Privacy-aware design for E-Health Information Systems (DISCUSSION PAPER)

Flora Amato[1], Giovanni Cozzolino[1], Francesco Moscato[2], Vincenzo Moscato[1], Antonio Picariello[1], and Giancarlo Sperlì[1]

[1] University of Naples "Federico II", Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione, via Claudio 21, 80125 Naples, Italy
<name.surname>@unina.it
[2] University of Campania "Luigi Vanvitelli" DiSciPol, Caserta, Italy
francesco.moscato@unicampania.it

Abstract. Many research works had the aim of creating frameworks able to model systems, of defining their requirements and properties, and of verifying their satisfiability. Many approaches involve the usage of the Model Driven Engineering throughout the whole system lifecycle in order to build systems that are correct by construction. Using models as primary artefacts helps to reduce costs and time of development. The possibility of having automatic code generation from model tools enact the possibility to produce (theoretically) bug-free code starting from correct models. In this work we describe the usage of a modelling and verification tool, MetaMORP(h)OSY, for performing quality control in E-Health Domain.

Keywords: Multi agent systems · E-Health · Model Driven Engineering · Formal Verification.

1 Introduction

In many applications, standards and/or regulatory processes are defined to ensure that systems are going to operate as intended. In critical systems this is very important, because an error or a failure could lead to serious damages in terms of human lives, environment and money. In these systems it is necessary to use formal methods, i.e. mathematical techniques used for the specification, development and verification of software and hardware systems.

In recent years, Model Driven Engineering (MDE) has developed a lot, laying its foundations on the massive use of models as its primary artefacts.

In literature, many research works have been devoted to exploit the abstraction resulting from the creation of models in the MDE and in the application of

the modelling techniques on various kinds of systems. In particular, researchers are trying to create frameworks that allow to completely describe systems (and their requirements and properties) and to verify their correctness.

The principle is to use MDE approaches throughout the system lifecycle. The aim is to build systems that are correct by construction. Using models as primary artefacts helps to reduce costs and time of development, and since automatic code generation from model tools are already available, it is clear that is possible to produce (theoretically) bug-free code (starting from correct models). The standard use of the MDE involves the use of a Domain Specific Modelling Language (DSML) for the definition of system models. With this language, different levels of abstraction of the system under study are created. These abstraction levels must somehow contain consistent in formation about the system.

This is the reason why Model-to-Model (M2M) and Model-to-Test (M2T) translation tools are used as bridges between abstraction levels.

These transformations, besides being useful to treat models in the most convenient way, allow us to introduce automation in our process. In order to build a process that leads automatically (or semi-automatically) to the creation of a correct system, it is necessary to solve some problems of high complexity, for example:

- to guide the user in the construction of an high-level model of the system and its specific analysis properties;
- automating the definition and configuration of the analysis process (e.g. by calling the right tools at the right time)

Such approach can be used also to setup, in a very fast and flexible way, many different kind of systems, that can be applied to the analysis of complex domains, like social networks [1–4], big-data, or human-understanding interfaces [5].

Many works [5, 3] exploit Artificial Intelligence tecniques, in order to construct a model able to represent and satisfy system requirements, while other works are model-oriented [6–8]. In recent works, as [9], we propose MetaMORP(h)OSy (Meta-modeling of Mas Object-based with Real-time specification in Project Of complex Systems), a framework aiming to verify if a system model satisfies a given property. This framework is still prone to some issues related to the development of an automatic (or semi-automatic) chain of tools for the analysis and proper construction (model driven) of systems, that can be fixed with a semantic approach for the correlation of produced models [10, 11].

MetaMORP(h)OSy exploits a multi-agent modelling paradigm. A meta-formalism extending UML is used to express actors, system components and requirements. Algorithms are used in the design phase to translate Multi Agent Systems models into appropriate formal models.

2 The methodology

2.1 Methodology Description

MetaMORP(h)OSy implements a methodology for the design, validation and verification of critical systems. For this purpose, it uses techniques and approaches of MDE and MAS. In particular, MetaMORP(h)OSy exploits a multi-agent modelling paradigm. A meta-formalism extending UML is used to express actors, system components and requirements. Algorithms are used in the design phase to translate MAS models into appropriate formal models. The definition of requirements allows the choice of appropriate Observers to be performed for model analysis. After the validation of the design models, vertical transformations are implemented, which produce stubs for the generation of the system at run-time. The framework must generate monitors during the design and execution phase in order to allow the verification of the requirements. In particular, the monitors in the execution phase must compare the behavior of a system under analysis with the one predicted in design analysis.

