Applying a MDE Approach to a Healthcare Environment: a case study of an AE dept

Hussein Gannud, Hao Wu and Joseph Timoney
Computer Science Department,
National University of Ireland, Maynooth.
Email: {hganuud, haowu, jtimoney}@cs.nuim.ie

***Abstract*—One of the main challenges of Healthcare services is to find a suitable way of managing limited resources within a highly demanding environment. This challenge can now be tackled by deploying reliable software systems that are supported by Software Engineering practices. In this paper, we present a Model Driven Engineering (MDE) approach that, by way of an example, is applied to an Accident and Emergency (AE) department. This approach works by designing a UML class diagram annotated with a set of OCL constraints. These constraints formally express restrictions on interactions between various elements of the system. We formally evaluate our model by using a logic solver. This approach lays the foundation of our intentions to further develop a more complete health care system. We report the valuable lessons learnt from the work and explain the limitation we observed.**

I. Introduction

In many countries, and notably including Ireland, healthcare services face many challenging problems associated with increased demands on their services [3]. For example, coping with the random nature of patient arrivals, or the organisation of rosters for a large number of staff members with different specialities. Information technologies are needed to tackle these challenges through carefully designed software artefacts that facilitate the automation of many processes, reducing the burdens on resources. Healthcare services have already been profoundly influenced by computing, transforming its functionalities from an old-style paper-based organisation into one with software-managed systems. Both commercial bespoke and publically available solutions exist, and notably recent open source projects for Healthcare information systems include Model Driven Health Tools (MDHT) [28], GNUHealth [12], and LibreHealth [20]. However, there still exists gaps between the theoretical potential of healthcare software artefacts and what they currently are achieving.

Away from Healthcare, it is recognised that many private sector industries have seen significant benefits following the introduction of customised high quality software solutions [30]. One Software Engineering approach that has stood out in particular for such systems is known as Model-Driven Engineering (MDE). It allows the exploitation of models to simulate, estimate, understand, communicate, and produce code [11][30]. The Object Management Group promote Model-Driven Architecture (MDA) [1], that is derived from MDE, and offer many examples of its successful application that includes projects by the Rail division of Siemens [16], Deutsche Bank [15], and The Swedish Parliament [14]. Such testimonials indicate that the utility of MDE should lead to superior solutions to the challenges for Healthcare systems.

Better insights into tackling these challenges can be obtained by separating them into two distinct categories as facilitated by MDE, that is, Static and Dynamic [8]. Static features of challenges are mainly infrastructure-oriented and are not time-dependent. They influence the designing/adapting of systems that facilitate effective management and/or access to resources. In the context of healthcare these resources could be staff members' time sheets or patients' history records. Dynamic features are concerned with issues that occur within time spans that necessitate the allocation of limited resources efficiently. For example, assigning specialist treatment to patients in a manner such that the waiting time is kept to a minimum [2]. An obvious problem area that should benefit from the application of this MDE approach is the Accident and Emergency (AE) department of a hospital. According to the media coverage it appears that many AE departments are not functioning correctly in either static and dynamic dimensions. By implication then, the supporting software systems are not available to rectify this.

Therefore, in this paper as a first step we present a Model-Driven Engineering (MDE) approach that models a static aspect of an AE department. This is adopted as it well accepted that it is sensible to tackle the static problems first. The focus within the paper is on modelling the relationships between patients and an AE department, and specifying the necessary rules as the constraints over these relationships. Additionally, to guarantee quality, the correctness of designed models is formally *verified*. The use of MDE approach gives us several advantages: 1) It is a widely accepted approach to model software artefacts. 2) It provides a variety of structural and behavioural components so that we are able to *precisely* build our models and express constraints. 3) It enables us to employ verification techniques to *formally* verify the correctness of our models.

The main contribution of the paper are summarised as follows:

1. We design a model for an AE department by using UML class diagrams and establish a set of constraints that are expressed using Object Constraint Language (OCL) (Section III).
2. We formally evaluate our model by casting it to a

Satisfiability Modulo Theories problem (Section IV-B).

