Reduction Techniques for Efficient Behavioral Model Checking in Adaptive Case Management

Christoph Czepa, Huy Tran, Uwe Zdun
Research Group Software Architecture
Faculty of Computer Science
University of Vienna
Währingerstraße 29
1090 Vienna, Austria
{firstname.lastname}@univie.ac.at

Thanh Tran, Erhard Weiss, Christoph Ruhsam
ISIS Papyrus Europe AG
Alter Wienerweg 12
2344 Maria Enzersdorf, Austria
{firstname.lastname}@isis-papyrus.com

ABSTRACT

Case models in Adaptive Case Management (ACM) are business process models ranging from unstructured over semi-structured to structured process models. Due to this versatility, both industry and academia show growing interest in this approach. This paper discusses a model checking approach for the behavioral verification of ACM case models. To counteract the high computational demands of model checking techniques, our approach includes state space reduction techniques as a preprocessing step before state-transition system generation. Consequently, the problem size is decreased, which decreases the computational demands needed by the subsequent model checking as well. An evaluation of the approach with a large set of LTL specifications on two real-world case models, which are representative for semi-structured and structured process models and realistic in size, shows an acceptable performance of the proposed approach.

CCS Concepts

•Applied computing → Business process modeling; Business process management systems; •Software and its engineering → Model checking; Software verification;

Keywords

Case Management, Business Process Management, Verification, Model Checking, State Space Reduction

1. INTRODUCTION

Many software vendors offer Adaptive Case Management (ACM) as their business process management solution [9]. A case model in ACM is a business process model that describes the basic structure and behavior of the case instances (aka business process instances) that originate from it. Consequently, a case model must neither contain structural errors (e.g., inaccessible elements) nor undesired behavior (e.g., non-compliance with laws, standards or best practices) since all case instances derived from this model would be affected.

Case modeling is emerging as a major approach for business process modeling (cf. e.g., [9, 15, 17]. While there exist ways to detect structural inconsistencies in case models (cf. [5]), the behavioral verification of case models has not yet been considered. A powerful, yet computationally expensive approach for detecting undesired behavior is model checking [4]. Model checking is a verification technique that explores all possible execution traces of a (business process) model. Any undesired behavior is detected, provided the specification is correctly defined. However, model checking has a downside as well, namely its computational complexity, which is *PSPACE-complete* [23]. That is, the runtime of model checking grows exponential with the problem size. The work presented in this paper addresses the following research questions: (RQ1) Since model checking is known to be computationally expensive, how can case models be model checked efficiently? (RQ2) Can model checking be applied to real-world case models that are realistic in size and structure?

This paper discusses a model checking approach for detecting undesired behavior in case models. The presented approach comprises four steps: (1) Case elements that are not required for the detection of undesired behavior are removed (*Case Element Reduction*). (2) Conditions are abstracted by pre-computing all possible outcomes (*Condition Reduction*). (3) A case model is transformed to a state-transition system for model checking (*Model Transformation*). (4) Model checking is performed to find out whether the system meets a specified desired behavior (*Verification by Model Checking*). Steps 1 & 2 aim at improving the performance of model checking by reducing the state space that is to be considered in model checking of a case model. The approach is applied to two real-world case models that are representative for different degrees of structuredness, and both are realistic in size.[1] The applied reduction techniques and the overall approach enable the verification of those real-world case models within reasonable response times.

2. MOTIVATION

Let us consider an example from health care regarding the treatment of a fracture, shown in Figure 1. The case model is described

[1]Completely unstructured case models are not considered since a design time verification of those would be pointless due to the unconstrained order of execution of the elements of such a process.

SAC 2017,April 03-07, 2017, Marrakech, Morocco

http://dx.doi.org/10.1145/3019612.3019617

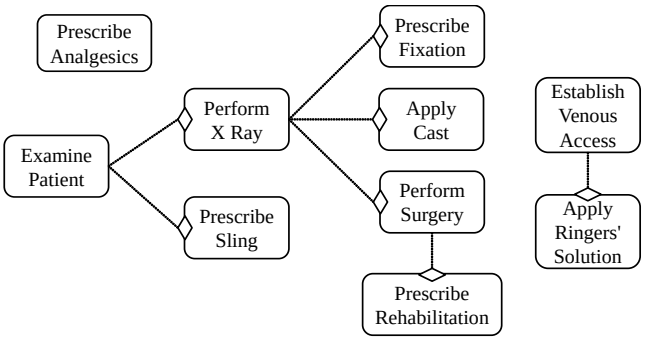


Figure 1: Case model for the treatment of a suspected fracture

in Case Management Model and Notation (CMMN) [19] which can be used to model the essential structures of ACM case model (cf. [5, 17, 15]). In total, this case model has ten tasks, seven dependencies (dotted lines) and seven entry criteria (diamond symbols). We will use this simple example to discuss our approach. Please note that the approach is able to handle complex case model (i.e., nested stages, goals) as well. *Examine Patient*, *Prescribe Analgesics* and *Establish Venous Access* are not dependent on any other element and can be started as decided by the business users. Other elements are dependent: For example, *Perform Surgery* is dependent on the completion of *Perform X Ray* and can only be started if the condition "*diagnosis == 'compound fracture'* " (note: displaying conditions of criteria is omitted in the diagram for reasons of clarity) of the entry criterion is met, and *Prescribe Fixation* is dependent on the completion of *Perform X Ray* and can only be started if the condition "*diagnosis == 'contusion'* " of the entry criterion is met.

Formal verification of models is a recurring research interest. While flow-driven business processes models have already been studied extensively (cf. e.g., [8, 13, 22, 21]), the verification of case models to detect undesired behavior has not yet been investigated to this extent. In this paper, we employ a well-established verification technique, namely model checking [4], which requires us to define the semantics of case models as a state-transition system. This state-transition system is then checked against a specification in a formal temporal logic (cf. e.g., [3]) such as Linear Temporal Logic (LTL) or Computation Tree Logic (CTL). This paper is concerned with finding an adequate state-transition system for case models that enables their formal verification within acceptable response times.

As regards this subject, we identify the following main issue: As execution times in model checking grow exponentially with the problem size, the problem must be kept small in size, which is challenging due to the rich semantics of case models.

