A Workflow for Healthcare Systems via OCL and SMT Solving

Short Paper

Hao Wu
Department of Computer Science
Maynooth University
haowu@cs.nuim.ie

Laure Hinsberger
Department of Computer Science
Université de Lorraine
laure.hinsberger@telecomnancy.net

Joseph Timoney
Department of Computer Science
Maynooth University
jtimoney@cs.nuim.ie

ABSTRACT

Modelling can be applied to all aspects of healthcare systems but one area in particular, clinical pathways, are of great interest currently due to many flow-oriented issues that are well-documented in the media. These pathways typically describe sequences of sorting and treatment activities such as surgical procedures or the care process for managing injuries, for example bone fractures. Previous efforts in using modelling languages have not been promoted as being highly generalizable nor have emphasised the inclusion of constraints defining rules for particular treatment activities. In this paper, we propose a workflow for building flexible models for healthcare systems by exploiting the combination of UML, OCL, and SMT solving. This paper serves as an exposition of an idea that can be developed into a more complete framework that could be used to create workflow models for improving the efficiency and safety of more complex clinical activities. A good application would be to tackle the prevalent problems of emergency departments and some of the challenges in this respect are discussed.

KEYWORDS

Healthcare service, UML, OCL, SMT

ACM Reference Format:
Hao Wu, Laure Hinsberger, and Joseph Timoney. 2018. A Workflow for Healthcare Systems via OCL and SMT Solving: Short Paper. In *SEHS'18: IEEE/ACM International Workshop on Software Engineering in Healthcare Systems , May 27, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3194696.3194704

1 INTRODUCTION

Healthcare systems across the world are coming under enormous strain due to a combination of factors. These include rising life expectancy that has led to greater numbers seeking help along with more complicated treatment procedures. The situation worsens if individuals are suffering from multiple medical conditions. The use of computerization to support traditional manual systems for administration has introduced benefits of efficiency and cost reduction. However, the ever-growing medically complex and tightening regulatory environment that surrounds these healthcare systems means that the level of automation must keep pace. Many challenging problems exist. One in particular is that if rigorous analysis is to be used to improve healthcare systems, how can correct software models be built that will *formally* capture the rules and regulations associated with healthcare activities and services? Achieving this means that health care staff and patients can completely place their trust in their application.

In this paper, we pursue this challenge by proposing a Model-Driven Engineering (MDE) based workflow for building healthcare systems. MDE is an ever-growing methodology for designing different kinds of systems. It is typically tailored to address software engineering requirements related to productivity, flexibility and reliability by using multiple types of models at different stages of a system design. These models are used to capture a process or a structural aspect of a system and therefore can also be applied to any system including healthcare. In fact, the idea of using MDE to model health care services is not new. A variety of different types of models have been proposed with example applications including care monitoring, community care, collaborative care, and dynamic healthcare checklists [2, 7, 11, 13, 16].

However, these models are limited by the fact that they do not take necessary constraints into account. We believe that the use of models excluding constraints are *formally* insufficient. This inhibits *verifying* the correctness of a model. Our approach to building healthcare systems distinguishes itself from others by specifying constraints in Object Constraint Language (OCL) and solving these constraints using a Satisfiability Modulo Theories (SMT) solver. The combination of OCL and a SMT solver in our workflow gives us two advantages. First, it allows us to specify constraints without introducing ambiguities since OCL is based on first-order logic (FOL). Second, with recent advances in SMT solvers, solving/verifying a variety of different kinds of constraints in an efficient manner is now possible. Particularly, one can treat an SMT solver as a black-box engine. This enables the automation of the proving/disproving of the correct behaviours of a system with minimal effort.

2 OUR WORKFLOW

Our workflow for modelling a healthcare system consists of four main steps as depicted in Figure 1. First, medical experts analyse documents specifying rules and regulations for particular clinical pathways that are written in natural language. At this step, they

SEHS'18, May 27, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5734-0/18/05…$15.00
https://doi.org/10.1145/3194696.3194704

focus on comprehending the semantics of the documents and extracting the necessary information that is interpreted as entities, relationships (among them), conditions and constraints. Once the semantics are well understood, an appropriate UML model such as a class diagram can be established. More importantly, a set of mandatory constraints can also be specified in the form of either class invariants or operational contracts in OCL. The third step requires users to translate the complete model along with the constraints into a set of logic sentences that can be solved by SMT solvers. This step can now be done via many tools. For example, our previous research in verifying UML models allows us to automatically translate a model annotated with OCL constraints into first-order sentences [21, 22]. The final step of our workflow is to use an SMT solver to verify the translated logic sentences and produce either an instance of the model or a counterexample that contradicts the semantics captured by the model.

