all context variables, contained in the preconditions of any action $a \in A$. Thereby, all possible contexts can be represented in terms of permutations of context variables. Therefore we include the following context tuples $(v(p),r(p)) \in CT$ to the initial states of the state transition system:

$\{(v(p),r(p)) \in CT \mid r(p) \subseteq dom(p), \exists (v(q),r(q)) \in pre(a), v(q)=v(p), a \in A, q \in CT\}$.

By means of this extension, our running example can be extended as seen in Figure 2.

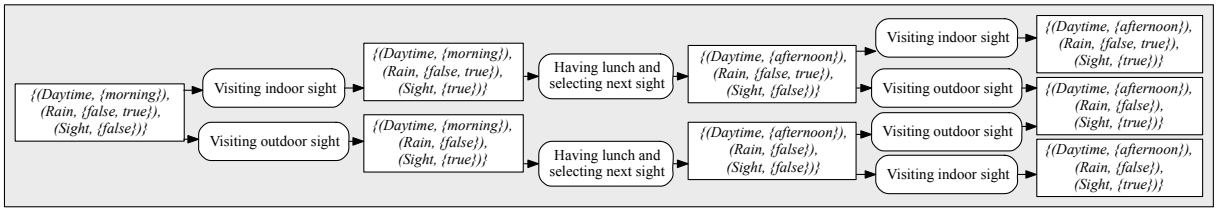


Figure 2. Extended planning graph including static context information.

4.2 Consider non-static context within the planning of process models

So far, context is considered as static throughout the process, whereas in real world processes changes to context variables could occur when conducting the process due to environmental influences (Dobson *et al.*, 2006). For example, it could start raining while conducting the action *Visiting outdoor sight* and thus, the performer may be required to react on this dynamic context change. Thus, we have to address how to cope with non-static context throughout the process in the following.

To consider process exogenous changes of the values of context variables caused by nature or process-external agents but without the involvement of a performer of the process, we need to model this exogenously initiated state transition. We denote a specific event, which may occur dynamically throughout the process as a *context signal*. As a context signal represents an occurred event or state, we denote a context signal $sig \in SIG$ (with SIG as the set of context signals) as a set of context-related belief state tuples $sig:=\{(v(p),r(p)) \mid (v(p),r(p)) \in CT\}$. For example, the context signal *rain starts falling:= {(Rain, {true})}* denotes, that the value of the context variable *Rain* changed to *true* due to a process exogenous event. We define a context signal as follows:

Definition 4 *(context signal).* SIG is a finite set of context signals. A context signal $sig \in SIG$ contains a set of context tuples with $(v(p),r(p)) \in sig \subseteq CT$.

Besides the state representation, we have to cope with the state transition, which means, we need a model element that depicts the according transition of context variables (i.e., the value of the context variable *Rain* changes from *false* to *true*). Thus, we define a *receive context action* $c_{sig} \in C$ (with C as the set of receive context actions) similar to a regular action (in terms of preconditions and effects) but distinct the sets of receive context actions and regular actions. This is due to the fact that regular actions need to be executed by a performer and could not influence context variables while receive context actions are triggered by nature or (process-external) agents who are not conducting the process. Thus, we define a receive context action as follows:

Definition 5 *(receive context action). C* is a finite set of receive context actions. A receive context action $c_{sig} \in C$ – related to a context signal *sig* – is defined as a triple consisting of the name of the context signal that is handled by this receive context action and two sets, which we will write as $c_{sig}:=(name(c_{sig}), pre(c_{sig}), eff(c_{sig}))$. The first set $pre(c_{sig}) \subseteq BST$ are the preconditions of c_{sig} and the second set $eff(c_{sig})=sig \subseteq CT$ are the effects of c_{sig}.

Context signals are process exogenous and thus independent of regular belief state variables representing information created within the process. Therefore, within the effects of receive context actions only context variables are considered. However, as process endogenous information may be used to determine if a receive context action is applicable, regular belief state variables may be used within the preconditions of receive context actions. Within our running example the receive context action for the context signal *rain starts falling*, for instance, needs to be considered only if the performer selects an outdoor sight (denoted by the belief state variable *NextSight*) and we therefore include *(NextSight, {outd.})* in the preconditions of the according receive context action.

Considering our planning domain, the determination whether an action is applicable in a belief state is based on the according preconditions and the belief state itself. If the preconditions of an action hold in a belief state, the action is applicable, if the preconditions do not hold, the action is not applicable. Based on this, context signals that may influence the execution of an action are identified by analyzing the preconditions of this action.

Following this, an action would be interrupted if the preconditions of a (regular) planned action no longer hold (due to an occurred context signal). For example, within our running example the action *Visiting outdoor sight* will be interrupted by the mentioned context signal *rain starts falling* ($eff(c_{rain\ starts\ falling}) = \{(Rain, \{true\})\}$) as $(Rain,\{false\}) \in pre(Visiting\ outdoor\ sight)$ and $true \notin \{false\}$ which is the intuitive way to denote that the planned action must not be performed in case of bad weather. Thus, formally written, a planned action *a* must be interrupted by a context signal *sig* if and only if $\exists\ (v(p),r(p)) \in sig : \exists\ (v(q),r(q)) \in pre(a),\ v(q)=v(p),\ r(p) \cap r(q)=\emptyset$.

Furthermore, context signals may also be relevant for actions that are not necessarily interrupted by the context signal. For example the context signal *rain stops falling* may occur during *Visiting indoor sight* and thus a performer may continue this action as sunny weather does not prevent him from visiting an indoor sight while another performer might prefer *Visiting outdoor sight* to *Visiting indoor sight* if the rain stops falling. This means that such a context signal may lead to an interruption under certain circumstances but does not necessarily interrupt the considered action. Hence, the further process is conditionally influenced by the context signal. In terms of our planning domain, this means, that the restrictions of at least one context tuple of the considered context signal overlap with those of the preconditions of the planned action. Regarding our example, $r(Rain) = dom(Rain)$ is present in the preconditions of the action *Visiting indoor sight* and thus, the action is applicable regardless of the weather. However, it may be interrupted if a performer prefers outdoor activities. Based on such user preferences it can be decided whether to continue or to interrupt the conduction of the action based on its preconditions. In terms of our planning domain, we distinct both cases due to the fact, that the planned action is applicable again in the belief state after the occurrence of a context signal.

