Fundamenta Informaticae 168 (2019) 1–13
DOI 10.3233/FI-2019-1800
IOS Press

A Computational Model for The Access to Medical Service in a Basic Prototype of a Healthcare System

Luigia Petre
Distributed Systems Laboratory
Åbo Akademi University and Turku Centre for Computer Science, Finland

Usman Sanwal, Gohar Shah, Charmi Panchal, Dwitiya Tyagi
Computational Biomodeling Laboratory
Åbo Akademi University and Turku Centre for Computer Science, Finland

Ion Petre*
Computational Biomodeling Laboratory
University of Turku and Turku Centre for Computer Science, Finland
National Institute for Research and Development in Biological Sciences, Romania
ion.petre@utu.fi

Abstract. How robust is a healthcare system? How does a patient navigate the system and what is the cost (e.g., number of medical services required or number of times the medical provider had to be changed to get access to the required medical services) incurred from the first symptoms to getting cured? How will it fare in the wake to a sudden epidemic or a disaster? How are all of these affected by administrative decisions such as allocating/diminishing resources in various areas or centralising services? These are the questions motivating our study on a formal prototype model for a healthcare system. We propose that a healthcare system can be understood as a distributed system with independent nodes (healthcare providers) computing according to their own resources and constraints, with tasks (patient needs) being allocated between the nodes. The questions about the healthcare system become in this context questions about resource availability and distribution between the nodes. We construct in this paper an Event-B model capturing the basic functionality of a simplified healthcare system: patients with different types of medical needs being allocated to suitable medical providers, and navigating between different providers for their turn for multi-step treatments.

*Address for correspondence: University of Turku and Turku Centre for Computer Science, Finland

1. Introduction

Healthcare systems are highly complex environments involving many different stakeholders (e.g., clinicians, patients, administrators) with highly diverse objectives (e.g., driven by medical concerns, need of effectiveness, need of efficiency, focus on costs or on service availability). Changes in a healthcare system are almost always driven by economical or administrative constraints and it is highly difficult to predict their consequences on the overall patient-focused quality of the system. We are interested in this paper in describing some of this complexity by building a formal model capturing the basic architecture of a healthcare system: medical providers with specific capabilities, the connections between them, and the way patients navigate the system to have their medical needs addressed. The model captures the connections between the users and their first point of contact (primary providers), and the connections between the providers, from the small local units to the highly-specialized units, with the aim of capturing the path of a user through the system from general nurse advise to sophisticated treatments by specialist teams. The user is assumed to have registered at the nearest primary provider. The primary provider addresses all the needs within its expertise. If more medical services are needed, the primary provider assigns the patient to a secondary provider where the required service could be obtained. The same scenario repeats here, with the patient being potentially sent to other providers as well. Throughout the process the medical situation can be re-evaluated and the medical services needed for that patient updated. An individual can have multiple needs and a provider can be connected to many other providers. The architecture of our basic prototype healthcare system is illustrated in Figure 1.

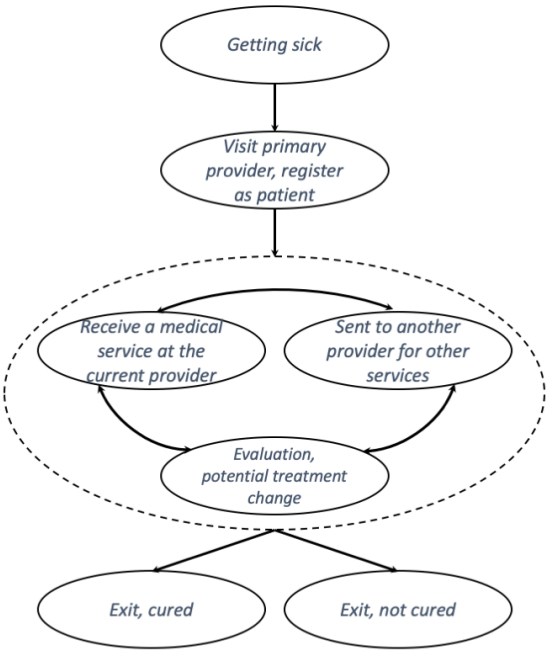

Figure 1. Overview of the architecture of our basic prototype of a healthcare system model

The model we build in this paper uses the state-based Event-B formalism [1]. Event-B is an extension of the B-method [2], with elements of Action Systems [3], TLA [4], and UNITY [5], introduced for modelling and reasoning about systems and software.