We will use the example given in Figure 1 to discuss possible ways for reducing the problem size as a preprocessing step before model checking. Let us consider specifications that describe desired behavior. Here, we consider well-established patterns from software verification [6] and business process compliance [7]. For instance, the *Exclusive* pattern "*P EXCLUSIVE Q*" (where the presence of P mandates the absence of Q) can be applied to express that *Prescribe Fixation* demands the absence of *Prescribe Rehabilitation*. This specification could help to avoid unnecessary costs when a rehabilitation is medically not indicated. For the verification of the specification "*Prescribe Fixation EXCLUSIVE Prescribe Rehabilitation*", it is not necessary to consider each and every element of the case. Only those elements that may have an influence on the outcome of the model checking must be retained. By removing *Prescribe Analgesics*, *Prescribe Sling*, *Establish Venous Access*, *Apply Ringers' Solution*, and *Apply Cast*, the case model can be reduced in size considerably. Moreover, since *Examine Patient* must inevitably lead to *Perform X Ray* before *Prescribe Rehabilitation* and *Prescribe Fixation*, they can be joined to a single activity, which further reduces the problem size. What remains as a possible way to reduce the problem size for model checking are the conditions of the criteria. Since conditions are evaluated on basis of data, it would require the inclusion of data in the state-transition system for model checking of the specification, which would introduce new variables and increase the state space. To avoid this, we propose to divide the problem into smaller pieces by precomputing the behavior of conditions. For example, once *Perform X Ray* is complete, the outgoing dependencies to the entry criteria of *Prescribe Fixation* and *Perform Surgery* get triggered. Since the condition "*diagnosis == 'contusion'* " of the entry criterion of *Prescribe Fixation* is contradictory to the condition "*diagnosis == 'compound fracture'* " of the entry criterion of *Perform Surgery*, only either *Prescribe Fixation* or *Perform Surgery* is possible, but not both of them. By applying these reductions, the model is reduced in size and complexity, which improves model checking performance. The given example will be revisited in Section 5 & 6 where the reduction techniques are discussed in detail.

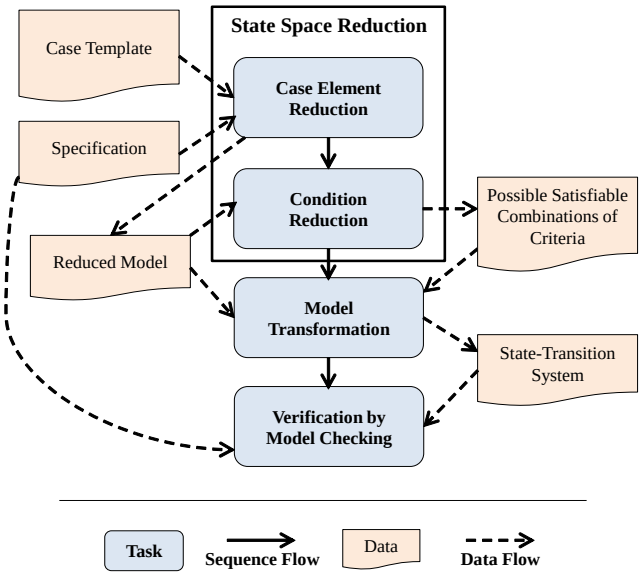


Figure 2: Approach Overview

3. APPROACH OVERVIEW

Figure 2 shows an overview of the approach. *Case Element Reduction* uses the provided *Case Template* (which is first checked to be free of structural inconsistencies, cf. [5]) and *Specification* to create a *Reduced Model*. Tasks, goals, stages, criteria and dependencies that are not needed to model check the given specification have been removed from this model. *Condition Reduction* preprocesses all the possible combinations of criteria that can be activated at once. By this, the approach circumvents the explicit

modeling of these conditions, which would also require the explicit consideration of all data attributes that are referenced in the conditions of criteria. *Model Transformation* uses the *Reduced Model* and the *Possible Activation Combinations of Criteria* to create a *State-Transition System* for model checking. The *Verification by Model Checking* uses this *State-Transition System* and evaluates it against the provided *Specification*.

4. FORMALIZATION

A case model $\mathcal{M}$ is a tuple $(\mathcal{T}_a, \mathcal{T}, \mathcal{G}, \mathcal{S}, \mathcal{E}, \mathcal{X}, \mathcal{C}, \mathcal{D}, \zeta_\mathcal{E}, \zeta_\mathcal{X}, \eta, \alpha, \mathcal{T}_\mathcal{F}, \mathcal{F}, \phi, \delta, \mathcal{ED}, \rho, \mathcal{CE}, \sigma_{\mathcal{CE}})$ where

- $\mathcal{T}_a \supseteq \mathcal{T} \cup \bigcup_{p \in \mathcal{F}} p.\mathcal{R}$ is a set of all tasks, $\mathcal{T}$ is a set of case tasks, $\mathcal{G}$ is a set of goals, $\mathcal{S}$ is a set of stages, $\mathcal{E}$ is a set of entry criteria, $\mathcal{X}$ is a set of exit criteria, $\mathcal{C} = \mathcal{E} \cup \mathcal{X}$ is a set of criteria, $\mathcal{D}$ is a set of dependencies (where $d = (d_s, d_t) \in \mathcal{D}$ means that d_t is dependent on d_s),
- $\zeta_\mathcal{E} : \mathcal{E} \mapsto \mathcal{T} \cup \mathcal{G} \cup \mathcal{S}$ is a total non-injective function which maps an entry criterion to a task, goal[2], or stage,
- $\zeta_\mathcal{X} : \mathcal{X} \mapsto \mathcal{T} \cup \mathcal{S}$ is a total non-injective function which maps an exit criterion to a task or stage,
- $\eta : \mathcal{G} \mapsto \mathcal{G}$ is a partial non-injective function which maps a goal to a parent goal,
- $\alpha : \mathcal{C} \mapsto \mathcal{G}$ is a total non-injective function which maps a criterion to a goal as an initialization criterion of that goal,
- $\mathcal{T}_\mathcal{F} \subsetneq \mathcal{T}$ is a set of process tasks, $\mathcal{F}$ is a set of subprocesses,
- $\phi : \mathcal{T}_\mathcal{F} \mapsto \mathcal{F}$ is a total function which maps a process task to a subprocess,
- $\delta : \mathcal{T} \cup \mathcal{G} \cup \mathcal{S} \mapsto \mathcal{S}$ is a partial non-injective function which maps a task, goal, or stage to a parent stage,
- $\mathcal{ED} = \{mandatory, optional\}$ is a set of execution directives for tasks, goals, and stages,
- $\rho : \mathcal{T}_a \cup \mathcal{G} \cup \mathcal{S} \mapsto \mathcal{ED}$ is a total non-injective surjective function which maps a task, goal, or stage to an execution directive,
- $\mathcal{CE} = \{immediate, listening\}$ is a set of evaluation modes for entry criteria where immediate is possible for $e \in \mathcal{E}$ iff $\exists d \mid d = (d_s, d_t) \wedge d_t = e \wedge (\zeta_\mathcal{E}(e) = t \mid t \in \mathcal{T})$,
- $\sigma_{\mathcal{CE}} : \mathcal{E} \mapsto \mathcal{CE}$ is a total non-injective surjective function which maps an entry criterion to an evaluation mode,