Based on the Definitions 1' to 5, we are now able to define a nondeterministic context-aware state transition system as follows:

Definition 6 (*Nondeterministic context-aware state transition system*). A nondeterministic context-aware belief state transition system is a tuple $\Sigma = (BS, A, C, R)$, where

- *BS* is a finite set of belief states. A belief state $bs \in BS$ contains a set *BST* of belief state tuples. Here every belief state tuple *p* exists one time at the most. A belief state tuple $p \in BST$ is a tuple of a belief state variable $v(p)$ and a subset $r(p)$ of its predefined domain $dom(p)$, which we will write as $p:=(v(p),r(p))$. The set of context tuples is denoted as $CT \subseteq BST$.
- *A* is a finite set of actions. Each action $a \in A$ is a triple consisting of the action name and two sets, which we will write as $a:=(name(a), pre(a), eff(a))$. The set $pre(a) \subseteq BST$ are the preconditions of *a* and the set $eff(a) \subseteq BST \setminus CT$ are the effects of *a*.

- An action a is *applicable* in a belief state bs if $\forall u \in pre(a)\ \exists\ w \in bs: v(u)=v(w) \wedge (r(u) \cap r(w) \neq \emptyset)$.
- SIG is a finite set of context signals. A context signal $sig \in SIG$ contains a set of context tuples with $(v(p),r(p)) \in sig \subseteq CT$. An according receive context action $c_{sig} \in C$ further is defined as a triple consisting of the name of the context signal handled by this receive context action and two sets, which we will write as $c_{sig} := (name(c_{sig}), pre(c_{sig}), eff(c_{sig}))$. The set $pre(c_{sig}) \subseteq BST$ are the preconditions of c_{sig} and the set $eff(c_{sig})=sig \subseteq CT$ are the effects of c_{sig}.
- The context signal sig that is handled by a receive context action c_{sig} interrupts a planned action a in a belief state bs if $\exists t \in eff(c_{sig}): \exists u \in pre(a), v(t)=v(u) \wedge (\forall x \in pre(c_{sig})\ \exists w \in bs: v(w)=v(x) \wedge ((\nexists y \in pre(a), v(y)=v(x), r(x) \cap r(w) \neq \emptyset) \vee (\exists y \in pre(a), v(y)=v(x), r(x) \cap r(y) \cap r(w) \neq \emptyset))$.
- $R: BS \times A \times C \rightarrow 2^{BS}$ is the transition function. R associates to each belief state $bs \in BS$, each action $a \in A$ and each receive context action $c_{sig} \in C$ the set $R(bs, a, c_{sig}) \subseteq BS$ of next belief states.

The transition function R basically consists of two parts. In the first part, the belief state resulting from the application of an action a is retrieved. Further, in the second part, if a context signal sig interrupts the action a, the according belief state resulting from the interruption is determined taking into account the according receive context action c_{sig}.

More precisely, in the first part the belief state bs prior to a is joined with the preconditions of a while the effects of a are concatenated with bs by means of a set theoretic union of each belief state variable's restriction to retrieve the belief state resulting from the application of a. For example, the belief state variable *Sight* is changed to *true* by the action *Visiting outdoor sight* within our running example.

In the second part, the probable interruption by means of the context signal sig has to be taken into account. Here, it is uncertain (during design time) whether and to which extent the effects of the interrupted action a have to be applied. Thus, within the belief state after the interruption, the value of a belief state variable could either be the value that is defined in the effects of the action a (if the effects are applied) or the value of the variable in the belief state bs prior to a. Within our example, the variable *Sight*, as already stated, is changed to *true* in case the action *Visiting outdoor sight* is applied. As the value is *false* in the belief state prior to this action, we have to unify those two values so that *Sight* may be *false* or *true* in the next belief state after the interruption. This can be referred to as context-related uncertainty (corresponding to the well-known initial state uncertainty). However, when we determine the belief state after the interruption the particular context signal that interrupts the action has to be considered as well. This means, the preconditions and effects of the according receive context action have to be applied to the previously constructed belief state regarding our state transition function in Definition 6. Thus, considering the context signal *rain starts falling*, the value of the context variable *Rain* is *true*, which is the unique representation of the state, as the value of a context variable that is addressed by a context signal is known (i.e., no context-related uncertainty in this case).

After both steps, if the resulting belief state already exists within the planning graph (e.g. in case a loop is created), the further planning for the current path stops here, as the belief state has already been checked for applicable actions.

By means of the nondeterministic context-aware state transition system in Definition 6, we are able to construct a comprehensive process model that considers context signals throughout the process. In order to visualize the resulting planning graph, we extend our running example by the two already mentioned context signals and their according receive context actions:

rain starts falling ($pre(c_{rain\ starts\ falling})=\{(Rain, \{false\}), (NextSight, \{outd.\})\}$, $eff(c_{rain\ starts\ falling})= \{(Rain, \{true\})\}$)

rain stops falling ($pre(c_{rain\ stops\ falling})=\{(Rain, \{true\})\}$, $eff(c_{rain\ stops\ falling})= \{(Rain, \{false\})\}$)

The context signal *rain starts falling*, for example, is relevant during the conduction of the action *Visiting outdoor sight* as the preconditions of this action will no longer hold in case of the occurrence of the context signal. Here, we additionally need the belief state variable *NextSight*, to avoid that the context signal *rain starts falling* interrupts the conduction of the action *Visiting indoor sight* and the according receive context action is planned, as no other action than *Visiting indoor sight* could be con-

ducted in this case. As seen in Figure 3, receive context actions are represented in terms of Accept Event Actions (cf., Object Management Group, 2013) and thus depicted by means of concave pentagon symbols labelled with the name of the according context signals. Further, we denote context signals that are relevant for planned actions by means of Interruptible Activity Regions (cf., Object Management Group, 2013) and thus by means of dashed, light gray areas around these actions. Further, we connect an Interruptible Activity Region to the according receive context action that handles the context signal by means of a dashed edge.