The paper is structured as follows. We overview in Section 2 some basics about Event-B. We present the construction of our model in Section 3. We conclude with a brief discussion in Section 4. All Event-B models discussed in the paper are available (both in Event-B format and in pdf format) at `http://users.abo.fi/ipetre/health-eventb-model.zip`.

2. Preliminaries

Event-B is a formal method focused on the stepwise development of models, building on earlier formalisms such as the B-Method [2] and the Action Systems [3]. An Event-B model is usually developed at different levels of abstraction, but the model in each level contains the initial value of the system state as well as the allowed changes to the system state. Thus, all the allowed *traces* of the system state are specified at each abstraction level and semantics-wise, an Event-B model is a state transition system viewed at different abstraction levels. For an introduction to Event-B we refer to [1]. We discuss here briefly a few basic concepts needed in the paper.

There are two types of modules in Event-B, called *contexts* and *machines*. A context contains definitions of constants, carrier sets, as well as axioms about them. A machine contains variables, invariants and events, and can *see* one or more contexts. We describe the system state via *variables*; the types of variables and other interesting properties that the state of the system must respect are specified as *invariants*. The system state changes as described by *events*; the initial system state is specified by a special event called *Initialisation*. An event can have a *guard*, which is a predicate on the machine variables and, possibly, on the local variables of the event. When the guard holds, we say the event is *enabled*. This means that it can be selected for execution. If two or more events are enabled at the same time, then only one is non-deterministically chosen and executed. Besides a guard, an event also has *actions*, which are assignments to various variables. If the event is chosen and executed, then its corresponding actions are executed and consequently the variables assigned to in the actions are updating their values; the rest of the state variables are kept unchanged. When no more events are enabled, we say the machine has deadlocked.

Event-B is centred around the notion of *refinement*, see [1]. A model (typically consisting of a machine and potentially a context) can be developed at different levels of abstraction. In the initial (and simplest) abstraction level, a model can be very basic and only encompass the elementary features of the system to model. Properties that we want the system to respect can be formulated already at this level, e.g., as invariants, and *proved* to hold. New features can be gradually added to a model, so that the traces of the more abstract model's state are still valid in the more concrete model's state and the new features still respect the properties specified for the more abstract model. In this case, we say that the more concrete model *refines* the more abstract model. The new features can, for instance, be modeled with new variables, and new events can be specified to model the allowed changes of these new variables. The old events can also be modified to include changes of new variables, but so that the traces of the old variables in the more concrete model are their traces in the more abstract model. This

type of refinement is called *superposition refinement*. Other types of refinement also exist, for instance *data refinement*, where some variables in the more abstract machine are replaced by other variables in the refined machine; in this case, a so-called *gluing invariant* is specified, that describes the relation between the old and the new variables. All the occurrences of the old variables are replaced in events in the refined machine by the new variables; the only place where the old variables still appear is in the gluing invariant. The refinement relation, denoted $\sqsubseteq$, is asymmetric and transitive. Examples of refinement-based modeling in Event-B are numerous. For instance, refinement in Event-B was used to model protocols for file transfer and bounded retransmission, to model control systems, concurrency, electronic circuits, network synchronisation, leader election algorithms, etc [1]. Refinement models have also been developed for smart cash card systems [6], vehicle platoons [7], topology discovery in graphs [8], self-recovery in sensor-actor networks [9], spacecraft systems [10], coordination in peer-to-peer networks [11], smart grid recoverability [12], proactive routing in wireless networks [13], etc.

Event-B is constructed on top of set theory and logic, and every model has to verify various rules, called *proof obligations*. For instance, when we specify some properties as invariants, we have to prove that the initialisation of the variables establishes the invariants and that the invariants hold before and after the execution of any event. More proof obligations need to be discharged when we refine models.