2.1 Fracture Treatment Model

In this section, we use a clinical pathway for fracture treatment presented in [6] as an example to illustrate our workflow [1]. We first read, analyse and interpret the documents ourselves to identify the relationships between different entities and necessary constraints. For example, a fracture can be typically treated using: *cast*, *repositioning*, *surgery* and *sling*. Thus, it is natural to identify those as 4 entities. Similarly, for the operations described in each sentence in the document we identify the subject, verb and object. We then map a verb to an operation call, an object to a parameter and a subject to an entity that contains an operation call [2]. For example, cast treatment applies a *cast* to the patient. We model *cast* as an entity, *apply* as an operation call that is contained in *cast* and *patient* as a parameter of *apply*.

We then build a UML class diagram for fracture treatment. This diagram is shown in Figure 2 along with a set of constraints. The diagram consists of 7 classes and 3 enumeration types. In particular, this UML class diagram models a relationship (*FractureTreatment*) between *FracturePatient* and *Treatment*. For each patient, the general information (modelled as attributes) such as *Gender* and *Severity* are preserved in the *Patient* class. The special purpose information for a fracture patient such as *cast_done* and *cast_removed* are stored in the subclass: *FracturePatient*.

In our example, there are four types of treatments available for fractures: *Repositioning*, *Cast*, *Surgery* and *Sling*. Applying a cast is probably the most common treatment for fractures. The *Cast* is removed after the fracture is healed. Dislocations are treated by *Repositioning*, and *Surgery* is required for complex injuries. Additionally, a *Sling* could also be used for a prescribed period of time.

For each treatment, it is modelled as a subclass of *Treatment*. Each different treatment has its own operation calls that represent the necessary operations that might be used for a fracture patient. For example, the operation call $apply(p : FracturePatient)$ in the *Cast* class denotes that a cast is applied to a specific patient.

[1] We choose this example because it is easy enough to be understood and also allows us to demonstrate the use of constraints in both OCL class invariants and operational contracts

[2] Note that this is not a general rule and it depends on different documents.

The use of a UML class diagram itself is not enough due to the fact that it is missing constraints that can capture the appropriate treatment process. Hence, we introduce a set of OCL constraints as shown enclosed in a box in Figure 2. These constraints specify the necessary restrictions when treating a patient with a fracture. The OCL constraints presented in the box in Figure 2 can be divided into two categories: class invariants (*inv*) and operational contracts (*pre*/*post*). The class invariants are used for expressing constraints that should hold all *stable* states. For example, the invariant for the class *Patient* states that every patient must be assigned with a unique identifier. Similarly, the invariant for the *FracturePatient* class suggests that every patient should be treated by at least one of the four treatments.

The OCL operational contracts are expressed as pre/post conditions for an operation call. A precondition specifies the conditions to be met before executing an operation call while a postcondition indicates what is to be achieved after executing an operation call. For example, the precondition for the operation call *apply* in the *Cast* class specifies that a patient must be x-rayed before applying a cast. The postcondition here implies that once the precondition is met, the cast procedure is complete. To ensure an appropriate precondition for each operation call, we use boolean attributes to store information such as whether a patient has been examined, checked, x-rayed and had a cast applied. For example, the precondition of *xray* in *Treatment* class requires the attribute *risk_checked* to be *true*. Hence, the set of OCL constraints defined in Figure 2 specifies the following constraints:

1. Every patient must have a unique id.
2. Every fracture patient must be treated using one of the four treatments.
3. Before prescribing any medicine, a patient must be examined.
4. Before taking an x-ray, a risk check must be performed.
5. A cast can only be removed after it is applied.
6. The doctor must perform an x-ray before applying a cast, performing repositioning or surgery.

These constraints listed above altogether essentially model the fracture treatment process for a patient.

2.2 Solving Constraints

To solve the constraints defined in Figure 2, we convert them into a set of logic sentences and then solve them using an SMT solver [8]. Each successful assignment found by the SMT solver is mapped to an instance of our model as shown in Figure 2. Our previous work on translating UML class diagrams along with OCL constraints allows us to automatically translate class invariants into SMT instances [20, 21]. We have also developed an approach that allows us to synthesise a call sequence from OCL operational contracts. This approach uses an SMT solver as a back-end engine for constraint solving.

