Towards Structural Consistency Checking in Adaptive Case Management

Christoph Czepa[1], Huy Tran[1], Uwe Zdun[1],
Thanh Tran Thi Kim[2], Erhard Weiss[2], and Christoph Ruhsam[2]

[1] Research Group Software Architecture, University of Vienna, Austria
{christoph.czepa,huy.tran,uwe.zdun}@univie.ac.at
[2] Isis Papyrus Europe AG, Maria Enzersdorf, Austria
{thanh.tran,erhard.weiss,christoph.ruhsam}@isis-papyrus.com

Abstract. This paper proposes structural consistency checking for Adaptive Case Management (ACM). Structures such as a hierarchical organization of business goals and dependencies among tasks are either created at design time or evolve over time while working on cases. In this paper, we identify structures specific to current ACM systems (as opposed to other BPM systems), discuss which inconsistencies can occur, and outline how to discover these issues through model checking and graph algorithms.

1 Introduction

In contrast to classical Business Process Management (BPM) which mainly utilizes predefined, rigid business processes, processes in knowledge-intensive domains (e.g., medical care, customer support, contract management) tend to be rather unpredictable and shall be handled in a more flexible manner. Flexibility and structuredness, however, do not necessarily contradict each other since structures can remain adaptable even at runtime. Adaptive Case Management (ACM) combines classical structural features such as process flows with new structures such as dependencies among tasks and business goal hierarchies [5, 11]. A high degree of flexibility increases the chance for potential errors or inconsistencies. For example, a new goal could be in conflict with an existing goal or a new dependency among two tasks could result in contradicting pre- and postconditions of these tasks.

In this paper, we illustrate exemplary structures that can be found in today's ACM solutions. Our study is based on our work with ISIS Papyrus[3], a state-of-the-art commercial ACM solution, several customer applications realized in Papyrus, our analysis of other solutions, and the Case Management Model and Notation (CMMN) standard[4] [8]. We discuss potential inconsistencies that can occur in such structures. Verification of classical process definitions has been discussed by a plethora of studies [1, 3, 4, 6, 7, 9, 10, 13], but we are not aware of any prior study related to structural consistency checking in the domain of ACM. We are working towards bridging this research gap.

[3] http://www.isis-papyrus.com
[4] http://www.omg.org/spec/CMMN/1.0/PDF/

2 Identification of Structures and Discussion of Inconsistencies

In this section, we illustrate structures that can be found in today's ACM solutions and discuss which inconsistencies can possibly occur in these structures. For the sake of illustration, we make use of the open CMMN standard instead of proposing a new or using a vendor-specific proprietary notation for the discussion of recurring structures in ACM and their inconsistencies. Please note that the focus is clearly on the structural concepts of ACM, not on a specific notation.

2.1 Goal Hierarchies

A goal defines what a knowledge worker has to achieve and thus, is directly linked to operational business targets and strategic objectives. Goals of a case can be structured hierarchically [5, 11]. On top of the hierarchy stands the main goal of a case which can be broken down into subgoals. Naturally, we would not want to pursue contradictory goals simultaneously. Therefore, consistency checks that reveal contradicting completion criteria of potentially simultaneously pursued goals would be helpful in the design of ACM cases.

Example 1. In the goal hierarchy depicted in Figure 1, a knowledge worker can work towards *Subgoal 1* and *Subgoal 2* or *Subgoal 1.1* and *Subgoal 2* at the same time, so their completion criteria must not be contradictory.

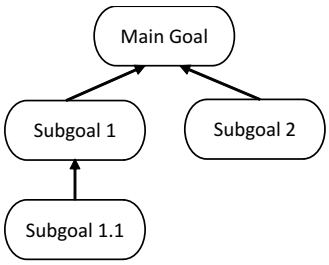

Fig. 1. Goal Hierarchy Example

2.2 Structures of Interdependent Criteria in the Case Structure

Starting to work on a case or the completion of a task usually leads to one or more other elements of the case to become accessed, depending on whether criteria along the way to such an element can be satisfied altogether. Every element of the case must be accessible. When an element is inaccessible due to an unsatisfiable combination of criteria, it is in any case an inconsistency that must be revealed. If an entry criterion is assigned to a goal, then it is also called a completion criterion. If an entry criterion (resp. exit criterion) is assigned to a task, then it is also called a precondition (resp. postcondition).

Example 2. Figure 2 extends the goal hierarchy presented in Figure 1 with tasks, subprocesses and substructures. In the following, we analyze for each element what combination of criteria must be satisfiable for its accessibility.