II. Related Work

Surveying the literature on healthcare and MDE it can be found that there has been an upswing of interest towards applying MDE to healthcare systems [29]. These models have been directed at improving various aspects of healthcare systems such as the data layers for storing patients medicine records and the control layers for monitoring patient flows [6]. Currently, there appears to be natural division within the literature in the sense that the work has been to either develop *static* or *dynamic* models but not both together.

With regard to static models, Lahboube et al. propose a hospital information system metamodel based on MDE using UML class diagrams. They realise that the difficulties in implementing complex Health Information Systems (HIS) are many but by having a formal framework it would contribute greatly to overcoming operational issues. The validation of their metamodel is done by deployment in a real hospital environment [19]. Similarly, Raghupathi and Umar use MDA with UML to design a prototype patient tracking information system. Advantages including abstraction, productivity gains, stable interoperability, portability, and cost efficiency [26]. Raistrick use MDA and executable UML for a new access control interface to patient data [21][22]. This is done to simplify the integration of old and new Software systems. It facilitated the use of code generators, derived from the executable models, for porting specific code implementations across multiple platforms [27]. Traore et al. present a methodology that combined MDA with a Service Oriented Architecture (SOA) to enable a telemedicine service to have interoperable exchanges across applications [29]. Another work mention that how model-driven application development fosters semantic interoperability and interconnected innovative application[10].

Research that investigates dynamic modeling and MDE include [33], [6] and [18]. [6] study the integration of a qualitative and quantitative approach to construct a patient flow model. Another work investigates different models with forecasting methods to analyse daily patient visits to an emergency department [6]. Lastly, Jiang et al. propose a dynamic decision support system based on MDA with UML [18].

Clearly, the static and dynamic modeling methodologies of MDE have different roles to play in the healthcare environment and are useful to a variety of applications. Interoperability and code generation currently appear to be the most desirable features of static MDE modeling. However, a key facility of static models, OCL, has not been introduced in the literature to date. OCL can be understood as a means to create very sophisticated interacting models which can be verified using a powerful solver to determine the existence of constraint conflicts. This does suggests that there is an opportunity to explore MDE with regard to this. Therefore, our paper distinguishes itself from prior work by demonstrating how to develop a MDE solution with OCL constrains in a static model. In particular, we demonstrate the feasibility of this approach by applying it to an AE system. We consider this approach as our first step towards building a more complete MDE model that integrates both static and dynamic models.

This paper is organised as follows: Section III gives details of our model by presenting a UML class diagram annotated with OCL constraints. Section IV-B shows the formal evaluation of our model by answering two research questions. Section V discusses the valuable lessons learnt from the approach and identifies one particular limitation. Finally, Section VI concludes the paper and gives the direction for our future work.

III. The Approach

A. Overview

Our approach to modelling an AE department can be viewed as three steps. First, we model the relationships between the patients and the AE department by using a UML class diagram. Second, we then define a set of constraints in Object Constraint Language (OCL) over these classes to rule out certain scenarios. For example, a staff member can never leave when a patient requires treatment urgently. Finally, we cast our models along with these constraints to a Satisfiability Modulo Theories problem so that we are able to formally verify our design.

B. Modelling AE Department

We use UML class diagrams to model multiple entities for an AE department including staff members, patients and relationships between them. The full UML class diagram is shown in Figure 1.

Definition 1. A UML class diagram can be defined as a tuple $\langle \mathcal{C}, \mathcal{A}, \mathcal{T} \rangle$, where

1. $\mathcal{C}$ is a set of classes.
2. $\mathcal{A}$ is a set of associations which captures the relationships among different classes.
3. $\mathcal{T}$ is an inheritance tree that describes inheritance relationships over $\mathcal{C}$.

By using the Definition 1, the classes in our model for the AE department in Figure 1 can be viewed as follows:

$$\begin{aligned} \mathcal{C} = \ & \{Hospital, Department, People, Patient, Staff, \\ & Triage\_Staff, Doctor, Nurse, Technicians, \\ & Porter, Receptionist\} \end{aligned}$$

For each class $c \in \mathcal{C}$, it consists of a set of attributes S where each one of them has its own type t. For example, in our model (Figure 1), the class $Department$ has 4 string type attributes: *name*,*id*,*loc* and *phone*. Similarly, the class $AE\_Department$ has 2 integer type attributes: $size$ and $capacity$ describing the number of staff members working there and the maximum number of patients it can admit respectively.

