ADDRESSING SECURITY REQUIREMENTS THROUGH MULTI-FORMALISM MODELLING AND MODEL TRANSFORMATION

Miriam Zia, Ernesto Posse and Hans Vangheluwe
School of Computer Science, McGill University, 3480 University Street, Montreal, Canada

Keywords: Multi-formalism modelling, model transformation, verification, security requirements, e-health.

Abstract: Model-based approaches are increasingly used in all stages of complex systems design. In this paper, we use multi-formalism modelling and model transformation to address security requirements. Our methodology supports the verification of security properties using the model checker FDR2 on CSP (Communicating Sequential Processes) models. This low-level constraint checking is performed through model refinements, from a behavioural description of a system in the Statecharts formalism. The contribution of this paper lies in the combination of various formalisms and transformations between them. In particular, mapping Statecharts onto CSP models allows for combination of the deterministic system model with non-deterministic models of a system's environment (including, for example, possible user attacks). The combination of system and environment models is used for model checking. To bridge the gap between these Statechart and CSP models, we introduce kiltera, an intermediate language that defines the system in terms of interacting processes. kiltera allows for simulation, real-time execution, as well as translation into CSP models. An e-Health application is used to demonstrate our approach.

1 INTRODUCTION

Model-driven approaches are becoming more prevalent in complex software systems engineering. Methodologies are developed which control the complexity of the software process and address system specifications at higher levels of abstraction. Tools are becoming available to automate the transformation between models, which possibly reside at different abstraction levels (Muller et al., 2005; de Lara and Vangheluwe, 2002), and to support their analysis (Bengtsson et al., 1995; Gardey et al., 2005). By hiding implementation details, and manipulating models described at an appropriate level of abstraction using suitable (possibly domain-specific) modelling formalisms systems are developed from high-level concepts, though model refinements and analysis.

Model-based approaches are increasingly used in all stages of systems design, and in recent years, have evolved around the development of dependable systems that must respect safety or reliability requirements, amongst others. While various methodologies have been designed to tackle these dependability issues (Mustafiz et al., 2006), other emerging constraints, such as security and privacy, need to be addressed. Indeed, as the Internet evolves, from being a basic communication mechanism to being a tool for effective business operation and service provision, more sensitive information is digitized. As a conse-

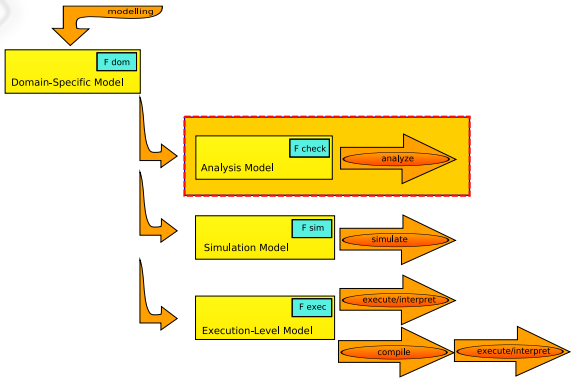

Figure 1: A model-driven engineering approach. We focus on the verification branch.

Zia M., Posse E. and Vangheluwe H. (2007).
ADDRESSING SECURITY REQUIREMENTS THROUGH MULTI-FORMALISM MODELLING AND MODEL TRANSFORMATION.
In *Proceedings of the Second International Conference on Software and Data Technologies - SE*, pages 129-137
DOI: 10.5220/0001347201290137

quence, various e-applications require the access to, manipulation, and transfer of this digital data, thus making it susceptible to third-party attacks, or even user abuses. Such e-systems must, therefore, be: 1) secure, and designed in such a way that agents (users or programs) can only perform allowed actions; and 2) privacy-sensitive, and effectively protect personal data from unauthorized access or disclosure.

Research has been carried out in the study of computer security and electronic privacy, and has mostly tackled problems in the network and communication domains. The work in that area is geared towards representing security threats in terms of attack graphs (Sheyner et al., 2002) and in developing model-checking methods for the verification of the security of communication (Lowe, 1996) or registration protocols (Germeau and Leduc, 1997). The latter require detailed model descriptions of the protocols, in languages such as the Communicating Sequential Processes (CSP) (Hoare, 1985) or LOTOS (ISO, 1989), to be checked with verification tools, such as the Failures Divergences Refinement Checker (FDR2). However, what these approaches do not offer is a general, application-independent methodology for representing a security problem at a high level, in such a way as to abstract complex details from, for example, a designer who needs to check that specific properties hold on a new security protocol being developed.