5. CASE ELEMENT REDUCTION

Since a specification usually contains merely a small amount of case elements (cf. [7, 6, 26]), those elements that are not contained in a specification are candidates for removal to reduce model complexity. However, since case element reductions might have a disturbing impact on the evaluation of formulas containing next operators, the approach is limited to next-free temporal logic formulas. The resulting reduced model must preserve the behavior of the original model, so not every case element which is not part of the specification can be removed. In this section we will discuss the *Case Element Reduction* approach which is taken as the first of two reduction steps (the second reduction step will be discussed in Section 6).

[2]The entry criterion of a goal is also called *completion criterion*.

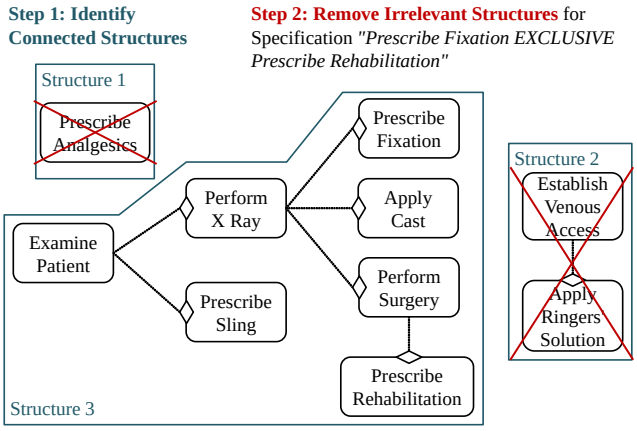


Figure 3: Example for *Reduction of Structures*

5.1 Reduction of Non-Relevant Structures

As a first reduction of case elements, those connected structures can be removed that do not contain any of the case elements of the specification formula. For this reason, a graph called *flattened case graph* is created in a first step:

A flattened case graph $G_{\mathcal{M}f} = (V, E)$ is a directed graph representation of $\mathcal{M}$, where $V = \mathcal{T} \cup \mathcal{G} \cup \mathcal{S} \cup \mathcal{C} \cup \mathcal{D}$ is a set of vertices. $E = E_e \cup E_x \cup E_{df} \cup E_{dt}$ is a set of edges where $E_e = \{(e, \zeta_\mathcal{E}(e)) \mid e \in \mathcal{E}\}$, $E_x = \{(\zeta_\mathcal{X}(x), x) \mid x \in \mathcal{X}\}$, $E_{df} = \{(f, d) \mid d = (f, t) \in \mathcal{D}\}$ and $E_{dt} = \{(d, t) \mid d = (f, t) \in \mathcal{D}\}$.

By this, structures of a case model that are connected through dependencies are identified. In a next step, those structures that do not contain elements of the specification formula and are just contained in a stage but not dependent in any other form on a stage can be removed from the case model because they do not have any influence on the verification of the given specification. Consequently, all elements V_s of a connected component s of $G_{\mathcal{M}f}$ are removed from $\mathcal{M}$ iff $\nexists v \mid v \in V_s \wedge (v \in specification \vee v \in \mathcal{S})$.

In Figure 3, *Reduction of Structures* is applied to the motivational example. After the identification of connected structures, all elements contained in structures that do not contain the elements of the specification are removed.

This reduction can be considered as a *macro reduction* because it is able to remove larger structures of a case model. After this first reduction step, it makes sense to perform *micro reductions* that try to decrease the number of case elements in remaining structures. We propose two micro reduction techniques, namely *Rear Reduction* and *Melting Reduction*.

5.2 Reduction of Non-Relevant Rear Elements

A rear reduction of a task, stage or goal $tgs \in \mathcal{T} \cup \mathcal{G} \cup \mathcal{S}$ is performed as follows:

- A task or goal $t \in \mathcal{T}$ is removed from $\mathcal{M}$ iff $t \notin specification \wedge \rho(t) \neq mandatory \wedge (\nexists x \mid (x \in \mathcal{X} \wedge \zeta_\mathcal{X}(x) = t \wedge (\nexists d \mid d = (d_s, d_t) \in \mathcal{D} \wedge d_s = x))) \wedge (\nexists d \mid d = (d_s, d_t) \in \mathcal{D} \wedge d_s = t)$.
- A goal $g \in \mathcal{G}$ is removed from $\mathcal{M}$ iff $g \notin specification \wedge \rho(g) \neq mandatory \wedge (\nexists d \mid d = (d_s, d_t) \in \mathcal{D} \wedge d_s = g)$.

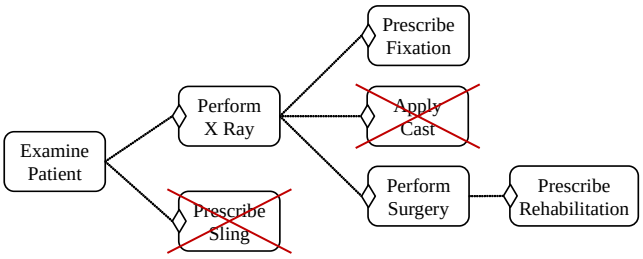


Figure 4: Example for *Rear Reduction*

- A stage $s \in \mathcal{S}$ is removed from $\mathcal{M}$ iff $s \notin specification \wedge \rho(s) \neq mandatory \wedge (\nexists x \mid (x \in \mathcal{X} \wedge \zeta_{\mathcal{X}}(x) = s \wedge (\nexists d \mid d = (d_s, d_t) \in \mathcal{D} \wedge d_s = x))) \wedge (\nexists d \mid d = (d_s, d_t) \in \mathcal{D} \wedge d_s = s) \wedge (\nexists tgs \mid tgs \in \mathcal{T} \cup \mathcal{G} \cup \mathcal{S} \wedge \delta(tgs) = s)$.