The types used for defining an attribute in a class could vary from primitive types to collection types such as Set and Bag. One of the types commonly used for defining an attribute is

$Enumeration$. With the use of this kind of type, an attribute can describe a set of possible values. For example, we define a patient's status (in the $Patient$ class) by using the enumeration type $Patient\_Status$. This type defines 4 possible values for a patient's status: $Expectant$, $Immediate$, $Delayed$ and $Minor$.

Our model in Figure 1 also depicts a set of associations as they are defined over the classes, and thus they are captured by the following set.

$\mathcal{A} = \{ConsistOf, Admit, WorkFor, Check\}$ where each association in the set $\mathcal{A}$ describes a *relationship* between two classes:

$$ConsistOf = (Hospital, Departments)$$
$$Admit = (Patient, AE\_Department)$$
$$WorkFor = (Staff, AE\_Department)$$
$$Check = (Patient, Triage\_Staff)$$

Each association also defines multiplicities at each association end. For example, the association $Admit$ describes a relation that an $AE\_Department$ can take in zero or more $Patient$s. Similarly, the association $WorkFor$ between classes $AE\_Department$ and $Staff$ describes that there are multiple staff members who work in the AE department. The association $Check$ captures that multiple patients could be examined by different triage staff members. In some scenarios, a class might consist of several other classes which are part of its components. In this context, a *composition* (Not-Shared Association) is used. For example, we use a composition to indicate that a $Hospital$ consists of many $Department$s.

Besides the classes and associations defined for our model, another important feature is inheritance. By appropriately defining inheritances among multiple classes, we are able to raise the abstraction level of our model. In our model, all classes that describe a person are inherited from a class called $People$. This is an *abstract* class that defines common attributes such as first name ($fname$), last name ($lname$), birthday ($birthday$), and gender ($gender$). Thus, we abstract these attributes so that classes such as $Patient$ and $Staff$ can share these common attributes. We apply the same abstraction to the class $AE\_Department$.

C. Defining Constraints

We have our class diagrams defined for $AE\_Department$ in Figure 1. We now can define constraints over our model by using Object Constraint Language (OCL). Object Constraint Language (OCL) is a part of the UML standard and is widely used by modellers to write logical constraints for different types of UML models including class diagrams and state diagrams [24], [23]. In general, OCL is different from general-purpose programming languages such as Java and C++. It is a declarative language and its constructs are similar to First-order Logic (FOL). OCL supports a variety of features including: collection types (sets, bags, lists), quantifiers, navigation, etc. The use of OCL in our model has several advantages: 1) We are able to express constraints in a precise manner since OCL is based on FOL. 2) We can adapt formal verification techniques to verify the correctness of the model annotated with OCL constraints.

For the class diagram in Figure 1, we have defined 10 constraints through class invariants over 4 different classes: $AE\_Department$, $Staff$, $Triage\_Staff$ and $Patient$. These invariants are shown in Figure 2. A class invariant is essentially a boolean expression such that it always evaluates to true. The keyword $context$ describes a particular class where the invariants are defined under. For example, for $AE\_Department$ class it has 3 class invariants.

We now describe our 10 class invariants defined in terms of their contexts.