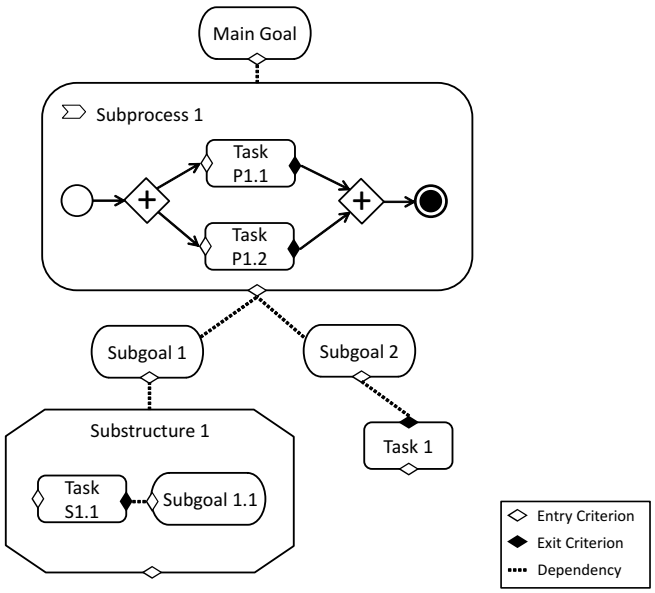

Fig. 2. Example Case

The precondition of *Task S1.1* must not contradict the entry criterion of *Substructure 1* because *Task S1.1* is the only task that can be started directly after having gained access to the substructure. *Subgoal 1.1* is only accessible from *Task S1.1* if the postcondition of the task is not contradictory to the completion criterion of the goal. To complete *Subgoal 1*, its completion criterion must be consistent with both the postcondition of *Task SS1.1* and the completion criterion of *Subgoal 1.1*. *Subgoal 2* is accessible if the postcondition of *Task 1* does not contradict the completion criterion of Subgoal 2. The process of *Subprocess 1* can only be started properly if the preconditions of *Task P1.1* and *Task P1.2* are consistent because they are logically and-connected due to the parallel split gateway ('process start structure'). Additionally, the structure composed of the entry criterion of the subprocess and the access structures of both *Subgoal 1* and *Subgoal 2* must be satisfiable ('subprocess access structure'). Consequently, the process start and subprocess access structure of *Subprocess 1* must also not be contradictory. To be able to fulfill the *Main Goal* of the case, the process finish structure composed of the postconditions of *Task P1.1* and *Task P1.2* must be consistent with the completion criterion of the *Main Goal*.

2.3 Dependency Loops

Dependencies can be arranged so as to create loops. There must be a way to enter this dependency loop.

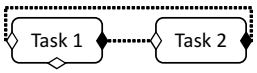

Fig. 3. Dependency Loop Example

Example 3. Figure 3 contains an example of a dependency loop. If we remove the second entry criterion of *Task 1* (which has no explicit dependency) the loop would lose its entry point and become inaccessible.

Please note that a dependency loop does not behave like a loop in flow-based business process language (e.g., BPMN) because a knowledge worker can decide whether or not to execute the next task and can, therefore, easily break out.

3 Outline of the Approach

In this section, we outline our approach for structural consistency checking of ACM case template. Our approach is based on model checking and graph algorithms. Model checking is used for the checking of goal hierarchies, structures of interdependent criteria and structured subprocesses. Graph algorithms are used for the discovery of loops and for expressing structures of interdependent criteria.

For the most part, structural consistency checking in ACM can be performed based on structures of interdependent criteria. This reduces the checking effort drastically because only parts of the model must be considered when checking for specific errors in the model. Flow-based subprocesses are the exception because the subprocess must also be verified as a whole.

$EF\ p$ is a Computation Tree Logic operator, and its meaning is that p must be satisfiable on at least one subsequent path. Model checking of this formula can be described as nondeterministic changes of data values until a fitting solution is found or all possibilities are exhausted. The most atomic checkable item in ACM is a single criterion c which must meet the specification $EF(c)$, followed by two criteria c_d and c'_d that are related to the same dependency d which must meet the specification $EF(c_d \wedge c'_d)$.

Example 4. Let us assume that both c_d and c'_d are criteria which are related to a string function 'starts with', where c_d demands that the first name of a person must start with 'T' and c'_d requires the first name to start with 'Chr'. Listing 1.1 contains code for checking the consistency of these two criteria. When we input this code into the model checker NuSMV [2], it detects that the specification is not satisfiable, so these interdependent criteria are contradicting each other.

```
VAR
Data#Person#First_Name#String : {Christoph, Christopher, Christian, Huy, Uwe, Thanh};

DEFINE Data#Person#First_Name#String#startsWith#Chr := (Data#Person#First_Name#String =
      Christoph) | (Data#Person#First_Name#String = Christian) | (Data#Person#
     First_Name#String = Christopher);

DEFINE Data#Person#First_Name#String#startsWith#T := (Data#Person#First_Name#String =
     Thanh);

CTLSPEC EF Data#Person#First_Name#String#startsWith#Chr & Data#Person#First_Name#String
     #startsWith#T
```

Listing 1.1. NuSMV code

For the generation and representation of logical expression trees that represent structures of interdependent criteria which then can be verified by model checking (generalization of Example 4), graph algorithms are leveraged in accordance with the analysis outlined in Section 2.2. Dependency loops can be discovered as strongly connected components in a graph representation of the case by Tarjan's algorithm [12] in linear time.