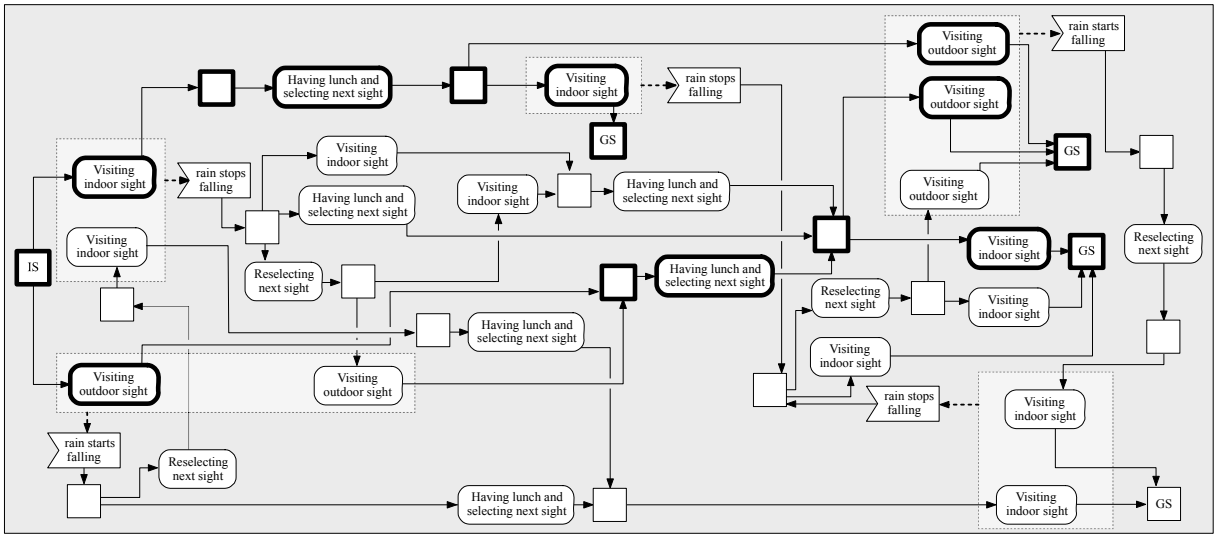


Figure 3. Planning graph including receive context actions and interruptible activity regions.

In Figure 3 we see the process model as seen in Figure 2, denoted in terms of bold actions and bold belief states[1]. Based on Definition 6, for instance, during the conduction of the action *Visiting outdoor sight* (see leftmost area) the context signal *rain starts falling* may occur (see Interruptible Activity Region). If so, the regarding path continues with the according receive context action *rain starts falling* and continues with regular actions. Precisely, in the belief state right after the receive context action *rain starts falling* (see leftmost belief state), the actions *Having lunch and selecting next sight* and *Reselecting next sight* (as the user needs to choose a new sight) are applicable. Continuing this planning, the goal state (labeled with "GS") at the very right of Figure 3 can be reached.

Thus, by constructing a context-aware process model, we are able to consider possibly occurring context signals already during design time. We further enable to reduce complexity during runtime and thus allow performers to execute a process without being mindful of possible context signals.

4.3 Algorithm

Within an algorithm[2] that realizes the presented nondeterministic context-aware state transition system, the planning graph is constructed by means of a depth first search, starting with the initial belief state. Applicable actions are retrieved by means of their preconditions and thus, regarding the state transition function, the following belief states are constructed until a goal state or an already determined belief state is reached. Further, relevant context signals for all planned actions are retrieved (cf., Primitive `isRelevantForActionAndState`; part of class `ContextSignal`; Listing 1).

The retrieval basically consists of two components: First, a context signal that affects the conduction of a planned action is identified, if at least one context variable of the effects of the receive context

[1] We deliberately omitted belief state tuples within belief states for reasons of readability. For a full version of Figure 3, including the belief state tuples see Appendix A.

[2] For the full pseudo-code of the algorithm, see Appendix B.

action c_{sig} (i.e., of the according context signal *sig*) is also present in the preconditions of the action *a* (cf., $\exists t \in eff(c_{sig}) : \exists u \in pre(a), v(t)=v(u)$; lines 2-9). In a second step, it is checked if the context signal could occur (i.e., if all preconditions of the according receive context action c_{sig} are fulfilled within *bs*; cf., lines 12-20). If both conditions hold, c_{sig} is planned (cf., Definition 6).

Thereafter, the belief state after the planned receive context action c_{sig} is determined by means of the context-aware state transition function (cf., Primitive `applyContextSignal` of class `State`). Now, the algorithm continues planning applicable actions for this newly determined belief state regarding the transition function. The forward search stops if either no action is applicable in a belief state or if a goal state or an already considered belief state is reached.

```
1  boolean isRelevantForActionAndState(Action a, State previousState){
2    boolean foundCommonContextVariable = false;
3    for (ContextTuple signalTuple in effects) {
4      ContextTuple actionTuple = a.preconditions.find { e ->
                                              e.variable == signalTuple.variable};
5      if (actionTuple != null){
6        foundCommonContextVariable = true;
7        break;
8      }
9    }
10   if (!foundCommonContextVariable)
11     return false;
12   for (BeliefStateTuple signalTuple in preconditions) {
13     BeliefStateTuple actionTuple = a.preconditions.find { e ->
                                              e.variable == signalTuple.variable};
14     BeliefStateTuple stateTuple = state.bst.find { e ->
                                              e.variable == signalTuple.variable};
15     if (actionTuple != null && stateTuple != null){
16       BeliefStateTuple intersectionTuple =
                          signalTuple.intersect(actionTuple).intersect(stateTuple);
17       if (intersectionTuple == null)
18         return false;
19       }
20   }
21   return true;
22 }
```

Listing 1. Pseudo-Code for the retrieval of relevant context signals.

5 Evaluation

We mathematically evaluated the feasibility of our approach by proving the key properties termination, completeness (i.e., all relevant receive context actions are considered in the resulting planning graph), correctness (i.e., no receive context actions are planned that must not interrupt a regular action) and computational complexity. Further, we evaluated our approach by means of a prototypical implementation of the previously presented algorithm and an experimental evaluation within real world fields of application.

5.1 Mathematical evaluation

In order to prove the feasibility of our approach, we need to ensure that it terminates (Theorem 1), considers all receive context actions (completeness; Theorem 2) and does not plan any incorrect context signals and receive context actions (i.e., is correct; Theorem 3). It is proven, that the approach meets all these requirements and its computational complexity is $O(n^4)$ in the number of context tuples or belief state tuples, which means, the algorithm is computationally efficient (cf., Arora and Barak, 2009; Cobham, 1965). For the proof sketches see Appendix C.

5.2 Prototypical implementation and experimental evaluation

To demonstrate the practical feasibility (cf., Peffers *et al.*, 2008) of our approach, we implemented the proposed algorithm by means of a prototype[3]. A Java implementation of a state transition system served as a basis for our work. We ensured the validity of our prototype by applying several tests using the JUnit framework including unit test, extreme value tests and integration tests. Further, persons other than the programmers validated the source code via a structured walkthrough.

In the next step, we applied our approach to the application fields *Day city trip* and *House building* to evaluate if the constructed context-aware process models are suitable for representing possibly occurring context signals and are thus valid (Adelman, 1991; Peffers *et al.*, 2008)[4]. Here, our running example is a simplified excerpt of a Day city trip process that was defined based on the attractions and activity categories of trip planning websites like *tripadvisor.com*.