In (Yeung et al., 2005) and (Roscoe and Wu, 2006), transformations are defined, which translate high-level behavioural models of a system into models suitable for verification with FDR2. This work represents a step towards a general methodology for the verification of requirements through multi-formalism modelling and model transformation. An example of this approach has been proposed for designing controlled anonymous applications (Naessens, 2006). Our contribution lies in introducing a multi-formalism model-driven methodology which supports requirements analysis and refinement, simulation-based performance analysis, and code synthesis. It is geared towards security constraint analysis and verification.

At each step of a multi-formalism design methodology, the system under consideration is modelled in the most suitable formalism (Figure 1). At the highest level, a domain-specific model appropriately describes the domain problem. From this model, one of two models can be derived. If the problem's state-space is too large for model-checking, a simulation model is constructed in a formalism with strong simulation capabilities. The gathered simulation results are used for performance analysis or are checked against a set of rules derived from the requirements.

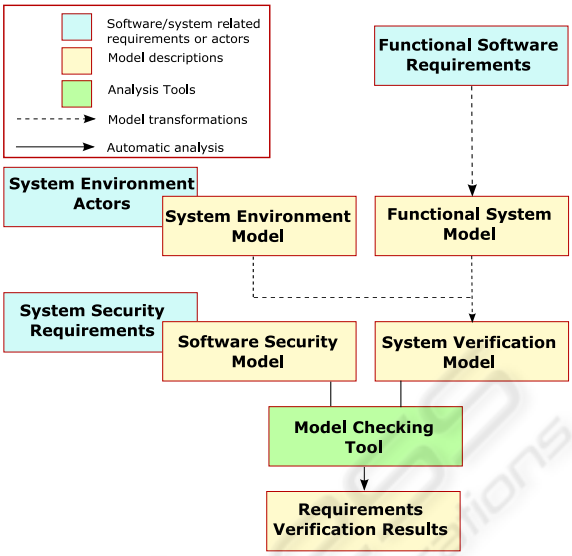

Figure 2: A model-based process for security requirement analysis.

Alternatively, an analysis model may be constructed in a formalism amenable to formal analysis and verification of the system requirements, covering all possible behaviours. Ultimately, an execution model is synthesized from the domain model, thus providing a continuous design process from the domain problem to the analysis model, and finally to the deployed system. All steps in the evolution, from requirements and constraints, to final application, represent the system at different levels of abstraction. Transformations between models represent refinements.

We present such a process to deal with security requirements. As depicted in Figure 2, this process allows for low-level constraint checking, through model refinements, from higher-level system specifications. At a high level of abstraction, we have a behavioural description of the system in a known formalism, such as in Statecharts. At a lower level, verification of desired properties is performed with existing tools, such as FDR2, on CSP models. Since models in Statecharts only depict required system behaviour and not that of the environment actors on this system, our approach introduces a middle step in the design, which acts as a central hub for model transformations. Although Statechart models could be verified at this level, they are first translated into a language which allows for the concise description of non-deterministic (environment) behaviour. These intermediate models are described in the modelling language kiltera (Posse and Vangheluwe, 2007). The nature of the kiltera language allows a straightforward translation into a CSP model, which is subsequently checked to ensure that, given the actors involved in the environment, the original system design adheres

to the required security constraints.

The paper is structured as follows. Section 2 presents background concepts relating to the CSP and the kiltera modelling languages, as well as the e-health case study chosen to illustrate our methodology. Section 3 proposes our modelling and verification process, for designing a secure infrastructure. The steps involved in this process are illustrated on an e-health use case. Finally, we draw some conclusion about our methodology in Section 4.

2 BACKGROUND

Since UML Sequence Diagrams (Rumbaugh et al., 1999) and Harel's Statecharts (Harel, 1987) are widely used modelling formalisms, in this section we focus, on the one hand, on giving an overview of the CSP and kiltera languages, and on the other hand, on introducing the e-health example used in this paper.

2.1 Multiple Formalisms

The Communicating Sequential Processes (CSP) and kiltera languages were both designed for describing systems of communicating components. In these languages, a process is described in terms of the possible interactions it can have with its environment. Interactions are described in terms of instantaneous atomic synchronizations, or events. Each process is an independent computational unit and proceeds concurrently with all other processes. Furthermore, a process is not necessarily a purely sequential computation, as it may be itself composed of parallel subprocesses. In CSP, a process is defined by its interface, grouping all the events it may engage in, and through which it interacts with other processes. In kiltera, a process is a modular component with a well defined interface consisting of a set of ports, through which all interactions occur.