- $AE\_Department$: this class defines 3 invariants. The first invariant imposes a constraint that an $AE\_Department$ must have some staff members ($size$ attribute) and its maximum number of patients ($capacity$ attribute) that it can take in. $self$ is an OCL keyword here. This keyword specifies the context of a constraint. In this example, the context here is class $AE\_Department$. The second invariant indicates that there must be some staff members working in the $AE\_Department$. The $staff$ used in this invariant is a navigation from the class $AE\_Department$ to $Staff$ through the association $WorkFor$. This captures the set of all staff members who work in the $AE\_Department$. The operation $notEmpty()$ indicates that the set cannot be empty. Similarly, the third invariant imposes a constraint that among all the staff members working in the AE department there must be someone who is a doctor. The $exists$ here is a quantifier that quantifies over the set of staff members working in the department. To determine whether a staff member is a doctor or not, we use the OCL built-in predicate $oclIsTypeOf()$ to ensure the staff type.
- $Staff$: We define two invariants for this class. The first invariant states that for any two different staff members in the hospital they must have different unique id numbers. In other words, this invariant imposes a constraint that every staff member must have a unique id. The operation $allInstances()$ here returns a set of objects whose type is $Staff$. Since the returned objects here is a set (collection), we use $forAll$ to quantify every object in this set. Similarly, the second invariant states that among all the staff members working in the hospital, some of them have to be triage staff.
- $Triage\_Staff$: We define three invariants here and each one specifies a particular staff type. For example, the second invariant indicates that there must be some nurses. Therefore, in an AE department we must have some doctors, nurses and technicians as triage staff. Note that the $\#$ symbol here denotes a particular value of an enumeration type.
- $Patient$: We have two invariants defined here for patients. The first invariant captures a scenario that if a patient's status is $Expectant$ or $Immediate$, then this patient must be assigned to a doctor (or doctors). Here, we use an attribute $assigned$ to indicate whether

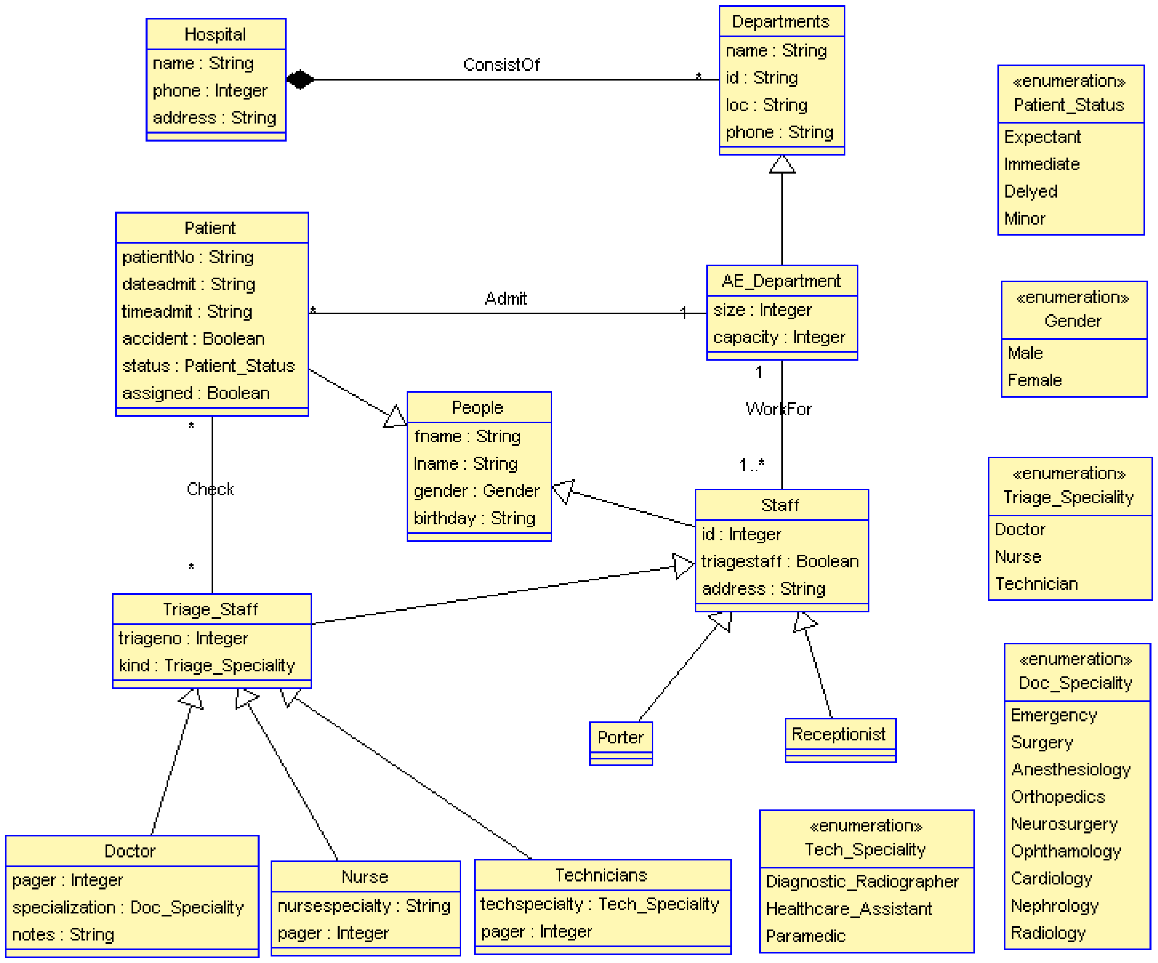

Fig. 1. The UML class diagram for AE Department.