Further, to evaluate the efficacy of our approach, we conducted a naturalistic ex-post approach (cf., Venable *et al.*, 2012) within the already mentioned application field *House building*. Within an experimental setting (cf., Hevner *et al.*, 2004) we questioned, in a first step, an experienced architect, which context signals, from his experience, could possibly influence a House building process. This process, according to the architect, consists of three major phases that are required in order to build a house. After the planning phase, which is mainly not context-aware, the shell is constructed. This phase could be heavily influenced by environmental events like rain, raising or lowering temperatures, polluted ground or rising ground water. After the shell is finished, the interior work has to be done. Within this phase, plaster may not dry, for example, which leads to delays or may be dried using construction dehydrators. Based on this given information, we planned both a regular, non-context-aware as well as a context-aware process model by means of our approach.

Thereafter, we presented both process models to the architect (in a next meeting) and questioned him on how he assesses the validity (Adelman, 1991) of the context-aware process model with respect to the context signals he told us before. The professional architect as well as several further experts in the area of process modelling supported the efficacy of our approach. They considered the resulting context-aware process model as valid as it is more suitable regarding the consideration of possibly occurring context signals within process models compared to the regular process model. By means of both, the prototypical implementation and the experimental evaluation we aimed to show that we are able to take process exogenous changes of context variables (due to environmental events) into account in process models of real world processes.

Process name	Number of distinct belief state variables, actions and context signals	Number of actions and belief states in the non-context-aware process model	Number of actions, belief states and receive context actions in the context-aware process model	Average duration for planning the non-context-aware process model	Average duration for planning the context-aware process model
Running example	6 / 3 / 2	5 / 4	26 / 22 / 11	0.16 s	0.19 s
Day city trip	9 / 12 / 10	86 / 46	3714 / 1228 / 2983	0.34 s	3.34 s
House building	29 / 30 / 8	59 / 37	8431 / 3659 / 3430	0.19 s	20.42 s

Table 1. Evaluation results by means of a prototypical implementation.

We examined the size of the non-context-aware process models (i.e., the number of actions and belief states) in contrast to the size of the context-aware process models (i.e., the number of actions, belief states and receive context actions; note that multiple planned receive context actions may be subsumed

[3] We executed the prototype on a Dell Latitude Notebook with an Intel Core i7-2640M, 2.80 GHz, 8GB RAM running on Windows 8.1 (Version 6.3.9600) 64 bit and Java 1.8.0 (build 1.8.0.-b132) 64 bit.

[4] Thereby we used our prototypical implementation, which was also suitable for creating the figures within this paper.

by means of Interruptible Activity Regions in Figure 3) as an indicator of the complexity of the planning problem on the one hand and for the resulting flexibility concerning environmental influences in general and context events in particular on the other hand. Large process models (like the House building process) may not be very suitable for visualizing processes but, however, they are particularly suitable as a foundation for process-aware information systems and especially for automated execution of processes by means of, for example, workflow-engines (cf. e.g., Khan *et al.*, 2010; Weber, 2007). A summary of all measured properties and the regarding durations for planning both the regular and the context-aware process model can be seen in Table 1.

6 Conclusion, limitations and further research

Within this paper, we presented an approach construct complex and comprehensive context-aware process models in an automated way and therefore contribute to a heretofore unsolved issue. Thereby, we consider both static and non-static context within the regular control-flow of process models. By means of such context-aware process models we are able to shift complexity from runtime to design time, which may be useful especially for performers without broad knowledge and expertise, and further enable the automated execution of context-aware processes by means of workflow-engines, for instance (cf. e.g., Khan *et al.*, 2010). As we use well-known process modeling elements like Accept Event Actions and Interruptible Activity Regions within our approach, we ensure readability and understandability of the constructed context-aware process models. We further ensure the feasibility of our approach by means of mathematical proofs of its key properties, a prototypical implementation and the application to real world processes.

However, there are some limitations of our work, which have to be addressed in further research. Especially considering large processes like, for example, the House building process (cf. Section 5), our graphical notation by means of receive context actions implies very large resulting process models. As in process modeling notations like UML Activity diagrams and BMPN, belief states are not (explicitly) considered, further research is needed to increase readability of context-aware process models by means of, for instance, reducing the amount of required nodes within the process model. Some promising preliminary ideas tend to identify areas of (subsequent) actions in process models that could all be interrupted by one context signal. Thus, Interruptible Activity Regions as used in UML (Object Management Group, 2013) seem promising to reduce the number of receive context actions. We already strived this to some extent (as seen in Figure 3), but, as we wanted to focus on the construction of context-aware process models, we did not use a specific process modeling language. However, we already consider the issue of transferring the graph resulting from the nondeterministic transition system to a process modeling language within our research by means of automatically constructing control flow patterns and using UML activity diagrams (cf. e.g., Heinrich *et al.*, 2012 and Heinrich *et al.*, 2009). Moreover, to enable the automated planning of context-aware process models in a user-friendly way, an existing process planning tool needs to be extended by the context-aware features.

Further, following the approach of Russell *et al.* (2006), recovery strategies to gracefully handle the interruption of actions due to context signals may be considered. For example, an already started action (e.g., a cycling tour) may be rolled back (return home) or even continued if a suitable action is available and applicable (take on rain jacket). Basically, we are able to plan such recovery strategies by means of modeling the according recovery actions. However, it seems promising to support a modeler in terms of proposing potential recovery actions in a (semi-)automated way where possible.

Additionally, besides context events, other users could affect a user during the conduction of his/her process. As the consideration of multi-user processes within the research strand of automated planning is an unsolved issue, this would be a relevant next step in order to fully cover adaptive process models regarding external influences regarding other users and context events.

Acknowledgements

The research was funded by the Austrian Science Fund (FWF): P 23546-G11.

References

Abowd, G., Dey, A., Brown, P., Davies, N., Smith, M. and Steggles, P. (1999), "Towards a Better Understanding of Context and Context-Awareness", in Gellersen, H.-W. (Ed.), *Handheld and Ubiquitous Computing*, *Lecture Notes in Computer Science*, Vol. 1707, Springer Berlin Heidelberg, pp. 304-307.

Adelman, L. (1991), "Experiments, Quasi-Experiments, and Case Studies: A Review of Empirical Methods for Evaluating Decision Support Systems", in *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 21 No. 2, pp. 293–301.

Arora, S. and Barak, B. (2009), *Computational complexity: a modern approach*, Cambridge University Press.

Attard, J., Scerri, S., Rivera, I. and Handschuh, S. (2013), "Ontology-based situation recognition for context-aware systems", in *Proceedings of the 9th International Conference on Semantic Systems*, ACM, pp. 113–120.