In CSP, three semantic models are available to reason about observable behaviour of processes, and to provide foundations for specification and verification techniques. The simplest of these mathematical models is the traces model. In CSP, each process is represented by its set of finite sequences of communications it can perform, or set of traces. In the traces model, system constraints are specified on traces, by characterizing which traces are acceptable, and which are not. These specifications are then used for traces refinement.

In contrast, kiltera's focus is on simulation of dynamic systems with an explicit notion of time and structural change. Simulation is typically used for performance analysis (non-functional requirements). kiltera combines the interaction and mobility model of the π-calculus (Milner et al., 1989) with the time model of Timed-CSP (Reed and Roscoe, 1986) in a single framework. Its formal semantics are given in terms of timed-labelled transition systems, which provides a basis for formal system analysis as well as a framework to relate kiltera to other languages and formalisms.

2.2 e-Health Case Study

adapID[1] is a project aimed at developing a framework for secure and privacy-preserving applications based on the Belgian e-ID card. The electronic ID card in Belgium is the first smart card with advanced public key cryptology capabilities, and contains digital personal data that may identify the card holder, and possibly lead to some privacy risks for the individual citizens. The focus of the project is on the development of a framework for advanced secure applications of the e-ID which do not reduce personal privacy of the users. Our focus primarily lies in e-health applications, and the security constraints they need to address.

E-health addresses the problem of improving the quality and efficiency of health care, as well as the reduction of corresponding costs, through the innovation of information and communication technologies. These advances all rely on the development of information systems, which store patient related information. Since a patient's medical history is accessible by care givers, it aids them in administering more appropriate services. However, abuses of the system pose severe security and privacy risks, for example if the e-health application is misused or if unauthorized access to a medical record is obtained.

2.2.1 Use Case: Visiting a Pharmacist

The following introduces a simplified "visiting a pharmacist" e-health system to be developed. We will use this case study to illustrate the methodology described in Section 3, and verify the security constraints it must adhere to.

The actors in one of the use cases in this case study are a pharmacist and a patient. A patient receives an electronic prescription from a doctor, who signs it with his e-ID card to guarantee its integrity. To get the prescription filled, the patient visits a pharmacist who uses his e-ID card to prove his pharmacist credential. Once the pharmacist shows his recognition, the patient anonymously shows him the prescription, with-

[1] https://www.cosic.esat.kuleuven.be/adapid/

out his or the doctor's identity ever being revealed. The drugs are issued to Patient upon verification of the validity of the prescription.

The security requirements related to this use case are as follows:

1. A malicious patient or pharmacist should not be able to successfully show a non-valid credential;
2. A malicious patient should not succeed in submitting a forged, altered or already filled prescription.

In either case of abuse, it should be possible to deanonymize the culprits.

3 MODEL-BASED PROCESS

This section describes the multi-formalism modelling process illustrated in Figure 3. Note that this is a concrete refinement of Figure 2 Note also that we only describe a simple use case her. Related work (Whittle and Schumann, 2000; Whittle and Jayaraman, 2006) shows how use cases (representing requirements) can be described using a collection of sequence diagrams and activity diagrams. The latter show allowed sequences of use cases for subsequent translation into Statecharts.

3.1 Overview

Initially, the functional software requirements of a system, for which we want to address security constraints, are described by a Sequence Diagram. A Statecharts *functional system design model* satisfying the requirements is derived from this description to represent the structure and behaviour of the desired system.

Secondly, we construct an environment model in kiltera, to describe *all possible* interactions of the user with the system. We now have two models: a deterministic Statechart model of the functional system and a non-deterministic kiltera model of the system environment. Then, the Statecharts model is translated into a kiltera one with an equivalent behaviour. The kiltera *functional system model* is then integrated with the *system environment (actors) model*. Grouped together, these kiltera models are a representation of the behaviour of environment and the system. At this point, it is possible to carry out extensive simulations (1) to gain insight into the dynamic behaviour of the system and (2) to analyze performance metrics (non-functional requirements). This aspect is not elaborated further in this paper.