To synthesise a call sequence with respect to each pre/postcondition and invariant, we model each operation call as a transition from one system state to another. For example, Figure 3 illustrates a transition of applying a cast to a fracture patient. Our logic sentences encode all possible transitions from the operation calls defined in Figure 2. We then let the SMT solver explore the search space for us in order to find the correct sequence. Our previous

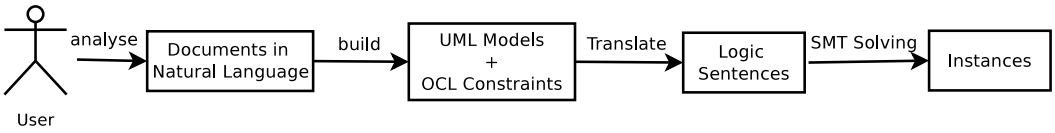

Figure 1: Our workflow for modelling software healthcare services.

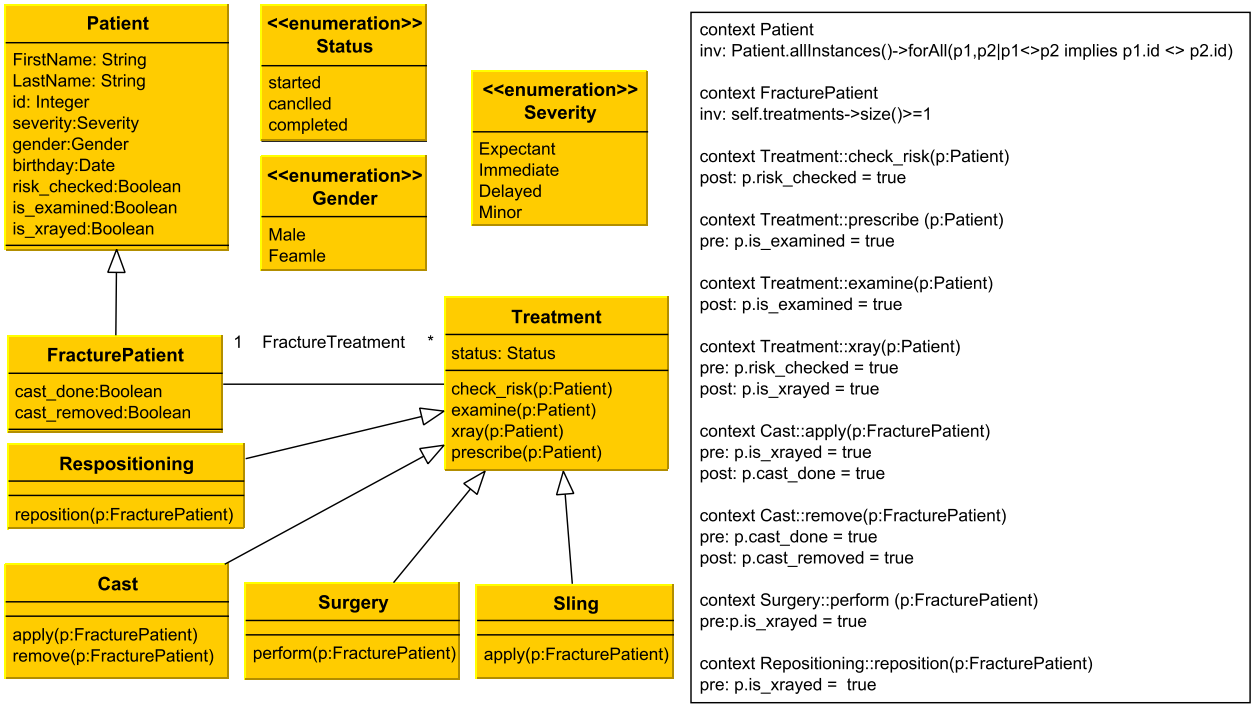

Figure 2: A UML class diagram for fracture treatment with class invariants and operational contracts specified in OCL.

experience of using SMT solvers has proved to us that those well-engineered tools are best suited to solve different kinds of constraints [21, 22].