A rear reduction by removal of tgs also causes the removal of all $d \mid d = (d_s, d_t) \in \mathcal{D} \wedge \zeta_{\mathcal{E}}(d_t) = tgs$, $c \mid c \in \mathcal{C} \wedge (\zeta_{\mathcal{X}}(c) = tgs \vee \zeta_{\mathcal{E}}(c) = tgs)$ and $d \mid d = (d_s, d_t) \in \mathcal{D} \wedge d_t = tgs$ from the case model $\mathcal{M}$.

That means, if no case element is dependent on a specific task or goal, and if this specific task or goal is not part of the specification and not mandatory, then it can be removed. Stages are treated similarly, with the additional condition that the stage must not contain any elements.

In Figure 4, *Rear Reduction* is applied to the case model from Figure 3. Tasks *Prescribe Sling* and *Apply Cast* which are not contained in the specification "*Prescribe Fixation EXCLUSIVE Prescribe Rehabilitation*" and do not have a successor are removed since they do not have any influence on the result of the model checking.

5.3 Melting

A melting reduction between two case elements e_1 and e_2 is performed iff

- $\neg(e_1 \in specification \wedge e_2 \in specification)$, and
- $\nexists d \mid d = (d_s, d_t) \in \mathcal{D} \wedge d_s = e_1 \wedge (d_t \neq e_2 \vee \zeta_{\mathcal{E}}(d_t) \neq e_2)$, and
- $\nexists x \mid x \in \mathcal{X} \wedge \zeta_{\mathcal{X}}(x) = e_1 \wedge (\exists d \mid d = (d_s, d_t) \in \mathcal{D} \wedge d_s = x \wedge (d_t \neq e_2 \vee \zeta_{\mathcal{E}}(d_t) \neq e_2))$,
- $\nexists d \mid d = (d_s, d_t) \in \mathcal{D} \wedge d_t = e_2 \wedge (d_s \neq e_1 \vee \zeta_{\mathcal{X}}(d_s) \neq e_1)$, and
- $\nexists e \mid e \in \mathcal{E} \wedge \zeta_{\mathcal{E}}(e) = e_2 \wedge (\exists d \mid d = (d_s, d_t) \in \mathcal{D} \wedge d_t = e \wedge (d_s \neq e_1 \vee \zeta_{\mathcal{X}}(d_s) \neq e_1))$,

and

- $\exists d \mid d = (d_s, d_t) \in \mathcal{D} \wedge d_s = e_1 \wedge (d_t = e_2 \vee \zeta_{\mathcal{E}}(d_t) = e_2)$, or
- $\exists x \mid x \in \mathcal{X} \wedge \zeta_{\mathcal{X}}(x) = e_1 \wedge (\exists d \mid d = (d_s, d_t) \in \mathcal{D} \wedge d_s = x \wedge (d_t = e_2 \vee \zeta_{\mathcal{E}}(d_t) = e_2))$,

and

- $(e_1 \in \mathcal{T} \wedge e_2 \in \mathcal{T}) \vee (e_1 \in \mathcal{G} \wedge e_2 \in \mathcal{G}) \vee ((e_1 \in \mathcal{S} \wedge e_2 \in \mathcal{S}) \wedge (\nexists s \mid \delta(s) = e_1 \vee \delta(s) = e_2))$, and
- $\rho(e_1) = \rho(e_2)$.

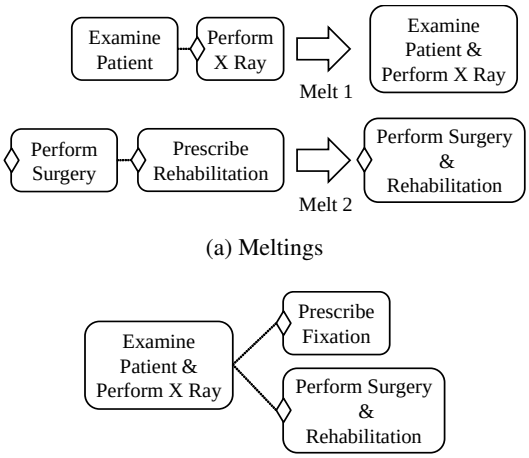


Figure 5: Example for *Melting Reduction*

Then, the melting reduction can be realized as follows:

1. All $rd \mid rd = (rd_s, rd_t) \in \mathcal{D} \wedge rd_s = e_1$ are to be removed from $\mathcal{M}$ and substituted by all $d \mid d = (d_s, d_t) \in \mathcal{D} \wedge d_s = e_2$, so that $d_s = e_1$, and
2. all $rx \mid rx \in \mathcal{X} \wedge \zeta_{\mathcal{X}}(rx) = e_1$ are to be removed from $\mathcal{M}$ and substituted by all $x \mid x \in \mathcal{X} \wedge \zeta_{\mathcal{X}}(x) = e_2$, so that $\zeta_{\mathcal{X}}(x) = e_1$, and
3. $e_1.label = \{e_1.label, e_2.label\}$, and e_2 is removed from $\mathcal{M}$.

The idea behind the melting reduction is the aggregation of elements, which we illustrate in Figure 5 by applying *Melting Reduction* to the case model from Figure 4. For the verification of '*Prescribe Fixation EXCLUSIVE Prescribe Rehabilitation*, *Examine Patient* and *Perform X Ray* as well as *Perform Surgery* and *Prescribe Rehabilitation* can be aggregated to joint activities.

6. CONDITION REDUCTION

To avoid the explicit consideration of conditions and their data attributes, we propose to precompute the possible combinations of criteria that are satisfiable altogether at the completion of a case element. When a case element (i.e., a task, goal or stage) is completed, it depends on the current state of data which criteria are fulfilled. In a first step, all the possible combinations of exit criteria that are satisfiable at once are computed (Algorithm 1). In Line 5 of Algorithm 1 the power set of all exit criteria is created. Line 6 defines a loop to iterate over this power set. Line 9 checks whether this combination of criteria is satisfiable at once (Algorithm 3). If this is the case, then the combination is added to a set, which is returned by this function (Line 9). Algorithm 2 is similar to Algorithm 1 as it computes which other combinations that include dependent criteria are satisfiable with the already identified exit combinations, so power sets are created and Algorithm 3 is called again to find all possible criteria combinations. Please note that power sets grow exponential with the size of their sets. However, the created sets of interdependent criteria are rather small (i.e., in case models very few criteria are interdependent). Consequently, the computation times remain acceptable, and there is a large performance improvement compared to the explicit consideration of conditions in model checking.