Thirdly, a second set of model transformation rules is used to transform the kiltera *behaviour and environment model* into an equivalent, but lower-level CSP representation. Combined with a CSP description of security requirements, a *system security verification model* is checked with the tool FDR2. Finally, the output from the model checker confirms whether the constraints are satisfied in the proposed design of the system, or whether further modifications need to be applied to the top level Statecharts model.

In the following, we further discuss the steps involved in this proposed methodology, and illustrate each step with an application to the use case described in Section 2.2.1.

3.2 Capturing Requirements: Modelling with Sequence Diagrams and Statecharts

Our model-driven process begins with a specification of the system requirements, in the form of a Sequence Diagram. The Sequence Diagram, illustrated in Figure 4 summarizes the required behaviour of the "Visiting a pharmacist" use case, and represents the flow of actions between the two actors of the system (the pharmacist and the patient). From this specification, a Statechart model (Figure 5) is automatically constructed as a very preliminary system design of the final application. The sending and receiving of events in the different orthogonal components corresponds to message sending in Figure 4. The translation between Sequence Diagrams and Statecharts is described in (Sun, 2007). Referring back to Figure 1, another advantage of using this formalism is that we can synthesize executable code from it (Feng, 2004).

3.3 Adding Non-deterministic Environment Scenarios: Statecharts and kiltera

At this point, properties of the design encoded in the Statechart model could be verified directly. Non-deterministic formalisms such as kiltera are more suitable (than Statecharts) to describe large numbers of environment scenarios (corresponding to non-functional constraints) compactly. We prefer not to use Statecharts for verification. For this reason, we propose a series of model transformations and refinements, resulting in a CSP model ready for checking. Note that we are currently working on a formalization of these transformations. The first such transformation generates a description in kiltera. Here we provide a brief sketch of how Statecharts are translated into kiltera processes. Each Statechart blob (AND or OR state) is mapped to a kiltera process, preserv-

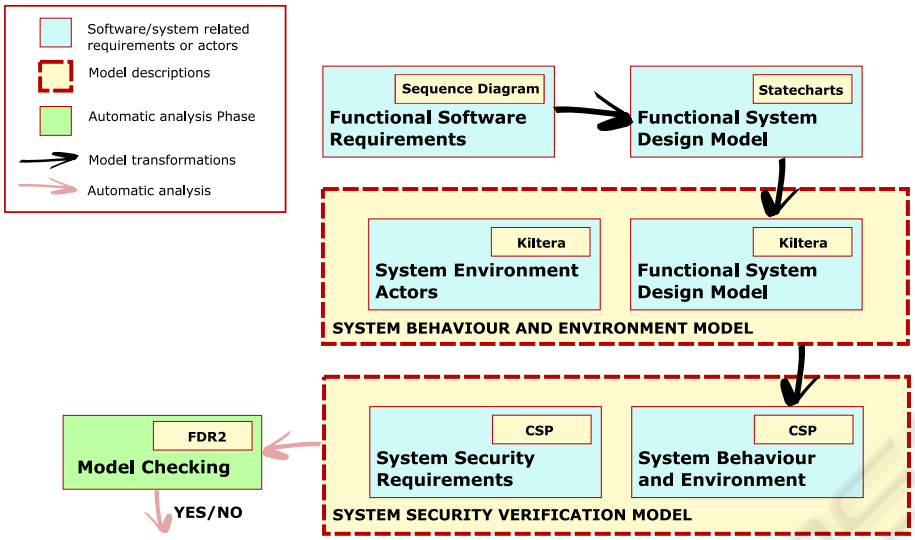

Figure 3: From behaviour models to verification models: a multi-formalism and model transformation approach for addressing security constraints.

ing the nesting structure of the Statechart. For each composite blob, we create an additional kiltera process called a "relay", which keeps track of the currently active substate(s), takes care of events relevant to that blob, and forwards messages intended for other "blobs". Each kiltera process corresponding to a blob has two ports: one which connects it to its parent "relay" and one output port. All messages go through these ports and therefore the structure of the overall kiltera process is completely modular, unlike a Statechart which may have transitions that cross a blob's boundaries. Finally, the obtained kiltera model gets simplified. This simplification can increase simulation and verification performance, albeit at the cost of sacrificing some modularity. The advantage of using kiltera at this level is due to its simulation engine. Such simulation capability is not supported for CSP models, for which the closest equivalent to a simulator would be the tool ProBE (Formal Systems Europe Ltd., 2003), an animator for CSP processes allowing the user to manually explore the behaviour of models.