a patient has been assigned to a doctor or not. The second invariant implies that for every patient admitted to the $AE\_Department$, the patient must be checked by some staff members in the department. Note that the $staff$ returns a set of objects (staff members) through a navigation from the class $Patient$ to the $Triage\_Staff$.

IV. Implementation & Evaluation

A. Implementation

We have implemented the class diagram shown in Figure 1 and OCL constraints in Figure 2 into USE. USE is a UML modelling environment that employs its own textual specification language to define the structural building blocks of model-like classes and associations. It supports the OCL expressions that can be used to define constraints and operations without any side effects [13]. We choose USE because it fully supports OCL constructs and we are able to freely define, evaluate, and validate different types of constructs. The full USE specification for our model is available at: http://www.cs.nuim.ie/~haowu/issc/AE_Department.use.

B. Evaluation

To formally evaluate our model, we focus on answering the following two research questions.

- **RQ1. How can we formally prove correctness of our model ?**
- **RQ2. How efficiently can we produce valid instances of our model ?**

RQ1. To answer the first research question, we need to show that there are no constraint conflicts in our model along with 10 class invariants. In other words, we show that our model is consistent. In order to do that, we map our model along with 10 class invariants into First-order Logic (FOL) formulas and prove these formulas by testing the *satisfiability* via a Satisfiability Modulo Theories (SMT) solver [9]. We say a formula $\mathcal{F}$ is satisfiable iff there exists one assignment in its truth table that can make $\mathcal{F}$ evaluate to true. Determing a propositional formula is satisfiable or not is NP-complete [1], and deciding satisfiability of a FOL is undecidable in general

[1]This is known as the boolean satisfiability (SAT) problem

```
context AE_Department
inv1: self.size>0 and self.capacity>0
inv2: self.staff->notEmpty()
inv3: self.staff->exists(s|oclIsTypeOf(Doctor))

context Staff
inv4: Staff.allInstances()->forAll(s1,s2|s1<>s2 implies s1.id <> s2.id)
inv5: Staff.allInstances()->exists(s|s.triagestaff=true)

context Triage_Staff
inv6: Triage_Staff.allInstances()->exists(s|s.kind=#Doctor)
inv7: Triage_Staff.allInstances()->exists(s|s.kind=#Nurse)
inv8: Triage_Staff.allInstances()->exists(s|s.kind=#Technician)

context Patient
inv9: Patient.allInstances()->forAll(p|p.status=#Expectant or
         p.status=#Immediate implies p.assigned=true)
inv10: Patient.allInstances()->forAll(p|p.staff->notEmpty())
```

Fig. 2. The 10 constraints defined as class invariants for the model in Figure 1.

[2][7], [5].

An SMT solver consists of multiple decision procedures that take in a number of FOL formulas and determine whether formulas are *satisfiable* or not in an efficient manner. Many challenging problems and proofs in software engineering can be cast to the problem of testing the satisfiability of FOL formulas such as program synthesis, type checking, model checking, and test case generation [17][25][4][31].

For our model, we also use this idea. Our proof of the correctness can be divided as three steps: we first define the correctness of our model, then map our model into a set of FOL formulas, and lastly we use an SMT solver to prove that our formulas are correct.

We first define the correctness of our model as follows:

Definition 2. A model is consistent/correct iff its corresponding FOL formulas are satisfiable.