Algorithm 1 Compute all possible satisfiable combinations of exit criteria at the completion of a case element

1: **function** COMPUTEPOSSIBLEEXITCRITERIACOMBINATIONS($ce \in \mathcal{T} \cup \mathcal{S}$)
2: initialize satisfiableExitCriteriaCombinations
3: **if** $\exists \zeta_{\mathcal{X}}(x) = ce$ **then**
4: allExitCriteria = $\{x \mid x \in \zeta_{\mathcal{X}}(x) = ce\}$
5: powerSet = $\mathfrak{P}$(allExitCriteria)
6: **for all** cs in powerSet **do**
7: **if** $|cs| > 0$ **then**
8: complement := allExitCriteria \ cs
9: **if** IsSatisfiableCombination(cs, complement, allExitCriteria) **then**
10: satisfiableExitCriteriaCombinations.add(cs)
11: **return** satisfiableExitCriteriaCombinations

Algorithm 2 Compute all possible satisfiable combinations of exit criteria and dependent criteria at the completion of a case element

1: **function** COMPUTEPOSSIBLEEXITANDDEPENDENTCRITERIACOMBINATIONS($ce \in \mathcal{T} \cup \mathcal{S}$)
2: initialize satisfiableExitAndDependentCriteriaCombinations
3: satisfiableExitCriteriaCombinations := ComputePossibleExitCriteriaCombinations(ce)
▷ cf. Algorithm 1 for function 'ComputePossibleExitCriteriaCombinations'
4: **for all** satisfiableExitCriteria in satisfiableExitCriteriaCombinations **do**
5: initialize criteriaDependentOnCeOrExitCombination
6: criteriaDependentOnCeOrExitCombination.addAll($\forall c \mid c \in \mathcal{C} \wedge \exists(ce, c) \in \mathcal{D}$);
7: **for all** x in satisfiableExitCriteria **do**
8: **for all** $d = (d_s, d_t) \mid d \in \mathcal{D} \wedge d_s = x \wedge d_t \in \mathcal{C}$ **do**
9: criteriaDependentOnCeOrExitCombination.add(d_t)
10: **for all** $d = (d_s, d_t) \mid d \in \mathcal{D} \wedge d_s = ce \wedge d_t \in \mathcal{C}$ **do**
11: criteriaDependentOnCeOrExitCombination.add(d_t)
12: powerSet = $\mathfrak{P}$(criteriaDependentOnCeOrExitCombination)
13: **for all** cs in powerSet **do**
14: initialize satisfiableSet
15: satisfiableSet.addAll(satisfiableExitCriteria)
16: satisfiableSet.addAll(cs)
17: initialize allSet
18: allSet.addAll(satisfiableExitCriteria)
19: allSet.addAll(criteriaDependentOnCeOrExitCombination)
20: unsatisfiableSet := criteriaDependentOnCeOrExitCombination \ cs
21: **if** IsSatisfiableCombination(satisfiableSet, unsatisfiableSet, allSet) **then**
▷ cf. Algorithm 3 for function 'IsSatisfiableCombination'
22: satisfiableExitAndDependentCriteriaCombinations.add(cs)
23: **return** satisfiableExitAndDependentCriteriaCombinations

Algorithm 3 Compute whether a combination of criteria is satisfiable

1: **function** ISSATISFIABLECOMBINATION($satisfiableSet \subsetneq \mathcal{C}, unsatisfiableSet \subsetneq \mathcal{C}, set \subsetneq \mathcal{C}$)
2: dataEnumerationMap := CreateEnumerationValues(set)
3: dataModel := CreateDataModel(dataEnumerationMap)
4: specification := $EF(\bigwedge_{c \in satisfiableSet}(c.bf) \wedge \bigwedge_{c \in unsatisfiableSet}(\neg c.bf))$
5: **return** performVerification(dataModel, specification)

Algorithm 3 creates a model (in Lines 2 & 3) of the criteria combination (Line 3) and checks whether this model meets the CTL formula in Line 4. For the model, it is necessary to analyze the conditions of the criteria to build up a proper enumeration set which is an abstraction of the possible values of the variables used in conditions. For example, for integer attributes it is sufficient to add the preceding and the succeeding integer values to the enumeration set. Attributes of other data types may require a more comprehensive treatment. For example, the treatment of strings is dependent on the string functions that a condition may contain.

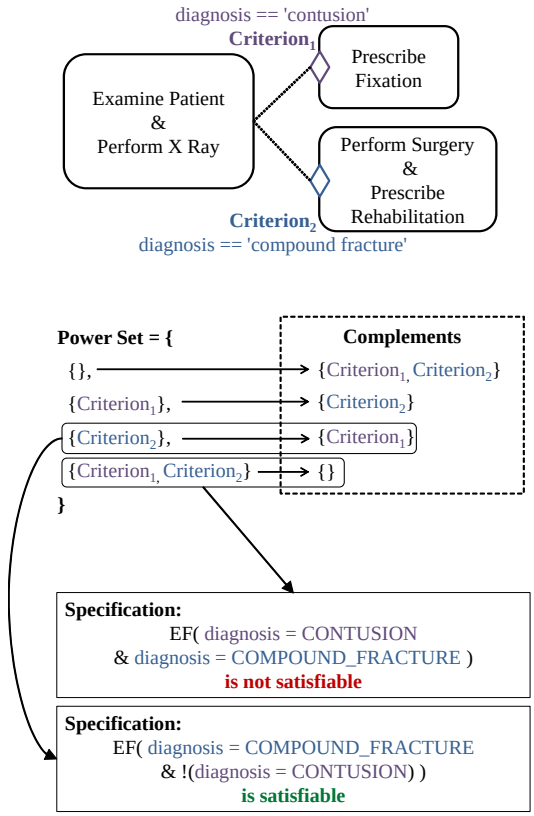


Figure 6: Example for *Condition Reduction*

In Figure 6, *Condition Reduction* is applied to the model from Figure 5. To find out, what the possible combinations of criteria are at the completion of *Examine Patient & Perform X Ray*, the power set of $\{\text{Criterion}_1, \text{Criterion}_2\}$ and the complement of each set of the power set is created. For each element of the power set and its complement, Algorithm 3 is called. The data model contains just a single attribute, namely *diagnosis* which has an enumeration set {COMPOUND_FRACTURE, CONTUSION, OTHER}, where OTHER is added representatively for values different from those present in conditions. Figure 6 illustrates the evaluation of two criteria combinations in greater detail. The question is whether it is possible at the completion of *Examine Patient & Perform X Ray* that (1) Criterion_2 is satisfied and Criterion_1 is not satisfied, (2) both Criterion_1 and Criterion_2 are satisfied. Since the CTL formula for (1) is satisfiable, it is a possibility that Criterion_1 is met while Criterion_2 is not met at the completion *Examine Patient & Perform X Ray*. Consequently, this is a satisfiable combination. The CTL specification for (2) is not satisfiable since the conditions of the two entry criteria are contradictory. Consequently, Criterion_1 and Criterion_2 are not satisfiable at the same time, so this is not a satisfiable combination.