Event-B is supported by an Eclipse-based software package called the Rodin platform [14]. The Rodin platform allows one to edit the models and it generates all the required proof obligations. Importantly, the Rodin platform uses several prover engines such as those from AtelierB, namely PP (Predicate Prover) and ML (the Mono Lemma prover), in addition to the implicit NewPP prover, pre-installed in Rodin, see [14]. Based on these provers, Rodin discharges automatically all the proof obligations that it can, using a mix of various deduction rules, proof rewriting techniques, resolution methods, arithmetic, etc. [15]. The proof obligations not discharged can be discharged interactively by the modeler. Often, the fact that a proof cannot be automatically discharged signals a problematic modeling aspect and the modeler is prompted to reconsider the model first. This interleaving of modeling and proving is a key feature of working with the Rodin platform and is quite similar to the compilation of programs [14].

3. Building the Event-B model

We developed the model in two steps, connected through model refinement. In the first step we introduced all the basic ingredients: individuals, diseases, providers, medical services, patients. We also formulated all the basic actions modelling the medical services being administered to the patients at various providers: being registered into the system, getting a service, getting the treatment plan updated, switching providers, exiting the system. In the second step we introduced a notion of capacity of a provider and modified the model to make sure that no providers is loaded beyond its capacity.

3.1. Model *M0*: users, medical needs, healthcare providers

Our basic model *M0* is concerned with defining the main elements of our healthcare system prototype, summarised in Table 1.

Table 1. The main elements of the model, defined as "Axioms" in the Event-B language

AXIOMS

axm1: $Individuals \neq \varnothing$
axm2: $Providers \neq \varnothing$
axm3: $Diseases \neq \varnothing$
axm4: $Services \neq \varnothing$
axm5: $partition(Exit, \{Cured\}, \{NotCured\})$
axm6: $Provider2Service \subseteq Providers \times Services$
axm7: $Disease2Service \subseteq Diseases \times Services$
axm8: $Provider2Provider \subseteq Providers \times Providers$
axm9: $Individual2PrimaryProvider \in Individuals \rightarrow Providers$
axm10: $InitialSick \subseteq Individuals \times Diseases$
axm11: $Provider2Service \neq \varnothing$
axm12: $Disease2Service \neq \varnothing$
axm13: $Provider2Provider \neq \varnothing$
axm14: $Individual2PrimaryProvider \neq \varnothing$
axm15: $InitialSick \neq \varnothing$

- *Individuals* which may have various *diseases* and healthcare *providers* that are able to admit and treat patients, offering different types of *services*. In context *C0* of *M0* we define four sets denoting our base types in this model: *Individuals*, *Providers*, *Diseases*, and *Services*. All these sets are non-empty and finite.
- The initial set of medical problems to be addressed is defined through the constant $InitialSick \subseteq Individuals \times Diseases$. The dynamic occurrence of new medical problems is left outside this basic model.
- Each individual has a primary service provider (e.g., a family doctor or some local clinic), that she will contact first and will get allocated to whenever she has a medical need. This is modelled in *M0* through constant $Individual2PrimaryProvider : Individuals \rightarrow Providers$.
- Each healthcare provider has a list of specific services that they can offer within their expertise. This is modelled in *M0* through constant $Provider2Service \subseteq Providers \times Services$.
- Each disease is associated with some services that may be used to address it. This is modelled in *M0* through constant $Disease2Service \subseteq Diseases \times Services$.
- If the current service provider can address some of the user's needs, then the patient will get service. If not, the provider will allocate her to a suitable provider that can address her needs; the relationship between user providers is important here. This is modelled in *M0* through constant $Provider2Provider \subseteq Providers \times Providers$. We only model it as a simple relation on the set of providers, that may include (albeit not explicitly in this model) considerations of the distance between providers, public/private character of the provider, etc.