Ayora, C., Torres, V., Reichert, M., Weber, B. and Pelechano, V. (2013), "Towards Run-time Flexibility for Process Families: Open Issues and Research Challenges", in *Business Process Management Workshops Lecture Notes in Business Information Processing*, No. 132, pp. 477–488.

Baldauf, M., Dustdar, S. and Rosenberg, F. (2007), "A survey on context-aware systems", in *International Journal of Ad Hoc and Ubiquitous Computing*, Vol. 2 No. 4, pp. 263–277.

Bertoli, P., Cimatti, A., Roveri, M. and Traverso, P. (2001), "Planning in nondeterministic domains under partial observability via symbolic model checking", in *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*, Vol. 1, pp. 473–478.

Bertoli, P., Cimatti, A., Roveri, M. and Traverso, P. (2006), "Strong planning under partial observability", in *Artificial Intelligence*, Vol. 170 No. 4–5, pp. 337–384.

Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A. and Riboni, D. (2010), "A survey of context modelling and reasoning techniques", in *Pervasive and mobile computing*, Vol. 6 No. 2, pp. 161–180.

Bider, I. (2005), "Masking flexibility behind rigidity: Notes on how much flexibility people are willing to cope with", in *Proceedings of the CAiSE*, Vol. 5, pp. 7–18.

Bucchiarone, A., Mezzina, C. and Pistore, M. (2013), "Captlang: a language for context-aware and adaptable business processes", in *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems, Pisa, January 23 -25, 2013*.

Chen, H., Finin, T. and Joshi, A. (2003), "An Ontology for Context-Aware Pervasive Computing Environments", in *The Knowledge Engineering Review*, Vol. 18 No. 3, pp. 197–207.

Cobham, A. (1965), "The intrinsic computational difficulty of functions", in *Logic, Methodology, and Philosophy of Science II*.

Dey, A.K. (2001), "Understanding and using context", in *Personal and ubiquitous computing*, Vol. 5 No. 1, pp. 4–7.

Dobson, S., Denazis, S., Fernández, A., Gaïti, D., Gelenbe, E., Massacci, F., Nixon, P., Saffre, F., Schmidt, N. and Zambonelli, F. (2006), "A survey of autonomic communications", in *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, Vol. 1 No. 2, pp. 223–259.

Fujii, K. and Suda, T. (2009), "Semantics-based context-aware dynamic service composition", in *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, Vol. 4 No. 2, p. 12.

Ghallab, M., Nau, D. and Traverso, P. (2004), *Automated Planning: Theory & Practice*, Morgan Kaufmann, San Francisco.

Gottschalk, F. and La Rosa, M. (2010), "Process Configuration", in ter Hofstede, A.H.M., van der Aalst, W.M.P., Adams, M. and Russell, N. (Eds.), *Modern Business Process Automation*, Springer Berlin Heidelberg, pp. 459–487.

Gottschalk, F., van der Aalst, W.M.P. and Jansen-Vullers, M.H. (2007), "Configurable process models—a foundational approach", in *Reference Modeling*, Springer, pp. 59–77.

Hallerbach, A., Bauer, T. and Reichert, M. (2008), "Context-based configuration of process variants", in *Proceeding of the 3rd International Workshop on Technologies for Context-Aware Business Process Management (TCoB 2008)*.

Hallerbach, A., Bauer, T. and Reichert, M. (2010), "Capturing variability in business process models: the Provop approach", in *Journal of Software Maintenance and Evolution: Research and Practice*, Vol. 22 No. 6-7, pp. 519–546.

Heinrich, B., Bolsinger, M. and Bewernik, M.-A. (2009), "Automated planning of process models: the construction of exclusive choices", in Chen, H. and Slaughter, S.A. (Eds.), *30th International Conference on Information Systems (ICIS)*, Springer, Phoenix, Arizona, USA, pp. 1–18.

Heinrich, B., Klier, M., Lewerenz, L. and Zimmermann, S. (2015), "Quality of Service-aware Service Selection: A Novel Approach Considering Potential Service Failures and Non-Deterministic Service Values", in *INFORMS Service Science, A Journal of the Institute for Operations Research and the Management Sciences*, Vol. 7 No. 1, pp. 48–69.

Heinrich, B., Klier, M. and Zimmermann, S. (2012), "Automated Planning of Process Models – Towards a Semantic-based Approach", in *Smolnik, S; Teuteberg, F; Thomas, O. (Ed.): Semantic Technologies for Business and Information Systems Engineering: Concepts and Applications. Hershey: IGI Global*, pp. 169–194.

Henneberger, M., Heinrich, B., Lautenbacher, F. and Bauer, B. (2008), "Semantic-Based Planning of Process Models", in Bichler, M., Hess, T., Krcmar, H., Lechner, U., Matthes, F., Picot, A., Speitkamp, B. and Wolf, P. (Eds.), *Multikonferenz Wirtschaftsinformatik (MKWI)*, GITO-Verlag, München, pp. 1677–1689.

Hevner, A.R., March, S.T., Park, J. and Ram, S. (2004), "Design science in information systems research", in *MIS Q*, Vol. 28 No. 1, pp. 75–105.

Hoffmann, J., Weber, I. and Kraft, F.M. (2009), "Planning@ sap: An application in business process management", in *Proceedings of the 2nd International Scheduling and Planning Applications woRKshop (SPARK'09)*.

Hoffmann, J., Weber, I. and Kraft, F.M. (2012), "SAP Speaks PDDL: Exploiting a Software-Engineering Model for Planning in Business Process Management", in *Journal of Artificial Intelligence Research*, Vol. 44 No. 1, pp. 587–632.

Hu, B., Wang, Z. and Dong, Q. (2012), "A Modeling and Reasoning Approach Using Description Logic for Context-Aware Pervasive Computing. Emerging Research in Artificial Intelligence and Computational Intelligence", in Lei, J., Wang, F.L., Deng, H. and Miao, D. (Eds.), *Communications in Computer and Information Science*, Springer Berlin Heidelberg, pp. 155–165.

Khan, F.H., Bashir, S., Javed, M.Y., Khan, A. and Khiyal, Malik Sikandar Hayat (2010), "QoS Based Dynamic Web Services Composition & Execution", in *arXiv preprint arXiv:1003.1502*.

La Rosa, M., Dumas, M., ter Hofstede, A.H.M. and Mendling, J. (2011), "Configurable multi-perspective business process models", in *Information Systems*, Vol. 36 No. 2, pp. 313–340.

Mattos, T.d.C., Santoro, F.M., Revoredo, K. and Nunes, V.T. (2014), "A formal representation for context-aware business processes", in *Computers in Industry*, Vol. 65 No. 8, pp. 1193–1214.