The basic idea of our encoding is to use a state function to encode every object in each state after an operation call is invoked. This includes those properties specified to be changed in the post-condition and those that are not meant to be changed (frame conditions). For example, the state function for the operation call *xray* indicating the *is_xrayed* attribute is changed to be *true* after executing this call and leaves other attributes unchanged. When the SMT solver finds an assignment for our state function, we interpret it back into a valid call sequence. For example, the blue sequence in Figure 4 shows a valid call sequence (with respect to the constraints) found by the SMT solver and the red one shows an invalid call sequence when one ignores the precondition for the operation call *xray* [3].

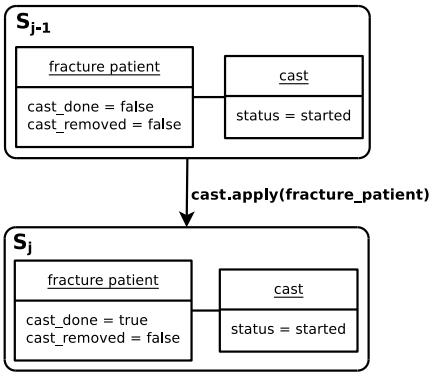

Figure 3: An example showing a transition from one system state to another via calling the *apply* operation defined in Figure 2.

[3]Here, we constrain that every call sequence must begin with an operation call *examine*, and each operation call applies to the same patient.

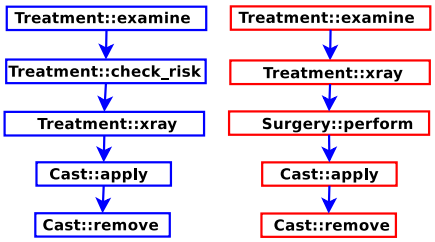

Figure 4: Two call sequences: the one in blue conforms to the constraints and the red one is invalid due to the missing x-ray risk check operation.

3 DISCUSSION

The results of the example (Figure 2) illustrate our initial effort in building a formal workflow model of patient treatment by introducing both OCL and SMT solving. Ultimately, our ambition is to generalize and extend our workflow to a larger framework and then apply it to more complex situations. We are particularly interested in Emergency Departments (ED) because they are constantly associated with problems such as overcrowding and staff scheduling [14]. EDs are extremely complicated, high-stress environments that require significant cross-departmental and cross-role coordination [3]. Hence, they impose many challenges on existing modelling techniques. Here, we outline some of those challenges and discuss how our workflow could possibly be further developed to tackle them.

Resource Scheduling. Resources in emergency departments should be allocated carefully. Planning an appropriate schedule for different resource will have impacts on care quality, budgets and staff morale [5, 18]. One direction to tackle this is by introducing a UML-based domain-specific language (DSL) that can model the dynamic aspect of the scheduling. One could then use this DSL to design different objective functions with respect to different scenarios. Then, SMT solving in our workflow here could be adjusted to solve the optimising of the schedule. Many SMT solvers have very dedicated algorithms for solving this type of optimisation [1].

Improving flow through Triage. The aim of the triage process is to improve medical staff preparedness and standardise the clinical response according to best practice [4, 9]. It is applied in Emergency Departments to incoming patients. Different practices for Triage exist. The workflow of Triage process currently relies on paper-based documents [10]. This makes interdependent constraints between different treatment procedures difficult to achieve. One possible way to improve this is we can explicitly express those interactive constraints in OCL and add an additional step in our workflow that allows its integration with guideline-based decision support and the electronic health record (EHR) systems. By implementing such a computer-supported early-assessment Triage procedure followed by a prompt initiation of treatment should lead to a reduction in the amount of time spent by patients in the emergency department [12].

4 CONCLUSION

In this paper, we propose a workflow for modelling for healthcare systems. This workflow exploits UML, OCL and SMT solving. Our workflow is exemplified with a fracture treatment pathway. The application of MDE to healthcare is an active topic currently [17]. The use of tools and techniques from formal verification is very attractive for building healthcare systems [15, 19].

We admit that the work presented here is an intial idea and requires more thorough evaluations. In the future, we plan to compare our modelling proposal against other formal approaches such as Alloy and Event-B by applying it to real-world healthcare scenarios. This inlcudes fully utlising different SMT solvers at both the specification and computational levels.