By the above definition, we map our model along with 10 invariants into FOL formulas. The mapping of each class in the model in Figure 1 is very straightforward. We simply use a predicate to denote each class and association. For an enumeration type such as $Patient\_Status$, we use a function to denote its possible ranges. Since each class invariant imposes a constraint over the model, we map it individually to a corresponding FOL formula. For simplicity, we only show the mapping of 3 invariants here: $inv1$, $inv5$ and $inv10$. The full mappings can be found at our website: http://www.cs.nuim.ie/~haowu/issc/proof.smt2.

[2]Note that some of FOL theories are decidable.

For $inv1$, we map it to the following formula:

$$\begin{aligned}&\forall o : Int.\ TypeDept(o)\\ &\Rightarrow \big(DeptSize(o) > 0 \land DeptCap(o) > 0\big)\end{aligned} \tag{1}$$

$TypeDept$ is a type predicate that determines whether an object o is of $AE\_Department$ type. The attributes $size$ and $capacity$ are mapped to two functions $DeptSize : Int \to Int$ and $DeptCap : Int \to Int$ since they are both integer types.

Similarly, we map $inv5$ to Formula 2, where $StaffTri : Int \to Bool$ is a function capturing the boolean attribute $triagestaff$ defined in $Staff$ class in Figure 1.

$$\exists o : Int.\ TypeStaff(o) \Rightarrow StaffTri(o) = true \tag{2}$$

Invariant $inv10$ imposes a constraint on an association ($Check$) between the $Patient$ and $Triage\_Staff$ classes. To capture this constraint over an association, we design a binary predicate $RelCheck : Int \times Int \to Bool$ to denote whether two objects are in an association. Therefore, Formula 3 expresses that every patient must be checked by some triage staff.

$$\begin{aligned}&\forall o_1 : Int\ \exists o_2 : Int\ \big(TypePatient(o_1)\land\\ &TypeTriStaff(o_2)\big) \Rightarrow RelCheck(o_1, o_2)\end{aligned} \tag{3}$$

Note that here we consider that binary association $Check$ as an asymmetric relation between a triage staff and a patient. For example, a triage staff can check a patient but not the other way around.

Now let F_i be an FOL formula representing the ith class invariant in a model. To prove a model is consistent, we conjoin each F_i and ask an SMT solver to check whether Formula 4 is *satisfiable*. The SMT solver used for our model is

Z3 [9], and the result from the solver is 'sat' representing that our formulas are satisfiable. Thus, now we can formally show that our model in Figure 2 along with the 10 class invariants in Figure 1 are consistent.

$$\bigwedge F_i \tag{4}$$

RQ2. To answer the second research question, we adapt an existing instance generation technique that is prototyped into an automatic tool called: ASMIG [32]. ASMIG reads in a UML class diagram annotated with OCL constraints, translates these into a set of SMT2 formulas, and calls a state-of-the art SMT solver to produce instances that provably conform to the constraints defined over a UML class diagram [9].

We configure ASMIG with a different number instances for each class to be generated and conduct 5 runs. All runs were performed on an Intel(R) i3 machine with 4 GB memory. The results of each run are recorded in Table I. In order to choose an appropriate number of instances to be generated for each class, we use ASMIG's internal module to automatically calculate a suitable number of instances for each class. We then ask ASMIG to enumerate different valid instances, and gradually increase this number for each run to measure the efficiency of generating instances for our model. ASMIG successfully finds 100 valid instances within 1 second. Each instance found by ASMIG is guaranteed to conform to the constraints defined in Figure 2. Figure 3 shows one example of generated instances.

Number of Runs	Time (in ms)	Number of Instances
run 1	468	10
run 2	500	50
run 3	717	70
run 4	907	100
run 5	1023	120

TABLE I
THE RESULTS OF 5 RUNS. THE COLUMN 'NUMBER OF INSTANCES' DENOTES THE NUMBER OF INSTANCES GENERATED BY ASMIG.

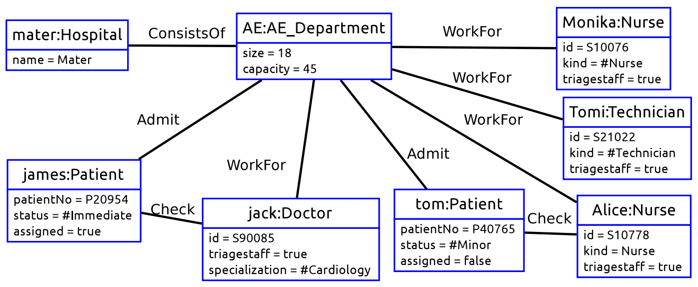

Fig. 3. One instance that provably conforms to our model in Figure 1 and the OCL constraints in Figure 2. Note that this instance is processed to remove some attributes for simplicity.