Meyer, H. and Weske, M. (2006), "Automated service composition using heuristic search", in *Business Process Management*, pp. 81–96.

Mulholland, A., Thomas, C.S., Kurchina, P. and Woods, D. (2006), *Mashup corporations: The end of business as usual*, Evolved Technologist Press.

Nadoveza, D. and Kiritsis, D. (2014), "Ontology-based approach for context modeling in enterprise applications", in *Special Issue on The Role of Ontologies in Future Web-based Industrial Enterprises*, Vol. 65 No. 9, pp. 1218–1231.

Object Management Group (2013), "OMG Unified Modeling Language TM (OMG UML). Version 2.5", available at: http://www.omg.org/spec/UML/2.5/Beta2/PDF (accessed 16 October 2014).

Object Management Group (2014), "Business Process Model and Notation (BPMN). Version 2.0.2", available at: http://www.omg.org/spec/BPMN/2.0.2/PDF (accessed 16 October 2014).

Öztürk, P. and Aamodt, A. (1997), "Towards a Model of Context for Case-Based Diagnostic Problem Solving", in *IN CONTEXT-97; PROCEEDINGS OF THE*, pp. 198–208.

Peffers, K., Tuunanen, T., Rothenberger, M.A. and Chatterjee, S. (2008), "A Design Science Research Methodology for Information Systems Research", in *Journal of Management Information Systems*, Vol. 24 No. 3, pp. 45–78.

Regev, G., Bider, I. and Wegmann, A. (2007), "Defining business process flexibility with the help of invariants", in *Software Process: Improvement and Practice*, Vol. 12 No. 1, pp. 65–79.

Reichert, M.U. and Weber, B. (2012), *Enabling flexibility in process-aware information systems: challenges, methods, technologies*, Springer.

Rosemann, M., Recker, J. and Flender, C. (2008), "Contextualisation of business processes", in *International Journal of Business Process Integration and Management*, Vol. 3 No. 1, pp. 47–60.

Rosemann, M., Recker, J. and Flender, C. (2010), "Designing context-aware business processes", in *Systems Analysis and Design: People, Processes, and Projects. Armonk, NY: ME Sharpe, Inc*, pp. 53–74.

Rosemann, M. and van der Aalst, W.M.P. (2007), "A configurable reference modelling language", in *Information Systems*, Vol. 32 No. 1, pp. 1–23.

Russell, N., van der Aalst, W.M.P. and ter Hofstede, A.H.M. (2006), "Exception Handling Patterns", in *Process-Aware Information Systems. Technical report, BPM Center Report BPM-06-04, BPMcenter. org*.

Sakurai, Y., Takada, K., Anisetti, M., Bellandi, V., Ceravolo, P., Damiani, E. and Tsuruta, S. (2012), "Toward sensor-based context aware systems", in *Sensors*, Vol. 12 No. 1, pp. 632–649.

Satyanarayanan, M. (2001), "Pervasive computing: Vision and challenges", in *Personal Communications, IEEE*, Vol. 8 No. 4, pp. 10–17.

Schonenberg, H., Mans, R., Russell, N., Mulyar, N. and van der Aalst, W.M.P. (2008), "Process flexibility: A survey of contemporary approaches", in *Advances in Enterprise Engineering I*, Springer, pp. 16–30.

Simons, C. and Wirtz, G. (2007), "Modeling context in mobile distributed systems with the UML", in *Visual Interactions in Software Artifacts*, Vol. 18 No. 4, pp. 420–439.

Soffer, P. (2005), "On the notion of flexibility in business processes", in *Proceedings of the CAiSE*, Vol. 5, pp. 35–42.

Strang, T. and Linnhoff-Popien, C. (2004), "A Context Modeling Survey", in *Workshop Proceedings. First International Workshop on Advanced Context Modelling, Reasoning And Management at UbiComp, Nottingham, England, September 7, 2004*.

Swenson, K.D. (2010), "Mastering the Unpredictable", in *How Adaptive Case Management Will Revolutionize the Way That Knowledge Workers Get Things Done, Glossary*, pp. 314–317.

Sycara, K., Paolucci, M., Ankolekar, A. and Srinivasan, N. (2003), "Automated discovery, interaction and composition of semantic web services", in *Web Semantics: Science, Services and Agents on the World Wide Web*, Vol. 1 No. 1, pp. 27–46.

Tealeb, A., Awad, A. and Galal-Edeen, G. (2014), "Context-Based Variant Generation of Business Process Models", in *Enterprise, Business-Process and Information Systems Modeling*, Springer, pp. 363–377.

van der Aalst, W., Adriansyah, A., Medeiros, A. de, Arcieri, F., Baier, T., Blickle, T., Bose, J., van den Brand, P., Brandtjen, R. and Buijs, J. (2012), "Process Mining Manifesto", in *BPM 2011 Workshops, Part I, LNBIP 99*, pp. 169–194.

van der Aalst, W.M.P., Dreiling, A., Gottschalk, F., Rosemann, M. and Jansen-Vullers, M.H. (2006), "Configurable process models as a basis for reference modeling", in Bussler, C.J. and Haller, A. (Eds.), *Business Process Management Workshops*, Springer, pp. 512–518.

van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B. and Barros, A.P. (2003), "Workflow Patterns", in *Distributed and Parallel Databases*, Vol. 14 No. 1, pp. 5–51.

van der Aalst, W.M.P., Weijters, T. and Maruster, L. (2004), "Workflow mining: Discovering process models from event logs", in *IEEE Transactions on Knowledge and Data Engineering*, Vol. 16 No. 9, pp. 1128–1142.

Venable, J., Pries-Heje, J. and Baskerville, R. (2012), "A Comprehensive Framework for Evaluation in Design Science Research", in Peffers, K., Rothenberger, M. and Kuechler, B. (Eds.), *Design*

Science Research in Information Systems. Advances in Theory and Practice, *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 423–438.

Wang, J., Zeng, C., He, C., Hong, L., Zhou, L., Wong, R.K. and Tian, J. (2012), "Context-aware role mining for mobile service recommendation", in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, ACM, Trento, Italy, pp. 173–178.

Wang, X.H., Zhang, D.Q., Gu, T. and Pung, H.K. (2004), "Ontology based context modeling and reasoning using OWL", in *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, pp. 18–22.

Weber, I. (2007), "Requirements for Implementing Business Process Models through Composition of Semantic Web Services", in Gonçalves, R., Müller, J., Mertins, K. and Zelm, M. (Eds.), *Enterprise Interoperability II*, Springer London, pp. 3–14.