2.2 Description Language

METAMORP(h)OSy provides the same graphic language for the system specification and its requirements formulation. This language is formally defined by a meta-model. The MetaMORP(h)OSy profile must allow the definition of:

- structural and behavioural views of the system;
- system properties and requirements to be verified;
- methods, techniques and metrics for analysing or measuring properties and requirements;
- expected and measured workloads of the systems.

Being MetaMORP(h)OSy designed to be used for critical systems, it allows the specification of temporal behaviour of agents and real-time properties[12]. The name of the modelling profile is RT-AML (Real-Time Agent Modelling Language). RT-AML describes MAS using a UML-based language. MetaMORP(h)OSy uses the BDI (Beliefs Desires Intentions) paradigm to describe agents[13].

Agents are characterized by their beliefs, the objectives they want to achieve and the plans available to achieve them. RT-AML profile uses four diagrams for MAS behaviours description: Class diagrams, RT Agent Diagrams, RT-Activity diagrams and RT-Sequence diagrams. Class diagrams are the same of UML class diagrams. They are used when it is not useful to describe objects as agents (for example in case of passive entities). The RT-Agent diagrams are the core of the RT-AML profile. They describe agents' structures, goals and beliefs. In addition, they declare the actions and the plans they can execute.

The RT-Activity diagrams allow for description of agents plans. The RT-Sequence diagrams describe agents' collaborations (and/or competitions). Their main goal is to define exchanges of messages and events (real-time stimula)

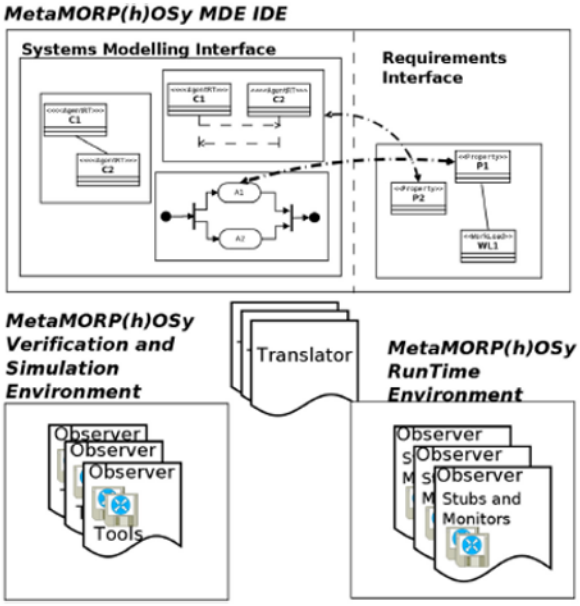

Fig. 1. Architecture of MetaMORP(h)OSy

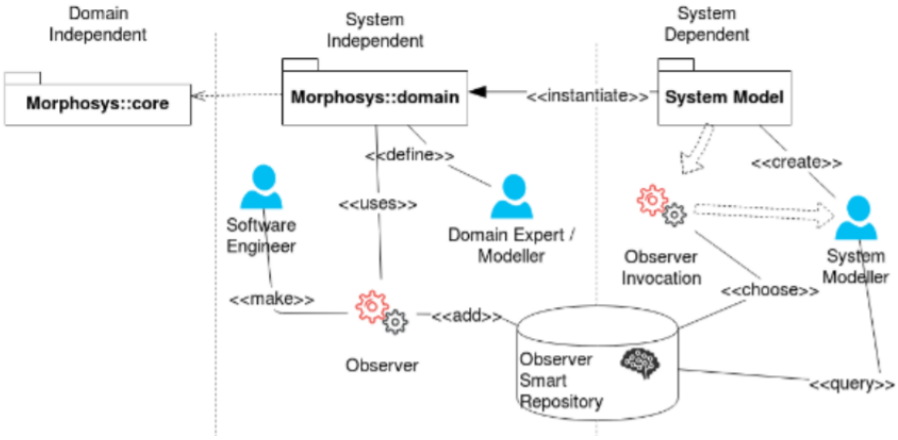

Fig. 2. MetaMORP(h)OSy component architecture

during execution of agents plans. As seen in Amato and Moscato 2015[9], main elements of RT-Agent diagram meta-model are depicted in Figure 2.