The current provider (for all individuals) is defined in machine *M0* through the variable *current_provider* $\subseteq$ *Individuals* $\times$ *Diseases* $\times$ *Providers*, indicating the provider currently treating a specific individual for a specific disease. The same individual may have different providers for different diseases. The treatment currently assigned for this individual and disease is recorded through the set of medical services currently anticipated to be performed in the future and modelled as a variable *treatments* $\subseteq$ *Individuals* $\times$ *Diseases* $\times \mathcal{P}$(*Services*). The whole record of all patients, diseases, services they received and providers they visited is modelled as a variable *patients* $\subseteq$ *Individuals* $\times$ *Diseases* $\times$ *Providers* $\times \mathcal{P}$(*Services*). A patient may be removed from the system with an outcome 'Cured' or 'Not cured' and this is modelled through a variable *outcome* $\subseteq$ *Individuals* $\times$ *Diseases* $\times$ *Exit*, where the set *Exit* only contains two values, modelling the two possible outcomes. The set of individuals that have a disease but have not yet been registered into the system is modelled as a variable *sick_not_yet_reg* $\subseteq$ *Individuals* $\times$ *Diseases*.

With these definitions we can also introduce two functions *NumProviders* and *NumServices* giving the number of providers that an individual had to visit for a given disease and the number of services she received for it. These are some basic measures of cost for a given patient and disease in the given model. All these variables are summarised in Table 2.

Table 2. The types of the variables are fixed through 'Invariants' in Event-B

INVARIANTS

inv1: *current_provider* $\subseteq$ *Individuals* $\times$ *Diseases* $\times$ *Providers*
inv2: *treatments* $\subseteq$ *Individuals* $\times$ *Diseases* $\times \mathbb{P}$(*Services*)
inv3: *patients* $\subseteq$ *Individuals* $\times$ *Diseases* $\times$ *Providers* $\times \mathbb{P}$(*Services*)
inv5: *outcome* $\subseteq$ *Individuals* $\times$ *Diseases* $\times$ *Exit*
inv4: *sick_not_yet_reg* $\subseteq$ *Individuals* $\times$ *Diseases*
inv6: *NumProviders* $\subseteq$ *Individuals* $\times$ *Diseases* $\times \mathbb{N}$
inv7: *NumServices* $\subseteq$ *Individuals* $\times$ *Diseases* $\times \mathbb{N}$

The functionality illustrated in Figure 1 is implemented in our model through several 'events': Register_Sick, Change_Provider, Evaluation, Exit_Cured, and Exit_Not_Cured. Each of these events carries its set of guards, i.e. logical conditions that must be satisfied for the event to be enabled, and a set of actions that are executed when the event is triggered. This approach forces the modeller to be explicit about her assumption and opens the door to the possibility of formulating global logical conditions that must be satisfied by the model. Rodin even offers the possibility of automatically proving such invariants, thus helping to demonstrate the soundness of the model. We discuss this in some details in the following.

The registration of an individual at her primary provider is modelled through the event Register_Sick shown in Table 3. The guards of this event specify that the individual i has a disease d and not yet registered. This is checked by looking up the pair individual-disease (i, d) in the sick-not-yet-registered set (grd3), and by checking that the pair is not yet in the registry of current treatments and providers (grd6 and grd7). In this case the registration can proceed and it updates accordingly the treatments and

providers registries. It initialises the number of providers for (i, d) to 1, and the number of services received to 0 (no medical service has been received yet).

Table 3. The Event-B implementation of a patient being registered at her primary provider

Event Register_Sick ⟨ ordinary ⟩ $\widehat{=}$

any

i

d

s

u

where

grd1: $i \in Individuals$true

grd2: $d \in Diseases$true

grd3: $i \mapsto d \in sick_not_yet_reg$true

grd4: $u \in Providers$true

grd5: $u = Individual2PrimaryProvider(i)$true

grd6: $s \subseteq Disease2Service[\{d\}]$true

grd7: $i \mapsto d \notin dom(treatments)$true

grd8: $i \mapsto d \notin dom(current_provider)$true

then

act1: $patients := patients \lhd\!\!+ \{i \mapsto d \mapsto u \mapsto s\}$true

act2: $treatments := treatments \lhd\!\!+ \{i \mapsto d \mapsto s\}$true

act3: $current_provider(i \mapsto d) := u$true

act4: $sick_not_yet_reg := sick_not_yet_reg \setminus \{i \mapsto d\}$true

act5: $NumProviders := NumProviders \lhd\!\!+ \{i \mapsto d \mapsto 1\}$true

act6: $NumServices := NumServices \lhd\!\!+ \{i \mapsto d \mapsto 0\}$true

end

Receiving a medical service is modelled through the event Perform_Service shown in Table 4. The guards of this event specify in all details the types of all the variables used in the implementation, and its action simply updates the treatment registry (showing one less service still to be received), the patient registry (recording the service that has been offered), and the number of services received by the patient for this disease.