7. EXPERIMENTAL RESULTS

For the evaluation of the proposed approach, a representative set of LTL (Linear Temporal Logic) patterns and case models is taken into account. As sources for LTL patterns, we make use of the property specification patterns by Dwyer et al. [6, 1] and the compliance patterns by Elgammal et al. [7]. Sources for case models are real-world process models that are either available from customers or from public sources. Here it is important to consider the degree of structuredness and size of such a model. For the evaluation of the approach, we select the largest to us available case models, one from a sales company (cf. [25]) with a high degree of structuredness ($\frac{|\mathcal{D}|}{|\mathcal{T} \cup \mathcal{G} \cup \mathcal{S}|} \approx 1.07$) and a total size of $|\mathcal{T} \cup \mathcal{G} \cup \mathcal{S} \cup \mathcal{C} \cup \mathcal{D}| = 80$ (where $|\mathcal{T}| = 23$, $|\mathcal{G}| = 3$, $|\mathcal{S}| = 3$, $|\mathcal{C}| = 20$, $|\mathcal{D}| = 31$) which we will refer to as *highly-structured case*, and another from health care (cf. [11][3]) with a medium degree of structuredness ($\frac{|\mathcal{D}|}{|\mathcal{T} \cup \mathcal{G} \cup \mathcal{S}|} = 0.5$) and total size of $|\mathcal{T} \cup \mathcal{G} \cup \mathcal{S} \cup \mathcal{C} \cup \mathcal{D}| = 75$ (where $|\mathcal{T}| = 26$, $|\mathcal{G}| = 6$, $|\mathcal{S}| = 4$, $|\mathcal{C}| = 21$, $|\mathcal{D}| = 18$) which we will refer to as *semi-structured case*.

The prototype is written in Java. It uses JGraphT[4] for graph-based parts of the approach, and it invokes NuSMV[5] (version 2.5.4) for model checking. The experiment was carried out on a common notebook computer with 8 GB RAM, Intel i5-4200U CPU (up to 2.6 GHz) and SATA II SSD on Windows 7, as we wanted to test our approach in the usual setting of a software developer or knowledge worker. The data of this evaluation was collected from 30000 model checks on the *semi-structured case* and *structured case*. LTL specifications for those model checks are based on 15 distinct temporal logic patterns [6, 7]. For each combination of pattern and case model, 1000 LTL formulas are generated from the states of tasks, goals and stages of the case model to create a huge set of specifications, which simulates verification runs carried out by a user in a large quantity.

Figure 7 shows the size reduction that is achieved by the *Case Element Reduction* step. The approach performs better on the semi-structured model than on the highly structured model. The computation of the *Case Element Reduction* step is finished between 0.1 and 0.5 milliseconds. *Condition Reduction* takes several orders of magnitude longer for the highly-structured case (about 500 milliseconds) than for the semi-structured case (0.005 milliseconds). The majority of the overall computation time is spent on model checking (Figure 8). Here, the semi-structured case model is verified within few seconds (in most cases even in a fraction of a second). It is not at all surprising that verifying the semi-structured case takes less time because the reduction techniques have a higher level of efficiency on semi-structured models. Obviously, more structures must remain in highly-structured models to preserve its semantics. Most of the verification runs terminate within 10 seconds, but there are also runs that take up to about 1000 seconds (i.e., when more structures must be preserved due to the properties of the LTL formula), which is still a good results when we consider the computational expensiveness of model checking in general.

We deliberately do not compare against the situation in which the proposed reductions are not applied because without the reduction techniques, the state space explodes and results are not to be expected within acceptable response times.

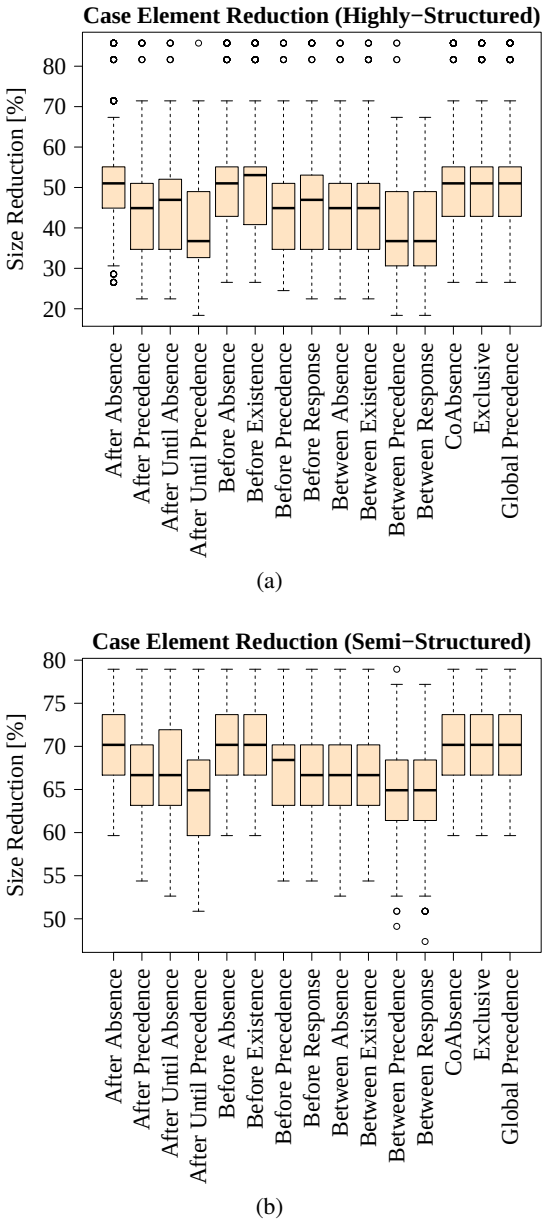
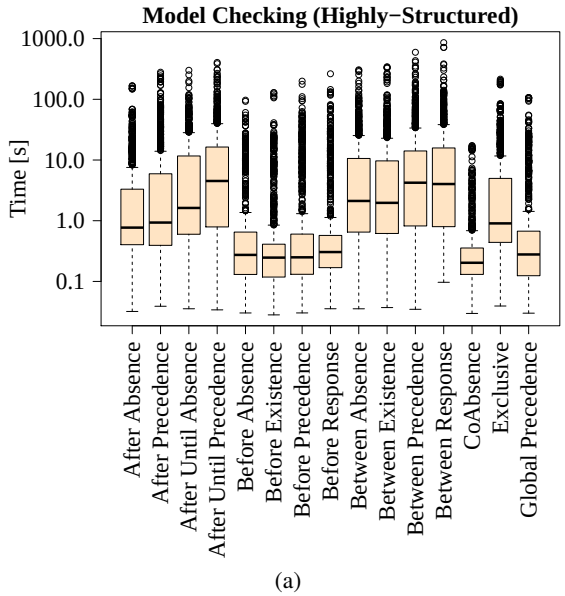