The *AgentRT* stereotype defines the structure of agents. The *PlanRT* stereotype defines the plans, which in turn define the behaviour of agents. Plans are associated with objectives, and typically these are common to multiple agents, who will work together to achieve them.

The *DgoalRT* stereotype defines decidable targets for *AgentRTs*, where a decidable target is a target whose reachability can be achieved under real-time constraints. The *BeliefRT* stereotype is used to model what the agent knows, in terms of internal status, the information about other agents and the external environment.

Stereotypes needed to model agent plans are defined in the *RT-Activity* diagram profile. *RT-Activity* Diagram inherits all these elements from the *UML Activity Diagram*, redefining states and transitions to specify real-time constraints. There are six stereotypes created ad-hoc:

ActionStateRT, *InitialState*, *FinalState*, *TransitionRT*, *SendObjectRT* and *ReceiveObjectRT*. When an agent starts a plan, he has information about his surroundings.

These Beliefs usually make up the initial state of the activity diagram, and they are specified as properties of the Initial State Stereotype. *FinalState* refers to a *DGoalRT*, and it is the final state of the activity diagram. Usually the temporal properties are relative to the final state. *ActionStateRT* derives from the classic state of the Activity Diagram UML. It is basically a state subject to real-time constraints. The stereotype in fact includes properties such as *ExecActionTime*, that is the time it takes to execute the action. In particular, to describe situations in which an agent sends or receives objects (messages or events), the stereotypes *SendObjectRT* and *ReceiveObjectRT* have been introduced. The last stereotype introduced for Activity Diagrams is TransitionRT, which is used to define time constraints on state transitions.

2.3 Model Components

Main elements of the RT-Sequence Diagram are *StimulusRT* and *MultiStimulusRT*. Before introducing the stereotype *StimulusRT* we recall the fact that a sequence diagram is basically composed by objects and lifelines. In this context, stimuli are messages that objects can send to themselves. There are five types of stimuli: *Call*, *Send*, *Return*, *Create* and *Destroy*. *StimulusRT* is used, therefore, to define stimuli subject to real-time constraints. Each *StimulusRT* element has the following properties:

- **StartTime**: when the decision to send a message during the pian is made.
- **SendTime** : when the message is sent.
- **TransmissionTime** : the time required to transmit the message.
- **StartReceiveTime** : the time the receiver starts receiving the message.
- **EndReceiveTime**: the time when the receiver has finished receiving the message.

– **Deadline** : the deadline for broadcasting.
– **MessageSync**: used to specify whether the message is synchronous or not.

MultiStimulusRT is derived from the *StimulusRT* stereotype, and it is used to define redundancy in message reception. MetaMORP(h)OSy has separate profiles for defining requirements and properties of systems. This allows to demand different requirements on the same system at different times, which is why requirement and property specifications are defined outs ide of the agent stereotypes.

A requirement is a list of properties that Observers will analyse on the models (at design-time or run-time). The properties can be functional or non-functional.

MetaMORP(h)OSy basic profile for properties defines stereo types for properties such as Availability, Reliability, Schedulability, Performances, etc. Obviously, if the properties are non-functional, it is necessary to define metrics for their definition, evaluation and monitoring. Users can add other metrics and properties to the basic MetaMORP(h)OSy profile.

Proper modules called Obervers (whose structure is defined in the modelling profile too) choose the best suitable Translator in the framework to execute proper Model Transformations and to verify requirements.

Since MetaMORP(h)OSy covers all system life cycle, it uses particular Observers at run-time for monitoring and testing purposes. So, Observers are divided by the profile into: Design Time Observers and Run Time Observers. They can be further specialized depending on translation algorithm, and analysis methods are used to analyse properties and requirements on system components.

Observers can be associated to Structural and behavioural components and obviously to the property to analyse or to monitor, as well as to the metrics used to collect and to produce results. The Observers introduce the intelligence needed to automate the system analysis. They are generated from the analysis models and they are dependent on the kind of the analysis that must be performed.