V. Lessons Learnt & Limitation

In this section, we report our lessons learnt from using the Model Driven Engineering (MDE) approach to model an AE department, and identify any limitations that have emerged.

Lessons Learnt. We have learned three important lessons from our experience of using MDE. 1) We found that it is easy to use class diagrams to model the relationships between patients and an AE department. This is mainly because UML class diagrams are naturally designed for capturing the relationships between different entities. 2) By writing constraints in Object Constraint Language (OCL), we are able to express rules which cannot be addressed by just using UML class diagrams. Therefore, using UML class diagrams along with OCL constraints is a much more expressive approach to capturing complicated rules such as the priority given to a patient based on his/her status ($inv9$ in Figure 2). 3) The verification technique used for verifying our models establishes our confidence in the actual implementation. We are guaranteed that our model is consistent and that there are no conflicts between the constraints.

Limitation. The limitation of our approach is that the cost of designing OCL constraints can be high. This is because writing a precise OCL constraint for a rule described in natural language is sometimes not straightforward. However, this can be avoided in a real-life implementation scenario by pairing a modelling expert with a domain specific expert. In this way, the domain specific expert is responsible for helping the modelling expert to digest the rules correctly. This may also require that the modelling expert iteratively uses instance generation techniques to generate valid instances and the domain expert validates that each one of them does indeed meets the rules.

VI. Conclusion

In this paper, we have shown an MDE-based approach to model the static aspects of an AE department through a UML class diagram annotated with OCL constraints, and verified our model by using SMT-based verification techniques. The evaluation results suggest that our approach is very promising. This approach is unique in two ways: 1) By integrating OCL into our model we are able to specify rules that are not expressible in UML class diagrams. 2) By logically verifying our model via SMT solving, we are confident about our design before implementing it into an actual software artefact. More importantly, this work lays the foundation of our first step towards a complete framework for modelling a health care system.

In the future, we intend to incorporate both static and dynamic models to build a framework based on more realistic assumptions such as patient flow and health information exchange. This involves specifying OCL constraints over dynamic models with respect to the information described in the static models. A more demanding problem is then to ensure consistency between the two kinds of models. To tackle this, one of the possible directions to explore is that of designing a set of new mappings from both kinds of models to a logic formalism that can be formally verified using an SMT solver.

References

[1] Object Management Group. http://www.omg.org.

[2] Norazura Ahmad, Noraida Abdul Ghani, Anton Abdulbasah Kamil, and Razman Mat Tahar. Modeling emergency department using a hybrid simulation approach. In *IAENG Transactions on Engineering Technologies*, pages 701–711. Springer, 2013.

[3] Ines Ajmi, Hayfa Zgaya, Lotfi Gammoudi, Slim Hammadi, Alain Martinot, Rgis Beuscart, and Jean-Marie Renard. Mapping patient path in the pediatric emergency department: A workflow model driven approach. *Journal of Biomedical Informatics*, 54:315–328, 2015.

[4] Alessandro Armando, Jacopo Mantovani, and Lorenzo Platania. Bounded model checking of software using SMT solvers instead of SAT solvers. *International Jounrnal on Software Tools for Technology Transfer*, 11(1):69–83, Jan 2009.

[5] Mordechai Ben-Ari. First-order logic: Undecidability and model theory *. In *Mathematical Logic for Computer Science*, pages 223–230. Springer, 2012.

[6] M. Chong, M. Wang, X. Lai, B. Zee, F. Hong, E. Yeoh, E. Wong, C. Yam, P. Chau, K. Tsoi, and C. Graham. Patient flow evaluation with system dynamic model in an emergency department: Data analytics on daily hospital records. In *2015 IEEE International Congress on Big Data*, pages 320–323, June 2015.