Figure 7: Achieved size reduction after applying *Case Element Reduction* to the (a) *Highly-Structured Case* and (b) to the *Semi-Structured Case*

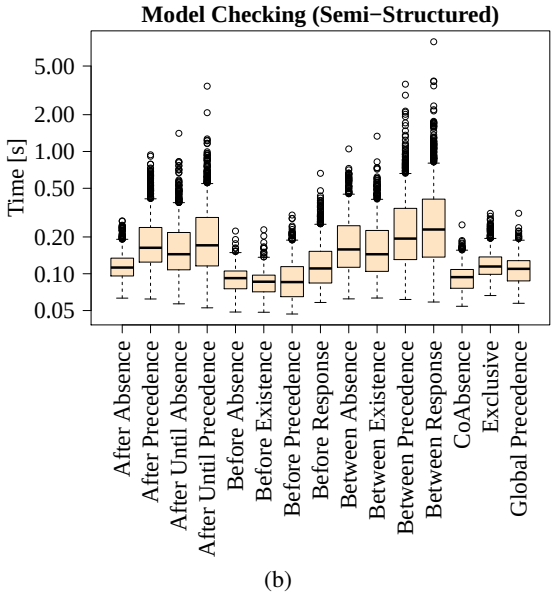
8. DISCUSSION

Although the proposed approach has a strong focus on ACM case models, it is to a large extent applicable to CMMN (Case Management Model and Notation [19]) models. However, not all parts of the CMMN standard are yet considered (e.g., flow-driven subprocesses). The presented work focuses on case management model elements and their semantics intentionally. Flow-driven subprocesses (e.g., BPMN processes) of case models are unfortunately out of scope of this paper. Nevertheless, the approach is capable to include flow-driven subprocesses of case models as well. This in-

[3]We slightly had to adapt the model from [11] because some goals did not have any completion criteria, so we added criteria to those goals.
[4]http://jgrapht.org
[5]http://nusmv.fbk.eu/



(a)



(b)

Figure 8: Computation time of *Model Checking* for (a) the *Highly-Structured Case* and (b) the *Semi-Structured Case*

volves applying reduction techniques for flow-driven processes [2], and conditions, such as present as guard conditions after exclusive and inclusive gateways in BPMN, can be precomputed and reduced in similar manner as described in Section 6.

By applying the approach to case models, it becomes possible to verify their behavior at design time. Nonetheless, it is impossible to guarantee compliance of a case instance by model checking of case models at runtime because runtime-specific circumstances, such as ad hoc actions and data adaptations, might introduce non-compliance. Consequently, it is not sufficient to provide only a tool support for design time verification. To keep cases compliant, runtime monitoring [16] of case instances is required. In particular, adequate tools must support business users to stay compliant.

Model checking is a powerful verification technique. However, a well-known limitation of model checking is the high demand on computational resources. If a problem grows too large in size, then model checking will become infeasible due to long computations times. In this work, we propose reduction techniques to decrease the size of the problem, which makes it possible to verify real-world case models that are realistic in size. Consequently, that general limitation of model checking is mitigated by the proposed reduction techniques. We consider further improving the performance as an important topic for future work.

The stage element enables nesting in case management models, and the proposed reduction approach implicitly considers nesting by performing reductions on each level. For that reason nested case models do not require any additional treatment.

9. RELATED WORK

Many related studies on the verification of business processes focus on flow-driven business processes, such as UML activity diagrams and BPMN models. Eshuis proposes a model checking approach using the NuSMV model checker for the verification of UML activity diagrams [8]. Kherbouche et al. use the model checker SPIN to find structural errors in BPMN models [13]. Sbai et al. use SPIN for the verification of workflow nets, which are Petri nets representing a workflow [22]. Köhler et al. describe a process by means of an automaton and check this automaton by NuSMV [14]. An approach presented by Awad et al. aims at checking compliance of flow-driven business process models using the visual query language BPMN-Q to describe constraints and performing model checking to assure constraints are met [2]. This approach reduces the complexity of BPMN models by analyzing LTL specifications before state space generation. Aforementioned approaches apply model checking for verification of business processes, but there also exist alternative approaches. For example, Raedts et al. propose the transformation of models such as UML activity diagrams and BPMN2 models to Petri nets for verification with Petri net analyzers [21]. Declarative workflow approaches (such as Declare [20], which has its origin in pattern-based LTL) are conceptually closer to the specifications for model checking (i.e., LTL or CTL) in our approach than the case model itself.

In 2014, the CMMN (Case Management Model and Notation) standard is released in version 1.0 by the OMG (Object Management Group) as "a common meta-model and notation for modeling and graphically expressing a Case, as well as an interchange format for exchanging Case models among different tools". Recent research indicates that CMMN is suitable for modeling knowledge-intensive processes and that the essential structural concepts of ACM cases are covered or can be realized by CMMN elements [15, 17, 5]. CMMN draws many influences, such as case handling [27], business artifacts [18], and the GSM language [12] for programming artifact-centric systems. Despite similarity between GSM and ACM case models, there are many conceptual differences (cf. Section 4 & [24]). Gonzalez et al. propose a specialized model checker for the GSM language [10]. By using the proposed state space reduction techniques, our approach enables the verification of case models by non-specialized model checkers (e.g., NuSMV).