The Observers are in charge of automatically establishing and realizing the specific MDE approach to validation and verification of the system under study. They use the UML models of the system and the information contained in the analysis models in order to choose a suitable analysis process and generate the formal models needed to fulfil the analysis objectives. They enact the process, also invoking the proper analysis/solution tools. The overall MetaMORP(h)OSy component architecture is depicted in figure 2.

3 Application to e-health domain

This section describes the specialization of MetaMORP(h)OSy methodology applied in processing data[14, 15] for the E-Health domain, using results from Amato and Moscato 2015[9]. The first step is to extend the MetaMORP(h)OSy RT-AML profile for the specific application domain. For this purpose, it is necessary to extend both the AgentRT and BeliefRT stereotypes.

These new Agents specialize the *AgentRT* stereotype are:

- **Patient** represents the entity that possess Electronic Health Record and grant the authorizations for the access to records by other agents;
- **DataController**: users can use this stereotype in order to declare agents like Hospitals that implement the E-Health system ;
- DataProcessor stereotype defines providers for records management and storing;
- **DataSubProcessor** usually refers to Cloud Storage providers;
- **Physician** is the entity related to doctors that interacts with patients. In particular, Family doctors and doctors who are eventually involved in emergencies specialize this stereotype.

The *BeliefRT* is specialized introducing:

- **HealthRecord**: the electronic health record of the patient;
- **EnCryptedHR**: encrypted version of the health record;
- **HiddenRecord**: used for hiding the health record to any agent able to read the patient's data;
- **EncryptKey**: used to encrypt data;
- **DecryptKey**: used to decrypt data;
- **AccessAuth**: physicians authorizations;

Please note that access to encrypted data is achieved through the coupled use of encryption and decryption keys. In Amato and Moscato 2015[9], verification of Privacy has been traced back to a problem of state reachability of some goals with given configuration of beliefs.

The goal is to obtain the following behavior: if an agent executes a plan, whose goal is reachable. In case of access of a patient's record without having any valid authorization, the verification must fail.

In Amato and Moscato 2015[9] there are some simplifying assumptions for the requirement verification:

- communication network is considered secure;
- in the example, only one health record is considered;
- encryption exploits asymmetric algorithms;
- key sharing protocols are considered safe and they will not be explicitly considered.

The problem is the storage of the health record on a third-party storage server (DataProvider in the following).

The generic owner of the record is a patient (thePatient in the following). ThePatient can always read its Health Record (HR). In order to store the record on the storage service provider, a previous encryption of the record has to be done.

The encryption is executed directly on the patient's smart device. The doctor (theDoctor) can access the record only if it is authorized by thePatient. By the way, the doctor cannot own the description key of thePatient. For this reason, during the authorization process, theDoctor shares its encryption key with the Patient, so that it can encrypt the HR using this key.

In order to manage emergencies, thePatient stores on the DataProvider another encrypted version of its HR. This time, in the encryption process, a common key is used.

DataProvider, thePatient and theDoctor have been defined as agents, so they must be described in a structural view with appropriate Agent Diagrams. A BeliefDesire-Intention logic approach has been used in defining Agent Diagrams.

Its beliefs contain information about thePatient's, theDoctor's and emergency keys. In this diagram only three plans have been considered:

- Save: for saving the record on DataProvider;
- ReadRecord: to read the record from DataProvider;
- Authorize: to grant authorization to doctors to retrieve encrypted records from DataProvider.

Each Plan is associated to a Goal. It is reached when the final action of the plan is executed. It is important to notice that beliefs with the same name share the same information between agents.

For each agent plan, the user must define an activity diagram. In the following it will be reported activity diagrams of Save and Authorize plans. Please recall the fact that plans are used to reach a goal, and very often this is obtained thanks to the communication between agents. This concept brings the notion of synchronization, expressed in the activity diagrams through guards on control flow edges. Guards contain declarations of events that are asserted or waited during plans executions.

The activity diagram we are going to analyse is the Save plan diagram, depicted in Figure 3. The objects on the left side of an activity diagram correspond to the diagram inputs, while the objects on the right side correspond to the diagram outputs. Please notice that in the Save plan there are three parallel requests for saving the record encrypted using: its encryption key, its common encryption key for emergencies and its authorized doctor's encryption key.