Wetzstein, B., Ma, Z., Filipowska, A., Kaczmarek, M., Bhiri, S., Losada, S., Lopez-Cob, J.-M. and Cicurel, L. (2007), "Semantic Business Process Management: A Lifecycle Based Requirements Analysis", in *SBPM*.

Ye, J., Dobson, S. and McKeever, S. (2012), "Situation identification techniques in pervasive computing: A review", in *Pervasive and mobile computing*, Vol. 8 No. 1, pp. 36–66.

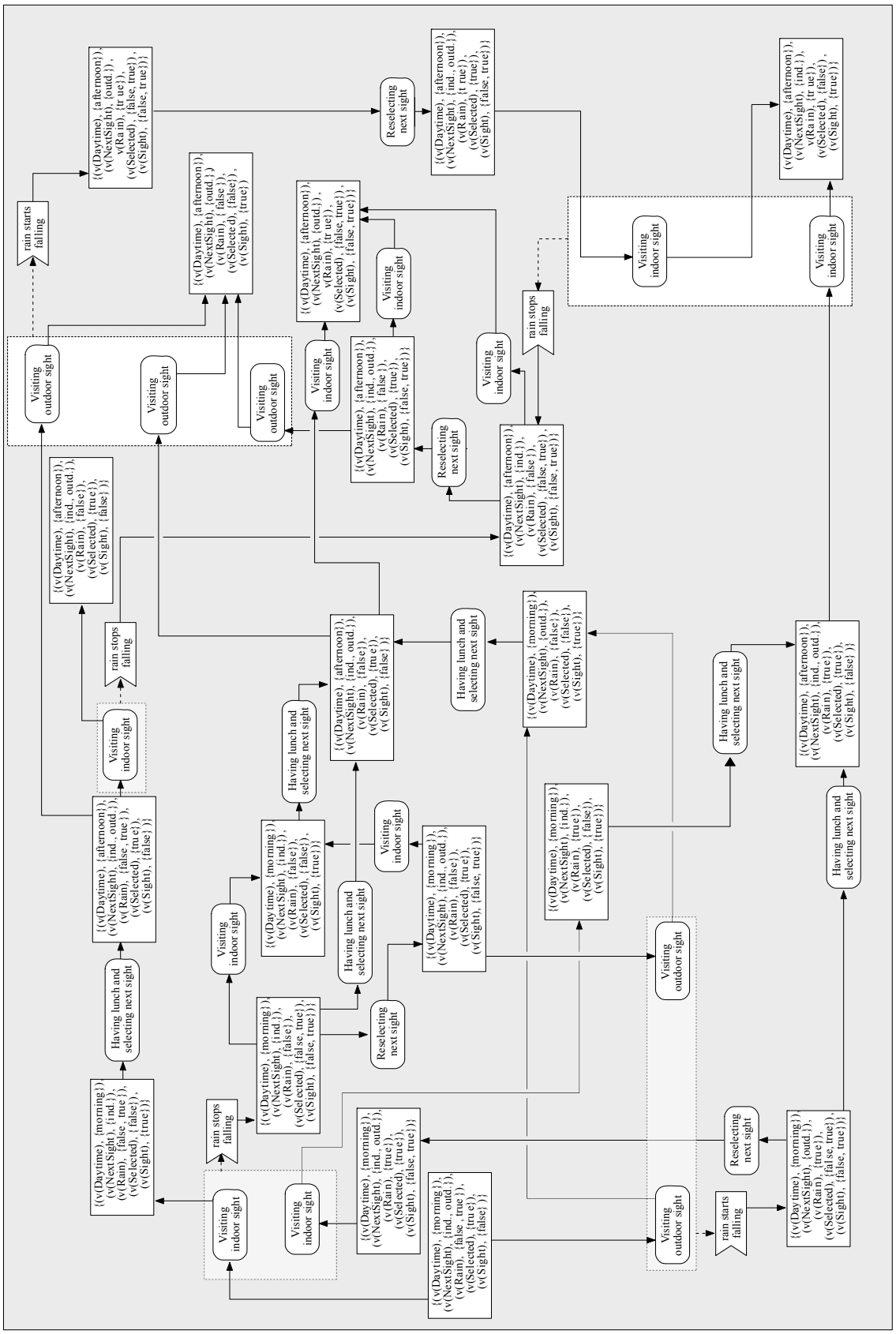
Zhang, D., Adipat, B. and Mowafi, Y. (2009), "User-Centered Context-Aware Mobile Applications—The Next Generation of Personal Mobile Computing", in *Communications of the Association for Information Systems*, Vol. 24 No. 1, p. 3.

Zhou, J., Gilman, E., Palola, J., Riekki, J., Ylianttila, M. and Sun, J. (2011), "Context-aware pervasive service composition and its implementation", in *Personal and ubiquitous computing*, Vol. 15 No. 3, pp. 291–303.

Zhu, X., Recker, J., Zhu, G., Santoro, F.M. and Al-Mashari, M. (2014a), "Exploring location-dependency in process modeling", in *Business Process Management Journal*, Vol. 20 No. 6.

Zhu, X., Zhu, G., Vanthienen, J., Baesens, B. and others (2014b), "Towards location-aware process modeling and execution", in *Business Process Management Workshops*.