In summary, the verification of ACM case models has not yet been sufficiently addressed in existing studies. Due to the increasing industry adoption of ACM, this topic is highly relevant, not only from a purely academic but also from a practical point of view.

10. CONCLUSION & FUTURE WORK

This paper presents a model checking approach for ACM case models. In particular, it discusses several techniques that aim at reducing the state space required for efficient model checking of case models (answer to RQ1). Reductions are on the one hand concerned with making use of a given specification to remove elements from a case model that are not required for the verification run, and on the other hand, conditions present in a case model are considered in an abstracted manner in the actual verification run. The experimental evaluation based on 30000 model checking runs on two real case models that are realistic in size and representative for different degrees of structuredness shows an overall good performance of the approach (answer to RQ2). As models in ACM are predominantly semi-structured, the fast computation times that can be achieved through the proposed reduction techniques are of great interest for a potential industry adoption of the proposed approach. In future work, it might be possible to further improve the performance by directly focusing on the specific properties of temporal pattern-based solutions (cf. e.g., [6]) instead of supporting arbitrary LTL and CTL formulas.

11. ACKNOWLEDGMENTS

The research leading to these results has received funding from the FFG project CACAO, no. 843461 and the WWTF, Grant No. ICT12-001.

12. REFERENCES

[1] Property Pattern Mappings for LTL. http://patterns.projects.cis.ksu.edu/documentation/patterns/ltl.shtml. Last accessed: December 1, 2016.

[2] A. Awad, G. Decker, and M. Weske. *BPM, Milan, Italy*, chapter Efficient Compliance Checking Using BPMN-Q and Temporal Logic, pages 326–341. Springer, 2008.

[3] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. Nusmv: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2, 2000.

[4] E. M. Clarke. *The Birth of Model Checking*, pages 1–26. Springer, 2008.

[5] C. Czepa, H. Tran, U. Zdun, S. Rinderle-Ma, T. Tran, E. Weiss, and C. Ruhsam. Supporting structural consistency checking in adaptive case management. In *CoopIS'15*, pages 311–319, October 2015.

[6] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *ICSE'99*, pages 411–420. ACM, 1999.

[7] A. Elgammal, O. Turetken, W.-J. van den Heuvel, and M. Papazoglou. Formalizing and appling compliance patterns for business process compliance. *Software & Systems Modeling*, 15(1):119–146, 2016.

[8] R. Eshuis. Symbolic model checking of uml activity diagrams. *ACM Trans. Softw. Eng. Methodol.*, 15(1):1–38, Jan. 2006.

[9] Forrester Research. The Forrester Wave™: Dynamic Case Management, Q1 2016.

[10] P. Gonzalez, A. Griesmayer, and A. Lomuscio. Verifying gsm-based business artifacts. ICWS '12, pages 25–32. IEEE Computer Society, 2012.

[11] N. Herzberg, K. Kirchner, and M. Weske. *Modeling and Monitoring Variability in Hospital Treatments: A Scenario Using CMMN*, pages 3–15. Springer, Cham, 2015.

[12] R. Hull, E. Damaggio, F. Fournier, M. Gupta, F. T. Heath, III, S. Hobson, M. Linehan, S. Maradugu, A. Nigam, P. Sukaviriya, and R. Vaculin. Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In *WS-FM'10*, pages 1–24. Springer, 2011.

[13] O. Kherbouche, A. Ahmad, and H. Basson. Using model checking to control the structural errors in bpmn models. In *RCIS'13*, pages 1–12, May 2013.

[14] J. Koehler, G. Tirenni, and S. Kumaran. From business process model to consistent implementation: a case for formal verification methods. In *EDOC'02*, pages 96–106, 2002.

[15] M. Kurz, W. Schmidt, A. Fleischmann, and M. Lederer. Leveraging cmmn for acm: Examining the applicability of a new omg standard for adaptive case management. S-BPM ONE '15, pages 4:1–4:9. ACM, 2015.

[16] L. T. Ly, F. M. Maggi, M. Montali, S. Rinderle-Ma, and W. M. van der Aalst. Compliance monitoring in business processes: Functionalities, application, and tool-support. *Information Systems*, 54:209 – 234, 2015.

[17] M. A. Marin, M. Hauder, and F. Matthes. Case management: An evaluation of existing approaches for knowledge-intensive processes. In *AdaptiveCM'15*, August 2015.

[18] A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Syst. J.*, 42(3):428–445, July 2003.

[19] OMG. Case Management Model and Notation (CMMN) Version 1.0. http://www.omg.org/spec/CMMN/1.0/PDF. Last accessed: December 1, 2016.

[20] M. Pesic and W. M. P. van der Aalst. A declarative approach for flexible business processes management. In *BPM Workshops*, BPM'06, pages 169–180, Berlin, Heidelberg, 2006. Springer-Verlag.

[21] I. Raedts, M. Petković, Y. S. Usenko, J. M. van der Werf, J. F. Groote, and L. Somers. Transformation of BPMN models for Behaviour Analysis. In *MSVVEIS*, pages 126–137. INSTICC press, 2007.

[22] Z. Sbai, A. Missaoui, K. Barkaoui, and R. Ben Ayed. On the verification of business processes by model checking techniques. In *ICSTE'10*, volume 1, pages V1–97–V1–103, Oct 2010.

[23] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, July 1985.

[24] D. Solomakhin, M. Montali, and S. Tessaris. Formalizing guard-stage-milestone meta-models as data-centric dynamic systems. Technical Report KRDB12-4, Free University of Bozen-Bolzano, 2012.

[25] K. D. Swenson, P. Nathaniel, M. J. Pucher, C. Webster, and A. Manuel. *How Knowledge Workers Get Things Done*, pages 155–164. Future Strategies Inc., 2012.

[26] W. M. P. van der Aalst and M. Pesic. *DecSerFlow: Towards a Truly Declarative Service Flow Language*, pages 1–23. Springer, 2006.

[27] W. M. P. van der Aalst and M. Weske. Case handling: A new paradigm for business process support. *Data Knowl. Eng.*, 53(2):129–162, May 2005.