In case none of the currently prescribed services are available at the current provider, the model includes the possibility of changing the provider to one who can offer some. This is implemented through the Event-B model Change_Provider shown in Table 5. The guards are indeed checking that no required medical service is available at the current provider (grd10) while there is a provider connected to the current one that has relevant services to offer (grd11). The event updates the registries of current provider and patients, and it increases by one the number of providers seen by the patient for this disease.

Table 4. The Event-B implementation of a patient receiving a medical service

Event Perform_Service ⟨ ordinary ⟩ $\widehat{=}$

any

i
d
p
s
s1
s2
n

where

grd1: $i \in Individuals$true
grd2: $d \in Diseases$true
grd3: $p \in Providers$true
grd4: $s \subseteq Disease2Service[\{d\}]$true
grd5: $s1 \in Disease2Service[\{d\}]$true
grd6: $s2 \subseteq Disease2Service[\{d\}]$true
grd7: $i \mapsto d \mapsto p \mapsto s \in patients$true
grd8: $s1 \in s$true
grd9: $s1 \in Provider2Service[\{p\}]$true
grd10: $s2 = s \setminus \{s1\}$true
grd16: $n \in \mathbb{N}$true
grd17: $i \mapsto d \mapsto n \in NumServices$true
grd18: $i \mapsto d \mapsto s \in treatments$true
grd19: $i \mapsto d \mapsto p \in current_provider$true

then

act1: $treatments := treatments \Leftarrow \{i \mapsto d \mapsto s2\}$true
act2: $patients := patients \cup \{i \mapsto d \mapsto p \mapsto \{s1\}\}$true
act3: $NumServices := NumServices \Leftarrow \{i \mapsto d \mapsto n+1\}$true

end

We skip here the discussion of the remaining three events Evaluation, Exit_Cured and Exit_Not_-Cured, as their semantic is clear from the code. All events are available at `http://users.abo.fi/ipetre/health-eventb-model.zip`.

To make sure that the model is sound, we also introduced a number of additional invariants listed in Table 6. They formulate the following conditions:

- An individual is only registered for a disease at a single current provider;
- An individual is only registered with a single treatment plan for a disease;

Table 5. The Event-B implementation of changing the medical provider

Event Change_Provider ⟨ ordinary ⟩ $\widehat{=}$

any

i

d

p

q

s

n

where

grd1: $i \in Individuals$true

grd2: $d \in Diseases$true

grd3: $p \in Providers$true

grd4: $q \in Providers$true

grd6: $s \subseteq Disease2Service[\{d\}]$true

grd7: $p \neq q$true

grd8: $p \mapsto q \in Provider2Provider$true

grd9: $i \mapsto d \mapsto p \mapsto s \in patients$true

grd10: $s \cap Provider2Service[\{p\}] = \varnothing$true

grd11: $s \cap Provider2Service[\{q\}] \neq \varnothing$true

grd12: $n \in \mathbb{N}$true

grd13: $i \mapsto d \mapsto n \in NumProviders$true

grd14: $i \mapsto d \mapsto p \in current_provider$true

grd15: $i \mapsto d \mapsto s \in treatments$true

then

act1: $current_provider := current_provider \Leftarrow \{i \mapsto d \mapsto q\}$true

act2: $patients := patients \cup \{i \mapsto d \mapsto q \mapsto s\}$true

act3: $NumProviders := NumProviders \Leftarrow \{i \mapsto d \mapsto n+1\}$true

end

- If an individual is registered with a disease, then she is assigned with a medical provider;
- The number of (visited) providers is a variable depending on the individual and the disease;
- The number of (received) services is a variable depending on the individual and the disease.

Remarkably, Rodin offered computational support to prove these invariants, most of them being approved automatically without the modeller's intervention. This indicates that the model was soundly written with respect to its intended functionality.