Running simulations of a kiltera model is useful for exploring a subset of the system's entire statespace, or for performance analysis. As a matter of fact, we can define performance metrics to be checked on the simulation results. Examples of a performance metric are the likelihood that a security constraint in breached, or an estimate of how many security breaches occur over the simulated time interval. The system modelled in Statecharts (Figure 5) consists of self-contained interacting components, which can conceptually be considered as communicating processes. The flow of actions between these processes defines the sequence and type of interactions that can occur between them, and is used to define the process event interface. These processes can be seen as synchronizing over the set of events which are common to the orthogonal Statechart components. Note that we are using Statemate Statecharts for historical reasons. Future work will map onto UML Statecharts. This has the advantage that it will make classes (whose instances appear in Figure 4) explicit. Currently, those classes are implicitly associated with the Statemate Statecharts's orthogonal components. From Figure 4, we note that the interactions taking place between the two actors of the system, can be viewed as occurring between a Pharmacist and a Patient processes. Below is the description of the Patient process in kiltera:

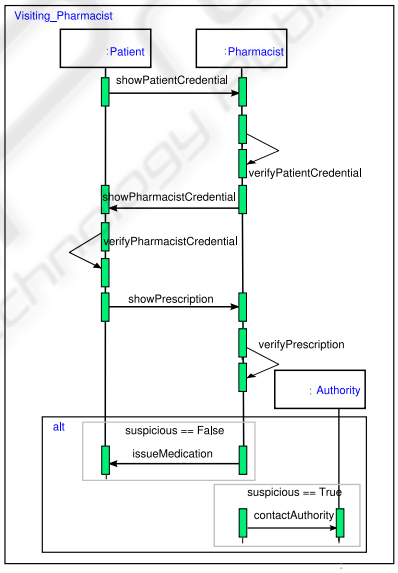

Figure 4: UML Sequence Diagram representing the "Visiting a pharmacist" system requirements.

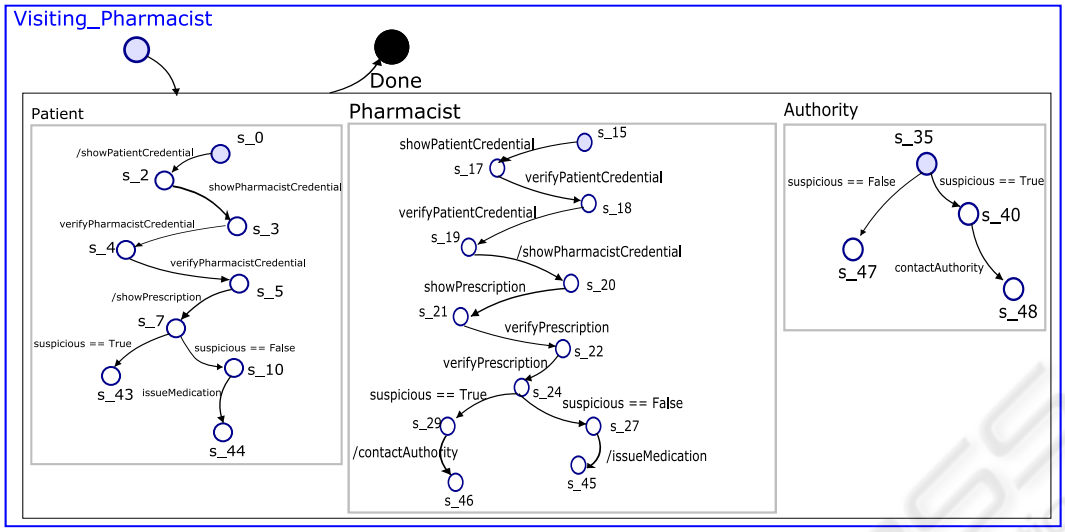

Figure 5: A Statechart. model of a design satisfying the requirements.

```
process Patient[secureComm, authority]
(PatientCredential, Prescription):
 sync send PatientCredential to secureComm ->
  sync receive PharmacistCredential from secureComm ->
   if PharmacistCredential = "pharmacistCred" then
    sync send Prescription to secureComm ->
     sync receive result from secureComm ->
      if result = "FillPres" then
       Prescription := "filledpres" ->
        Patient[secureComm, authority]
         (PatientCredential, Prescription)
   else
    send "reportPhCredFraud" to authority
```

The system environment is introduced into the model in the form of attack threats by the system actors. As such, the environment is not modelled in external components, but also inside the system components. In the case of our e-health application for example, the system actors may present invalid credentials, or attempt to produce fraudulent prescriptions.