The second activity diagram, depicted in Figure 4, describes the Read plan.

It shows actions that are executed by theDoctor when it requests a normal authorization to read the health record. Please notice that in this diagram there are two final states: one linked to the failure of the plan (ErrorAuth?) and one linked to the success of the plan (after receiving OkAuth!, it starts the decryption, and then theDoctor can access the record).

To finish the behavioral description of the system, proper sequence diagrams must be provided, in these diagrams, asserted events, which appear in agents' activity diagrams, are declared.

At the end of structural and behavioral description of the system, the user must prepare an Observer Diagram to declare the type of the analyses and monitoring activities it want to perform on the system.

MetaMORP(h)OSy offers many Observers which are already implemented. In some cases, it could be useful to define an ad-hoc Observer algorithm, such as it was done in Amato and Moscato 2015[9], in order to specify particular requirements on the model (like data privacy). In the following, it is reported

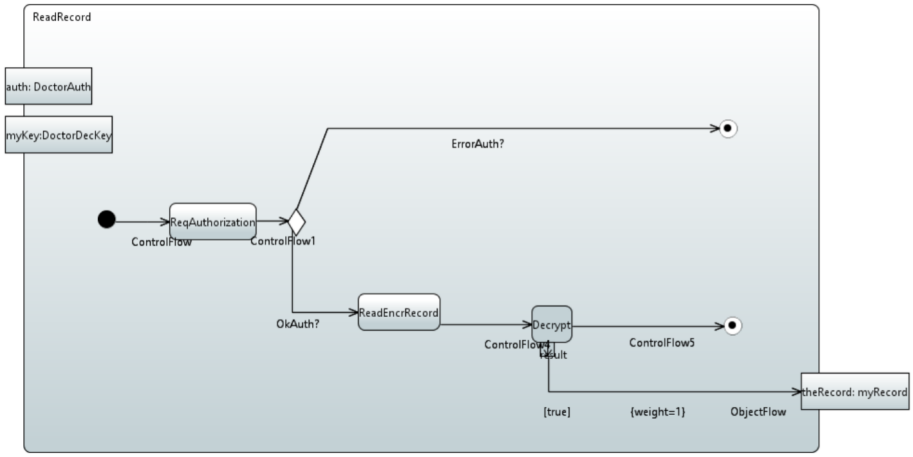

Fig. 3. Save Plan Activity Diagram

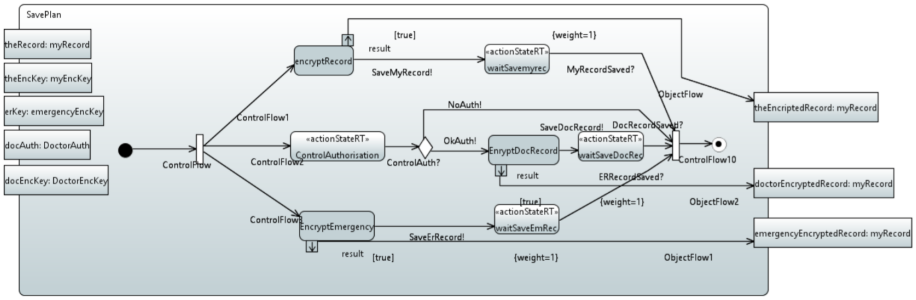

Fig. 4. Read Record Plan Activity Diagram

the solution proposed in the previously cited article. This algorithm translates the design model into Timed Automata.

A timed automaton is a finite state machine extended with a finite set of real-valued clocks. During a run of a timed automaton, clock values increase all with the same speed. Along the transitions of the automaton, clock values can be compared to integers. These comparisons are used in order to evaluate automaton transitions.

Without deepening into details, the Observer algorithm translates the RT-AML models D into a Timed Automata. This allows the use of Timed Automata analysis tools for verifying requirements. The last thing that has to be done is the translation of the data privacy requirement in an analyzable property for a Timed Automata Model Checker.