Table 6. Invariants proving the soundness of the model

INVARIANTS

inv8: $\forall i,d,p,q \cdot ((i \mapsto d \mapsto p) \in current\_provider \land (i \mapsto d \mapsto q) \in current\_provider \Rightarrow p = q)$

inv9: $\forall i,d,s1,s2 \cdot (i \in Individuals \land d \in Diseases \land s1 \subseteq Services \land s2 \subseteq Services \land i \mapsto d \mapsto s1 \in treatments \land i \mapsto d \mapsto s2 \in treatments \Rightarrow s1 = s2)$

inv10: $\forall i,d,j,k,p \cdot ((i \mapsto d) \in sick\_not\_yet\_reg \land (j \mapsto k \mapsto p) \in current\_provider \Rightarrow (i \neq j \lor d \neq k))$

inv11: $\forall i,d,p \cdot ((i \mapsto d \mapsto p) \in current\_provider \Rightarrow (\exists s \cdot s \subseteq Services \land (i \mapsto d \mapsto s) \in treatments))$

inv12: $\forall i,d,m,n \cdot ((i \mapsto d \mapsto m) \in NumProviders \land (i \mapsto d \mapsto n) \in NumProviders \Rightarrow m = n)$

inv13: $\forall i,d,m,n \cdot ((i \mapsto d \mapsto m) \in NumServices \land (i \mapsto d \mapsto n) \in NumServices \Rightarrow m = n)$

3.2. Model *M1*: providers have limited capacity

We introduce now the feature of each provider having a maximum capacity in terms of the maximum number of services they can have on their current provider list at any moment. We introduce this through a *refinement* of our basic model and we use Event-B to prove that the level of occupancy never exceeds the capacity.

Through the refinement all constants and events defined in *M0* are available also in machine *M1*. We only add in *M1* a new (constant) function *Capacity* defined as $Capacity \in Providers \rightarrow \mathbb{N}$ representing the maximum service capacity of each provider, that cannot be exceeded when taking new patients or updating the current patients services.

To indicate that the a provider can not commit to more services than its capacity we introduce several modifications in the events of *M1*.

The event Register_Sick gets a new guard $Occupancy(q) + card(s)) \leq Capacity(q)$ and a new action $Occupancy(q) := Occupancy(q) + card(s)$, where q id the primary provider of the individual being considered for registration.

The event Perform_Service receives a new action $Occupancy(p) := Occupancy(p) - 1$, showing that a service has been performed at the current provider q and so its occupancy is one less than before.

The event Change_Provider gets a new guard ensuring that the new provider q has enough service capability to take the patient: $Occupancy(q) + card(s)) \leq Capacity(q)$. It also gets two new actions $Occupancy(q) := Occupancy(q) + card(s))$ and $Occupancy(p) := Occupancy(p) - card(s))$, showing that provider q committed to new services, while the current provider p has less occupancy.

The event Evaluation gets a new guard $Occupancy(p) - card(s) + card(s1) \leq Capacity(p)$ ensuring that the intended change in treatment can be committed to by the current provider. It also gets a new action $Occupancy(p) := Occupancy(p) - card(s) + card(s1)$.

Finally, the event Exit_Not_Cured gets a new action $Occupancy(p) := Occupancy(p) - card(s)$ reflecting that the set s of services is not longer committed to this provider.

Our modelling of *M1* satisfies the invariant that the occupancy of all providers is always at most their capacity. We proved this result using the built-in theorem proving capability of Rodin. In fact, the invariant was automatically proved by Rodin without our intervention, thus showing the soundness of the model.

The model *M1* is available at `http://users.abo.fi/ipetre/health-eventb-model.zip`.

3.3. Model Statistics

The ratio of automatically discharged proofs to interactively proved ones is illustrated in Figure 2.

Element Name	Total	Auto	Manual
Full Model	62	53	9
M0	36	27	9
M1	26	26	0

Figure 2. The auto-to-manual proof ratio.

As is typical for Event-B developments, the automatic proofs were very useful, as were the cases when no automatic proof was generated. We have discovered in this way several shortcomings or incompleteness of previous versions of the model. The proofs non-automatically discharged for the final version of the model deal with well-definedness issues and ensurance of invariants, and were rather straightforward to discharge manually. The difficult step was in finding the best structures for modeling the different concepts; we found these structures in a (rather typical) feedback loop of modeling and proving.