Appendix A: Full version of Figure 3



Appendix B: Pseudocode of the algorithm

```
class Algorithm {

  generateGraph(){
    recursiveCreateGraph(initialState);
  }

  boolean recursiveCreateGraph(State state){
    if (state.isGoalState() || state.isAlreadyChecked()){
      return true
    } else {
      List<Action> applicableActions = state.getApplicableActions();
      if (applicableActions.isEmpty())
        return false;
      else {
        boolean successfullyReachedGoal = false
        for (Action action : applicableActions){
          State nextState = state.apply(action);
          successfullyReachedGoal =
                      (recursiveCreateGraph(nextState) || successfullyReachedGoal);
          for (ContextSignal contextSignal : allContextSignals){
            if (contextSignal.isRelevantForActionAndState(action, state){
              State stateAfterContextSignal =
                                  state.applyContextSignal(action, contextSignal);
              recursiveCreateGraph(stateAfterContextSignal);
            }
          }
        }
        return successfullyReachedGoal;
      }
    }
  }
}
```

```
class Action {

  List<BeliefStateTuple> preconditions
  List<BeliefStateTuple> effects

  boolean isApplicableInState(State previousState){
    for (BeliefStateTuple preconditionTuple in preconditions) {
      ContextTuple stateTuple = state.bst.find { e ->
                                              e.variable == signalTuple.variable};
      if (stateTuple != null){
        ContextTuple intersectionTuple = preconditionTuple.intersect(stateTuple);
        if (intersectionTuple == null)
          return false;
      } else return false;
    }
    return true;
  }
}
```

```
class ContextSignal {

  List<BeliefStateTuple> preconditions
  List<BeliefStateTuple> effects

  boolean isRelevantForActionAndState(Action a, State previousState){
```

```
    boolean foundCommonContextVariable = false;
    for (ContextTuple signalTuple in effects) {
      ContextTuple actionTuple = a.preconditions.find { e ->
                                                e.variable == signalTuple.variable};
      if (actionTuple != null){
        foundCommonContextVariable = true;
        break;
      }
    }
    if (!foundCommonContextVariable)
      return false;
    for (BeliefStateTuple signalTuple in preconditions) {
      BeliefStateTuple actionTuple = a.preconditions.find { e ->
                                                e.variable == signalTuple.variable};
      BeliefStateTuple stateTuple = state.bst.find { e ->
                                                e.variable == signalTuple.variable};
      if (actionTuple != null && stateTuple != null){
        BeliefStateTuple intersectionTuple =
                          signalTuple.intersect(actionTuple).intersect(stateTuple);
        if (intersectionTuple == null)
          return false;
        }
    }
    return true;
  }
}
```

```
class State {

  List<BeliefStateTuple> bst

  List<Action> getApplicableActions(){
    List<Action> applicableActions = new List<Action>();
    for (Action action : allAction){
      if (action.isApplicableInState(this))
        applicableActions.add(action);
    }
    Return applicableActions;
  }

  State apply(Action action){
    State nextState = this.clone();
    nextState.intersectBeliefStateTuples(action.preconditions);
    nextState.insertOrReplaceBeliefStateTuples(action.effects);
    return nextState;
  }

  State applyContextSignal(Action action, ContextSignal contextSignal){
    State nextState = this.clone();
    nextState.intersectBeliefStateTuples(action.preconditions);
    nextState.insertAndUnifyBeliefStateTuples(action.effects);
    nextState.intersectBeliefStateTuples(contextSignal.preconditions);
    nextState.insertOrReplaceBeliefStateTuples(contextSignal.effects);
    return nextState;
  }
}
```

Appendix C: Mathematical proofs of key properties

Theorem 1. The execution of the algorithm terminates.

Proof sketch. Termination is shown by proving that all relevant sets are finite and the algorithm could not reach an endless loop. We will, in the following, show that the algorithm terminates.

As the set *A* of process actions is finite, there are only finitely many actions that are applicable in a belief state *bs*. Further, there are only finitely many next belief states that are constructed by means of the regular state transition. In each of those newly constructed belief states then again finite actions are applicable. As the effects of actions are independent of the belief state the action was planned in, there can only arise finitely many belief states.

Hence, it is sufficient to prove that a belief state could not be considered twice by means of our search algorithm (cf., due to a loop in the planning graph). This is ensured by the termination criteria of the algorithm. If an already considered belief state is reached, the forward search stops per definition. By means of this criterion we ensure that a belief state is only considered once.

To prove that the planning of receive context actions terminates we need to consider the set of context signals *SIG*, which is finite as well. Thus, for each (of finitely many) combination of a belief state and a following action, only finite context signals could occur. Further, as a context signal (i.e., the effects of the according receive context action) is independent of the belief state and the action during which it occurs, there can only arise finitely many belief states by means of context signals. From these belief states the planning starts again, and during this newly planned new context signals could be relevant. As the termination criteria are considered here as well, we have proven that the planning of context-aware process models terminates. *q.e.d.*

Further, to ensure a comprehensive process model, we need to ensure that our approach considers all context signals and their according receive context actions (i.e., completeness).

Theorem 2. The approach is complete (i.e., it plans all receive context actions for all context signals, that could occur during planned actions).

Proof sketch. Assume there is a context signal *sig* that could occur and an according receive context action c_{sig} that is not planned. Following Definition 6 the planning graph has to contain a belief state bs followed by an action *a* satisfying $\forall t \in effects(c_{sig}) \, \exists u \in precond(a) | v(t) = v(u)) \wedge (\forall x \in precond(c_{sig}) \, \exists w \in bs \mid v(w) = v(x) \wedge (r(x) \cap r(w) \neq \emptyset))$ so that sig could occur. As our algorithm retrieves all context signals and according receive context actions for all planed actions and *sig* could occur during the conduction of a, c_{sig} is planned and we have a contradiction. *q.e.d.*

Theorem 3. The approach is correct (i.e., it does not plan receive context actions according to context signals that could not occur)

Proof sketch. Let us assume a context signal *sig* that could not occur during any action exists and an according receive context action c_{sig} is planned by the algorithm. As there is a point at which c_{sig} could have been planned, there has to be an action *a* in a belief state *bs* so that Definition 6 is met. As this is also the condition that a context signal could occur our assumption is not true and thus the algorithm is correct. *q.e.d.*

Further, we evaluated the computational complexity of our approach, which means, we analyzed based on our nondeterministic context-aware state transition system the planning of receive context actions. The following two statements must both hold in order to plan a receive context action:

1) $\exists\, t \in eff(c_{sig}): \exists u \in pre(a), v(t)=v(u)$ and
2) $\forall\, x \in pre(c_{sig}) \, \exists\, w \in bs : v(w) = v(x) \wedge ((\nexists\, y \in pre(a) : v(y) = v(x) \wedge r(x) \cap r(w) \neq \emptyset) \vee (\exists\, y \in pre(a) : v(y) = v(x) \wedge r(x) \cap r(y) \cap r(w) \neq \emptyset))$.

To ensure that 1) holds, at least one context variable must be present in both $eff(c_{sig})$ and $pre(a)$. Both sets must be compared pairwise and thus, the worst case here is that only the context variables in both sets compared at last are equal (i.e., $v(t)=v(u)$) and both sets do not overlap in any other context variable. Thus, the pairwise comparison of the context variables in those two sets results in a complexity of

$O(|CT|^2)$, $O(|C|)$ and $O(|A|)$. For the second statement, $|BS|$ belief states, $|A|$ preconditions of actions and $|C|$ preconditions of receive context actions must be compared to each other and for each of these combinations two pairwise comparisons for each ($|BST|$) belief state tuple are required. Thus, the computational complexity of our approach is $O(|BST|^2)$, $O(|CT|)$, $O(|C|)$, $O(|BS|)$ and $O(|A|)$. To sum it up, in the worst-case-scenario, the algorithm is in quadratic time in the number of context tuples or belief state tuples, which means, the algorithm is computationally efficient (cf., Arora and Barak, 2009; Cobham, 1965).