In Amato and Moscato 2015[9], the privacy requirement is expressed in terms of state reachability. The tool that the authors of the article used as model checker is Uppaal. It is a powerful tool for modelling, validation and verification of real-time systems modelled as timed automata. Uppaal supports Timed-Computational Tree Logics (CTL) for queries specification and model checking. Again, without deepening into details, it is possible to translate privacy requirements into a CTL formula. This formula become our property to be analyzed by using Uppaal. In Amato and Moscato 2015[9], two CTL formulas were defined in order to verify the privacy requirement, and the Observer has assured that the model previously described satisfies both. In the second example made in chapter 2, one of the CTL formulas used in Amato and Moscato 2015[9] was translated in a set of Prolog rules, simplifying their function.

4 Conclusions

In this work we have shown that the adoption of Multi-Agent System models, combined with the use of Model-Driven Engineering can provide significant advantages. The combination of MAS and MDE leads flexibility and intelligence in the analysis of properties and requirements of systems.

Frameworks like MetaMORP(h)OSy helps us in automating (or semi-automating) the verification and validation process. When adopting this framework, the user only needs to derive his particular modelling profile from the standard Meta MORP(h)OSy's RT-AML meta-model. As a further step, the user must produce proper diagrams and observers, in order to analyse system properties and requirements in the correct way.

In the last part of the work, an architectural description of MetaMORP(h)OSy is provided, with specific focus on its RT-AML profile specialization for particular purposes. It has been described, in addition, an example of MetaMORP(h)OSy application in E-Health domain.

Through this work, the advantages of using MetaMORP(h)OSy, in system property analysis have been highlighted for e-Health domain. We firmly believe that this kind of applications will be adopted and developed not only in the academic world, but certainly in industry world, too.

5 Acknowledge

This work was funded by "Synergy-net: Research and Digital Solutions against Cancer" Project (funded in the framework of the POR Campania FESR 2014-2020).

References

1. A. Amato, G. Cozzolino, and M. Giacalone. Opinion mining in consumers food choice and quality perception. *Lecture Notes in Networks and Systems*, pages 310–317, 2020.
2. A. Amato, W. Balzano, G. Cozzolino, and F. Moscato. Analysis of consumers perceptions of food safety risk in social networks. *Advances in Intelligent Systems and Computing*, pages 1217–1227, 2020.
3. A. Castiglione, G. Cozzolino, V. Moscato, and G. Sperli. Analysis of community in social networks based on game theory. pages 619–626. Institute of Electrical and Electronics Engineers Inc., 2019.
4. A. Amato and G. Cozzolino. Trust analysis for information concerning food-related risks. *Lecture Notes on Data Engineering and Communications Technologies*, pages 344–354, 2019.
5. R. Canonico, G. Cozzolino, A. Ferraro, V. Moscato, A. Picariello, F.R. Sorrentino, and G. Sperlì. A smart chatbot for specialist domains. *Advances in Intelligent Systems and Computing*, pages 1003–1010, 2020.
6. Madeleine Faugere, Thimothee Bourbeau, Robert De Simone, and Sebastien Gerard. Marte: Also an uml profile for modeling aadl applications. In *12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007)*, pages 359–364. IEEE, 2007.
7. Douglas C Schmidt. Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY-*, (2):25, 2006.
8. Richard Soley et al. Model driven architecture. *OMG white paper*, (308):5, 2000.
9. Flora Amato and Francesco Moscato. A model driven approach to data privacy verification in e-health systems. *Trans. Data Privacy*, (3):273–296, 2015.
10. F. Amato, G. Cozzolino, V. Moscato, and F. Moscato. Analyse digital forensic evidences through a semantic-based methodology and nlp techniques. *Future Generation Computer Systems*, pages 297–307, 2019.
11. G. Cozzolino. Using semantic tools to represent data extracted from mobile devices. pages 530–536. Institute of Electrical and Electronics Engineers Inc., 2018.
12. Hermann Kopetz. *Real-time systems: design principles for distributed embedded applications*. Springer Science & Business Media, 2011.
13. Michael Wooldridge. Agent-based software engineering. *IEE Proceedings-software*, (1):26–37, 1997.
14. A. Amato, G. Cozzolino, and V. Moscato. Big data analytics for traceability in food supply chain. *Advances in Intelligent Systems and Computing*, pages 880–884, 2019.
15. F. Amato, G. Cozzolino, F. Moscato, V. Moscato, A. Picariello, and G. Sperli. Data mining in social network. *Smart Innovation, Systems and Technologies*, pages 53–63, 2019.