4 Related Work

Kherbouche et al. use model checking to find structural errors in BPMN2 models [6]. Eshuis proposes a model checking approach for the verification of UML activity diagrams [4]. Van der Aalst created a mapping of EPCs to Petri nets to check whether there are structural errors such as gateway mismatches [1]. Sbai et al. also use model checking for the verification of workflow nets (Petri nets representing a workflow) [10]. 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 [9]. Xiao et al. transform XPDL structures to Petri nets [13]. Köhler et al. describe a process by means of an automaton and check this automaton by model checking [7]. El-Saber et al. provide a formalization of BPMN and propose a verification through model checker [3]. These approaches have been created for flow-based business processes. None of the existing approaches considers the structural features that are present in Adaptive Case Management.

5 Conclusion and Future Work

This paper discusses structural consistency checking in the domain of Adaptive Case Management and proposes the use of model checking and graph algorithms as a potential solution. We identify several structural features, discuss possible inconsistencies, and propose preliminary concepts for discovering inconsistencies. Our study is based on our work with checking cases in ISIS Papyrus, a state-of-the-art ACM commercial product, as well as our analysis of other solutions and the CMMN standard. In future work, we intend to elaborate and formalize our checking approach and evaluate its scalability and performance.

Acknowledgements. The research leading to these results has received funding from the FFG project CACAO, no. 843461 and the Wiener Wissenschafts-, Forschungs- und Technologiefonds (WWTF), Grant No. ICT12-001.

References

[1] Van der Aalst, W.M.P.: Formalization and verification of event-driven process chains. Information and Software Technology 41(10), 639 – 650 (1999)
[2] Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: NUSMV: a new Symbolic Model Verifier. In: 11th Conf. on Computer-Aided Verification (CAV). pp. 495–499. Springer (July 1999)
[3] El-Saber, N., Boronat, A.: BPMN formalization and verification using Maude. In: 2014 Workshop on Behaviour Modelling-Foundations and Applications (BM-FA). pp. 1:1–1:12. ACM (2014)
[4] Eshuis, R.: Symbolic model checking of UML activity diagrams. ACM Trans. Softw. Eng. Methodol. 15(1), 1–38 (2006)
[5] Greenwood, D.: Goal-oriented autonomic business process modeling and execution: Engineering change management demonstration. In: 6th Intl. Conf. BPM. pp. 390–393. Springer (2008)
[6] Kherbouche, O., Ahmad, A., Basson, H.: Using model checking to control the structural errors in BPMN models. In: 7th Intl. Conf. on RCIS. pp. 1–12 (2013)
[7] Koehler, J., Tirenni, G., Kumaran, S.: From business process model to consistent implementation: a case for formal verification methods. In: 6th Intl. Conf. on EDOC. pp. 96–106 (2002)
[8] Kurz, M., Schmidt, W., Fleischmann, A., Lederer, M.: Leveraging cmmn for acm: Examining the applicability of a new omg standard for adaptive case management. In: 7th Int. Conf. on Subject-Oriented BPM. pp. 4:14:9. ACM, New York, NY, USA (2015)
[9] Raedts, I., Petković, M., Usenko, Y.S., van der Werf, J.M., Groote, J.F., Somers, L.: Transformation of BPMN models for Behaviour Analysis. In: MSVVEIS. pp. 126–137. INSTICC (2007)
[10] Sbai, Z., Missaoui, A., Barkaoui, K., Ben Ayed, R.: On the verification of business processes by model checking techniques. In: 2nd Int. Conf. on ICSTE. vol. 1, pp. 97103 (Oct 2010)
[11] Stavenko, Y., Kazantsev, N., Gromoff, A.: Business process model reasoning: From workflow to case management. Procedia Technology 9(0), 806 – 811 (2013)
[12] Tarjan, R.: Depth first search and linear graph algorithms. SIAM Journal on Computing (1972)
[13] Xiao, D., Zhang, Q.: The implementation of xpdl workflow verification service based on saas. In: Int. Conf. on ICSS. pp. 154158 (May 2010)