4. Discussion

We constructed in this paper a prototype formal model of a healthcare system consisting of multiple health service providers and patients with various types of medical needs. We built an Event-B model for such a system, focusing on the dynamics of how a patient navigates the system between various health service providers. The model includes some of the important basic aspects of a healthcare system such as interactions between users and providers, and between providers themselves, mapping a user to a provider, navigation of a user between healthcare providers, and occupancy levels of providers. We see this as a proof-of-concept prototype model demonstrating that some of the complexity of a healthcare system may be captured with an Event-B model. The model can also be seen as a domain-specific distributed scheduling system, and thus of general interest in a wider range of applications.

Other aspects of healthcare systems were also described through formal methods. For example, the interactions between healthcare agencies at the local, state, and federal levels has been done in [16], with a focus on the flow of information between these agencies. Demonstrating the correctness or the safety of medical protocols has been done with the help of formal specifications [17], model checking [18], process definition languages [19], and business process modelling notations [20]. Our model adds to the literature the patient experience throughout her navigation between healthcare providers.

Many details of a typical healthcare system are to be added in a further version of our model, including a continuous inflow of patients, different frequencies of people getting sick, more detailed costs of treatment per patient and per system (including cost of the access to a provider, such as

transportation). We may also model a strategy for changing providers, such as choosing one that maximises the services being offered, or minimises the cost of changing providers. We also consider introducing types of diseases categorised on levels of complexity. All these details may be added directly on the basic models we built in this paper. Having them opens up the possibility of formally investigating notions of robustness of the system, such as the waiting times being upper bounded by an a-priori defined constant. Of high interest will also be to test the dynamics of the system in the case of various extreme scenarios such as epidemics/pandemics or natural disasters, modelled through a spike in the inflow of patients with a certain type of needs or in certain locations. A more detailed version of our model may also allow to prove that certain temporary modifications in the system (such as changes in the capacity of some providers, or adding services to others) may keep the system robust even when in such extreme scenarios. We plan to return to these aspects in a separate study.

References

[1] Abrial JR. Modeling in Event-B: System and Software Engineering. Cambridge University Press, New York, NY, USA, 1st edition, 2010. ISBN 0521895561, 9780521895569.

[2] Abrial JR. The B-book: Assigning Programs to Meanings. Cambridge University Press, New York, NY, USA, 1996. ISBN 0-521-49619-5.

[3] Back RJ, Kurki-Suonio R. Decentralization of Process Nets with Centralized Control. In: Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing, PODC '83. ACM, New York, NY, USA. ISBN 0-89791-110-5, 1983 pp. 131–142. doi:10.1145/800221.806716. URL http://doi.acm.org/10.1145/800221.806716.

[4] Lamport L. Specifying Systems: The TLA+ Language and Tools for Hardware and Soft- ware Engineers. Addison-Wesley, 2003.

[5] Chandy KM, Misra J. Parallel Program Design: A Foundation. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1988. ISBN 0-201-05866-9.

[6] Butler M, Yadav D. An incremental development of the Mondex system in Event-B. *Formal Aspects of Computing*, 2007. **20**(1):61–77. doi:10.1007/s00165-007-0061-4. URL http://dx.doi.org/10.1007/s00165-007-0061-4.

[7] Lanoix A. Event-B Specification of a Situated Multi-Agent System: Study of a Platoon of Vehicles. In: Theoretical Aspects of Software Engineering. IEEE Computer Society, Los Alamitos, CA, USA. ISBN 978-0-7695-3249-3, 2008 pp. 297–304. doi:http://doi.ieeecomputersociety.org/10.1109/TASE.2008.39.

[8] Hoang TS, Kuruma H, Basin D, Abrial JR. Developing Topology Discovery in Event-B. In: Leuschel M, Wehrheim H (eds.), Integrated Formal Methods, volume LNCS 5423. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-642-00255-7, 2009 pp. 1–19. doi:10.1007/978-3-642-00255-7_1. URL http://dx.doi.org/10.1007/978-3-642-00255-7_1.

