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 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

1Completely 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.
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 criterion 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.

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

![Figure 1: Case model for the treatment of a suspected fracture](image1)

![Figure 2: Approach Overview](image2)
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 $(T_0, T, G, S, E, \mathcal{X}, \mathcal{C}, D, \zeta, \xi, \eta, \alpha, T_F, F, \phi, \delta, \mathcal{E}, \mathcal{D}, \rho, \mathcal{E}, \sigma_{\mathcal{CE}})$ where

- $T_0 \supseteq T \cup \bigcup_{e \in F} p.R$ is a set of all tasks, $T$ is a set of case tasks, $G$ is a set of goals, $S$ is a set of stages, $E$ is a set of dependencies of criteria, $\mathcal{X}$ is a set of criteria, $\mathcal{C} = E \cup \mathcal{X}$ is a set of criteria, $D$ is a set of dependencies (where $d = (d_s, d_t) \in D$ means that $d_t$ is dependent on $d_s$),
- $\zeta_e : E \rightarrow T \cup G \cup S$ is a total non-injective function which maps an entry criterion to a task, goal$^5$, or stage,
- $\zeta_x : \mathcal{X} \rightarrow T \cup G \cup S$ is a total non-injective function which maps an exit criterion to a task or stage,
- $\eta : G \rightarrow G$ is a partial non-injective function which maps a goal to a parent goal,
- $\alpha : \mathcal{C} \rightarrow G$ is a total non-injective function which maps a criterion to a goal as an initialization criterion of that goal,
- $T_F \subseteq T$ is a set of process tasks, $F$ is a set of subprocesses,
- $\phi : T_F \rightarrow F$ is a total function which maps a process task to a subprocess,
- $\delta : T \cup G \cup S \rightarrow S$ is a partial non-injective function which maps a task, goal, or stage to a parent stage,
- $\mathcal{E}D = \{\text{mandatory, optional}\}$ is a set of execution directives for tasks, goals, and stages,
- $\rho : T_0 \cup G \cup S \rightarrow \mathcal{E}D$ is a total non-injective surjective function which maps a task, goal, or stage to an execution directive,
- $\mathcal{CE} = \{\text{immediate, listening}\}$ is a set of evaluation modes for entry criteria where immediate is possible for $e \in E$ iff $\exists d = (d_s, d_t) \land d_t = e \land (\zeta_e(e) = t \lor t \in T)$,
- $\sigma_{\mathcal{CE}} : E \rightarrow \mathcal{CE}$ 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).

$^5$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 $G_{MF} = (V, E)$ is a directed graph representation of $\mathcal{M}$, where $V = T \cup G \cup S \cup \mathcal{C} \cup D$ is a set of vertices. $E = E_e \cup E_s \cup E_g \cup E_h$ is a set of edges where $E_e = \{(e, \zeta_e(e)) \mid e \in E\}$, $E_s = \{(\zeta_x(x), x) \mid x \in \mathcal{X}\}$, $E_g = \{(f, d) \mid d = (f, t) \in D\}$ and $E_h = \{(d, t) \mid d = (f, t) \in D\}$.

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 $G_{MF}$ are removed from $\mathcal{M}$ iff $\forall v \in V_s \forall \rho \in \mathcal{E}D \exists v = v \in V_s \land \rho \in \mathcal{E}D$.

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 $tgs \in T \cup G \cup S$ is performed as follows:

- A task or goal $t \in T$ is removed from $\mathcal{M}$ iff $t \notin \mathcal{E}D \land \rho(t) = \text{mandatory} \land \exists x \in \mathcal{X} \land \zeta_x(x) = t \land (\exists d_s (d = (d_s, d_t) \in D \land d_s = x))$.
- A goal $g \in G$ is removed from $\mathcal{M}$ iff $g \notin \mathcal{E}D \land \rho(g) = \text{mandatory} \land (\exists d_s (d = (d_s, d_t) \in D \land d_s = g))$. 
A rear reduction by removal of \(tgs\) also causes the removal of all \(d \mid d = (d_x, d_t) \in D \land d_x = e_1 \land (d_t \neq e_2 \lor \zeta_X(d_t) \neq e_2)\), and
\[
\exists x \mid x \in X \land \zeta_X(x) = e_1 \land (\exists d \mid d = (d_x, d_t) \in D \land d_x = x \land (d_t \neq e_2 \lor \zeta_X(d_t) \neq e_2)),
\]
and
\[
\exists d \mid d = (d_x, d_t) \in D \land d_x = e_2 \land (d_t \neq e_1 \lor \zeta_X(d_t) \neq e_1),
\]
and
\[
\exists d \mid d = (d_x, d_t) \in D \land d_x = e \land (d_t \neq e_1 \lor \zeta_X(d_t) \neq e_1).
\]

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 ∈ \( T \cup S \))
2: initialize satisfiableExitCriteriaCombinations
3: if \( 3 \xi_c(x) = ce \) then
4: allExitCriteria = \( \{ x \mid x \in \xi_c(x) = ce \} \)
5: powerSet = \( \wp(\text{allExitCriteria}) \)
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 ∈ \( T \cup S \))
2: initialize satisfiableExitAndDependentCriteriaCombinations
3: satisfiableExitCriteriaCombinations := ComputePossibleExitCriteriaCombinations(ce)
4: for all satisfiableExitCriteria in satisfiableExitCriteriaCombinations do
5: initialize criteriaDependentOnCeOrExitCombination
6: criteriaDependentOnCeOrExitCombination.addAll(\( \{ c \mid c \in C \land \exists (ce, c) \in D \} \))
7: for all x in satisfiableExitCriteria do
8: for all d = \( \{ d_x, d_t \} \mid d \in D \land d_x = x \land d_t \in C \) do
9: criteriaDependentOnCeOrExitCombination.add(d_t)
10: for all d = \( \{ d_x, d_t \} \mid d \in D \land d_x = ce \land d_t \in C \) do
11: criteriaDependentOnCeOrExitCombination.add(d_t)
12: powerSet = \( \wp(\text{criteriaDependentOnCeOrExitCombination}) \)
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(criterionDependentOnCeOrExitCombination)
20: unsatisfiableSet := criterionDependentOnCeOrExitCombination \ cs
21: if IsSatisfiableCombination(satisfiableSet, unsatisfiableSet, allSet) then
22: satisfiableExitAndDependentCriteriaCombinations.add(cs)
23: return satisfiableExitAndDependentCriteriaCombinations

**Algorithm 3** Compute whether a combination of criteria is satisfiable

1: function ISSATISFIABLECombination(satisfiableSet ⊆ C, unsatisfiableSet ⊆ C, set \( \subseteq C \))
2: dataEnumerationMap := CreateEnumerationValues(set)
3: dataModel := CreateDataModel(dataEnumerationMap)
4: specification := \( EF(\forall c \in \text{satisfiableSet}(\exists b.f(c))) \land \land \neg \exists c \in \text{unsatisfiableSet}(b.f(c)) \land \) performVerification(dataModel, specification)
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 (\(\frac{|D|}{|T \cup G \cup S \cup C \cup D|} \approx 1.07\)) and a total size of \(|T \cup G \cup S \cup C \cup D| = 80\) (where \(|T| = 23, |G| = 3, |S| = 3, |C| = 20, |D| = 31\)) which we will refer to as highly-structured case, and another from health care (cf. [11]) with a medium degree of structuredness (\(\frac{|D|}{|T \cup G \cup S \cup C \cup D|} = 0.5\)) and total size of \(|T \cup G \cup S \cup C \cup D| = 75\) (where \(|T| = 26, |G| = 6, |S| = 4, |C| = 21, |D| = 18\)) 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 result 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](image)

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/
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 computation 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