[7] Stephen A. Cook. The complexity of theorem-proving procedures. In *3rd Annual ACM Symposium on Theory of Computing*, pages 151–158. ACM, 1971.

[8] Alberto Rodrigues da Silva. Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures*, 43:139–155, 2015.

[9] Leonardo De Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, Budapest, Hungary, 2008. Springer.

[10] Hans Demski, Sebastian Garde, and Claudia Hildebrand. Open data models for smart health interconnected applications: the example of openehr. *BMC Medical Informatics and Decision Making*, 16(1):137, 2016.

[11] T. Gherbi, D. Meslati, and I. Borne. Mde between promises and challenges. In *2009 11th International Conference on Computer Modelling and Simulation*, pages 152–155, March 2009.

[12] GNUHealth. http://http://health.gnu.org/support.html.

[13] Martin Gogolla, Fabian Büttner, and Mark Richters. USE: A UML-based specification environment for validating UML and OCL. *Science of Computer Programming*, 69(1-3):27–34, 2007.

[14] Object Management Group. Borland technology builds advanced parliamentary work-flow system, May 2016.

[15] Object Management Group. Deutsche bank bauspar ag uses arcstyler to embed existing cobol mainframe application into modern web-based systems, May 2016.

[16] Object Management Group. Siemens railcom and model driven architecture, May 2016.

[17] Sumit Gulwani, Susmit Jha, Ashish Tiwari, and Ramarathnam Venkatesan. Synthesis of loop-free programs. In *The 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 62–73. ACM, 2011.

[18] L. Jiang, B. Xu, C. Xie, and H. Cai. A framework of emergency clinical decision support system based on mda and resource model. In *Proceedings of the 2014 IEEE 18th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 451–456, May 2014.

[19] Farid Lahboube and Ounsa Roudies Nissrine Souissi. Building a his supervision metamodel. In *11th System of Systems Engineering*, pages 1–6. IEEE, 2016.

[20] LibreHealth. http://librehealth.io/.

[21] HongXing Liu, YanSheng Lu, and Qing Yang. Xml conceptual modeling with xuml. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 973–976, New York, NY, USA, 2006. ACM.

[22] Stephen J. Mellor and Marc Balcer. *Executable UML: A Foundation for Model-Driven Architectures*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[23] Object Management Group. Unified Modeling Language, Infrastructure Version 2.4.1, August 2011.

[24] Object Management Group. Object Constraint Language Version 2.3.1, Jan 2012.

[25] Zvonimir Pavlinovic, Tim King, and Thomas Wies. Finding minimum type error sources. In *The 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, pages 525–542. ACM, 2014.

[26] Wullianallur Raghupathi and Amjad Umar. Exploring a model-driven architecture (mda) approach to health care information systems development. *International Journal of Medical Informatics*, 77(5):305–314, 2008.

[27] Chris Raistrick. Uml modeling languages and applications. chapter Applying MDA and UML in the Development of a Healthcare System, pages 203–218. Springer-Verlag, Berlin, Heidelberg, 2005.

[28] MD Health Tools. https://projects.eclipse.org/projects/modeling.mdht.

[29] Boukaye Boubacar Traore, Bernard Kamsu-Foguem, and Fana Tangara. Integrating MDA and SOA for improving telemedicine services. *Telematics and Informatics*, 33(3):733 – 741, 2016.

[30] J. Whittle, J. Hutchinson, and M. Rouncefield. The state of practice in model-driven engineering. *IEEE Software*, 31(3):79–85, May 2014.

[31] Hao Wu. Generating metamodel instances satisfying coverage criteria via SMT solving. In *The 4th International Conference on Model-Driven Engineering and Software Development*, pages 40–51, 2016.

[32] Hao Wu, Rosemary Monahan, and James F. Power. Exploiting attributed type graphs to generate metamodel instances using an SMT solver. In *7th International Symposium on Theoretical Aspects of Software Engineering*, Birmingham, UK, 2013.

[33] M. Xu, T. C. Wong, K. S. Chin, S. Y. Wong, and K. L. Tsui. Modeling patient visits to accident and emergency department in hong kong. In *2011 IEEE International Conference on Industrial Engineering and Engineering Management*, pages 1730–1734, Dec 2011.