REFERENCES

[1] Nikolaj Bjørner and Anh-Dung Phan. 2014. νZ-Maximal Satisfaction with Z3. In *The 6th International Symposium on Symbolic Computation in Software Science*, Vol. 30. 1–9.
[2] R. Braun, H. Schlieter, M. Burwitz, and W. Esswein. 2016. BPMN4CP Revised – Extending BPMN for Multi-perspective Modeling of Clinical Pathways. In *49th Hawaii International Conference on System Sciences (HICSS)*. 3249–3258.
[3] S. Carrus, S. Corbett, and D Khandelwal. 2011. A hospital-wide strategy for fixing emergency-department overcrowding. https://www.mckinsey.com/. (2011).
[4] Michael Christ, Florian Grossmann, Daniela Winter, Roland Bingisser, and Elke Platz. 2010. Modern Triage in the Emergency Department. *Deutsches Ärzteblatt International* 107, 50 (2010).
[5] Stefan C. Christov, Heather M. Conboy, Nancy Famigletti, George S. Avrunin, Lori A. Clarke, and Leon J. Osterweil. 2016. Smart Checklists to Improve Healthcare Outcomes. In *SEHS (SEHS '16)*. ACM, 54–57.
[6] Carlo Combi, Giuseppe Pozzi, and Pierangelo Veltri. 2017. *Process Modeling and Management for Healthcare*. CRC Press.
[7] Anacleto Correia and Fernando Brito e Abreu. 2012. Adding Preciseness to BPMN Models. *Procedia Technology* 5 (2012), 407 417.
[8] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: an efficient SMT solver. In *14th TACAS*. Springer, Budapest, Hungary, 337–340.
[9] Agency for Healthcare Research and Quality. 2012. Emergency Severity Index (ESI): A Triage Tool for Emergency Department. (2012).
[10] Phil Gooch and Abdul Roudsari. 2011. Computerization of workflows, guidelines, and care pathways: A review of implementation challenges for process-oriented health information systems. *Journal of the American Medical Informatics Association* (2011).
[11] Javier Luis Cánovas Izquierdo, Jordi Cabot, Jesús J. López-Fernández, Jesús Sánchez Cuadrado, Esther Guerra, and Juan de Lara. 2013. Engaging End-Users in the Collaborative Development of Domain-Specific Modelling Languages. In *Cooperative Design, Visualization, and Engineering*. Springer, 101–110.
[12] Paul Richard Edwin Jarvis. 2016. Improving emergency department patient flow. *Clin Exp Emerg Med* (2016).
[13] Pilar Mata, Aladdin H. Baarah, Craig Kuziemsky, and Liam Peyton. 2014. An Application Meta-model for Community Care. *Procedia Computer Science* 37 (2014), 465 – 472.
[14] Megan McHugh, Kevin Van Dyke, Mark McClelland, and Dina Moss. 2014. *Improving Patient Flow and Reducing Emergency Department Crowding: A Guide for Hospitals*. Technical Report. Agency for Healthcare Research and Quality.
[15] Dominique Méry and Neeraj Kumar Singh. 2011. Medical Protocol Diagnosis Using Formal Methods. In *1st International Symposium on Foundations of Health Informatics Engineering and Systems*. 1–20.
[16] Shan Nan, Xudong Lu, Uzay Kaymak, Hendrikus Korsten, Richard Vdovjak, and Huilong Duan. 2017. A meta-model for computer executable dynamic clinical safety checklists. *BMC Medical Informatics and Decision Making* 17, 1 (2017).
[17] Rishi Kanth Saripalle. 2016. Need for a Specialized Metamodel for Biomedical and Health Informatics Domain. In *Smart Health*. Springer, 99–104.
[18] Seung Yeob Shin, Yuriy Brun, and Leon J. Osterweil. 2016. Specification and Analysis of Human-intensive System Resource-utilization Policies. In *SEHS (SEHS '16)*. ACM, 8–14.
[19] Xiaoliang Wang and Adrian Rutle. 2014. Model Checking Healthcare Workflows Using Alloy. *Procedia Computer Science* 37 (2014), 481 – 488.
[20] Hao Wu. 2016. An SMT-based Approach for Generating Coverage Oriented Metamodel Instances. *Internatinoal Journal of Information System Modeling and Design* 7, 3 (July 2016), 23–50.
[21] Hao Wu. 2017. Finding Achievable Features and Constraint Conflicts for Inconsistent Metamodels. In *13th European Conference on Modelling Foundations and Applications*. Springer, 179–196.
[22] Hao Wu. 2017. MaxUSE: A Tool for Finding Achievable Constraints and Conflicts for Inconsistent UML Class Diagrams. In *Integrated Formal Methods*. Springer, 348–356.