[9] Kamali M, Laibinis L, Petre L, Sere K. Self-Recovering Sensor-Actor Networks. In: Proceedings Ninth International Workshop on the Foundations of Coordination Languages and Software Architectures, FOCLASA 2010, Paris, France, 4th September 2010. 2010 pp. 47–61. doi:10.4204/EPTCS.30.4. URL http://dx.doi.org/10.4204/EPTCS.30.4.

[10] Salehi Fathabadi A, Rezazadeh A, Butler M. Applying Atomicity and Model Decomposition to a Space Craft System in Event-B. In: Bobaru M, Havelund K, Holzmann GJ, Joshi R (eds.), NASA Formal Methods. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-642-20398-5, 2011 pp. 328–342. doi:10.1007/978-3-642-20398-5_24. URL http://dx.doi.org/10.1007/978-3-642-20398-5_24.

[11] Petre L, Sandvik P, Sere K. Node Coordination in Peer-to-Peer Networks. In: Coordination Models and Languages - 14th International Conference, COORDINATION 2012, Stockholm, Sweden, June 14-15, 2012. Proceedings. 2012 pp. 196–211. doi:10.1007/978-3-642-30829-1_14. URL http://dx.doi.org/10.1007/978-3-642-30829-1_14.

[12] Horsmanheimo S, Kamali M, Kolehmainen M, Neovius M, Petre L, Rönkkö M, Sandvik P. On Proving Recoverability of Smart Electrical Grids. In: NASA Formal Methods - 6th International Symposium, NFM 2014, Houston, TX, USA, April 29 - May 1, 2014. Proceedings. 2014 pp. 77–91. doi:10.1007/978-3-319-06200-6_6. URL http://dx.doi.org/10.1007/978-3-319-06200-6_6.

[13] Kamali M, Höfner P, Kamali M, Petre L. Formal Analysis of Proactive, Distributed Routing. In: Software Engineering and Formal Methods - 13th International Conference, SEFM 2015, York, UK, September 7-11, 2015. Proceedings. 2015 pp. 175–189. doi:10.1007/978-3-319-22969-0_13. URL http://dx.doi.org/10.1007/978-3-319-22969-0_13.

[14] Abrial JR, Butler M, Hallerstede S, Hoang TS, Mehta F, Voisin L. Rodin: an open toolset for modelling and reasoning in Event-B. *STTT*, 2010. **12**(6):447–466.

[15] Maamria I, Butler M, Edmunds A, Rezazadeh A. On an Extensible Rule-Based Prover for Event-B. In: Frappier M, Glässer U, Khurshid S, Laleau R, Reeves S (eds.), Abstract State Machines, Alloy, B and Z, volume LNCS 5977. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-642-11811-1, 2010 pp. 407–407. doi:10.1007/978-3-642-11811-1_40. URL http://dx.doi.org/10.1007/978-3-642-11811-1_40.

[16] Baksi D. Formal interaction specification in public health surveillance systems using pi-calculus. *Comput Methods Programs Biomed*, 2008. **92**(1):115–120. doi:10.1016/j.cmpb.2008.05.007.

[17] ten Teije A, Marcos M, Balser M, van Croonenborg J, Duelli C, van Harmelen F, Lucas P, Miksch S, Reif W, Rosenbrand K, Seyfang A. Improving medical protocols by formal methods. *Artif Intell Med*, 2006. **36**(3):193–209. doi:10.1016/j.artmed.2005.10.006.

[18] Bottrighi A, Giordano L, Molino G, Montani S, Terenziani P, Torchio M. Adopting model checking techniques for clinical guidelines verification. *Artif Intell Med*, 2010. **48**(1):1–19. doi:10.1016/j.artmed.2009.09.003.

[19] Christov S, Chen B, Avrunin GS, Clarke LA, Osterweil LJ, Brown D, Cassells L, Mertens W. Formally defining medical processes. *Methods Inf Med*, 2008. **47**(5):392–398.

[20] Bowles J, Caminati MB, Cha S. An integrated framework for verifying multiple care pathways. In: 2017 International Symposium on Theoretical Aspects of Software Engineering (TASE). 2017 pp. 1–8. doi:10.1109/TASE.2017.8285628.