A simulation run on the kiltera behaviour and environment model generated for the "Visiting a pharmacist" use case, generates the following trace[2]:

```
<t=0.004 send patientCred       port=secureComm in Patient>
<t=0.004 receive patientCred    port=secureComm in Pharmacist>
<t=0.005 send ptValidatedCred   port=authority  in Pharmacist>
<t=0.007 send pharmacistCred    port=secureComm in Pharmacist>
<t=0.007 receive pharmacistCred port=secureComm in Patient>
<t=0.008 send phValidatedCred   port=authority  in Patient>
<t=0.009 send filledpres        port=secureComm in Patient>
<t=0.009 receive filledpres     port=secureComm in Pharmacist>
<t=0.011 send FillPres          port=secureComm in Pharmacist>
```

In this output, we notice a suspicious sequence of events, probably reflecting a breach of a security constraint. Indeed, it appears as though a previously processed prescription is being refilled. This could represent a flaw in the system design, and certainly indicates that the Statechart model should be reviewed.

[2]The trace syntax has been altered from its original and made more concise for clarity.

We can now perform the kiltera to CSP model transformation, and do full model checking *of all possible behaviours* using FDR2. Since both languages define similar constructs (though kiltera adds variable structure –not used here– and time), the translation between models is mostly concerned with the translation of time into a CSP clock process. Note that if we did not require efficient simulation capabilities, we could skip the intermediate kiltera step and directly translate Statecharts to CSP.

3.4 Verification with FDR2 of the Generated CSP Model

We perform verification of security constraints on a model translated from the behaviour and environment model described in Section 3.3. Following CSP terminology, this model is referred to as the *implementation* of the desired system.

Security requirements are directly captured in terms of restrictions on traces that the implementation process may engage in. These restrictions on traces are referred to as CSP *trace specifications*: either a description of a set of permissible traces, or alternatively, a set of unacceptable traces leading to an erroneous (security-breaching) state. Either way, this description is given in terms of a process (which we refer to as *security specification process*), which corresponds to a set of traces it can exhibit. We take this set to give precisely that secure behaviour which is required.

The specification process, along with the implementation model, are input into the FDR2 tool and its traces-refinement checker will verify whether the implementation trace-refines the security specification.

In CSP trace-refinement, an implementation process is said to refine the specification if all possible sequences of communication which it can perform are also possible for the specification process.

If the required security constraints are not effectively implemented in the original Statecharts model, in the case where the specification process described the permissible secure behaviour, FDR2 will reflect the failure to pass the CSP security trace specifications. In fact, the tool outputs a counter-trace, resulting from a sample execution that led to a security-violating state. In the case that the specification process described unacceptable behaviour, the breach of security is revealed if FDR2 finds that the implementation refines the erroneous behaviour description.

Therefore, in either case, the verification indicates a flaw in the original design, and points out the areas which need to be further tuned to achieve security. Design changes are directly made to the top-level Statecharts model, which is again transformed into a CSP model for further verification.

Applying this step to our example, the following is the CSP process equivalent to that described in Section 3.3:

```
PATIENT(PatientCredential, prescription) =
  secureComm!PatientCredential
  -> secureComm?PharmacistCredential ->
  if PharmacistCredential == pharmacistCred
  then pharmacistAccredited
     -> session!Prescription -> PATIENT
  else pharmacistNotAccredited
     -> contactAuthority -> PATIENT
```

The security constraints described in Section 2.2.1 are translated into permissible sequences of event occurrences, which describe a secure behaviour of the system. For the second security constraint, relating to the validity of the prescription, the following three specification processes are described:

```
VALID_PRES_FILLING = patientAccredited
  -> pharmacistAccredited -> prescriptionIsValid
  -> fillprescription -> commit
  -> VALID_PRES_FILLING
FORGED_PRES_FILLING = patientAccredited
  -> pharmacistAccredited -> prescriptionIsForged
  -> deanonymizePatient -> FORGED_PRES_FILLING
FILLED_PRES_FILLING = patientAccredited
  -> pharmacistAccredited -> prescriptionIsFilled
  -> deanonymizePatient -> FILLED_PRES_FILLING
```

The CSP implementation model generated above was refinement-checked against these specification processes, and was found to fail the *FILLED_PRES_FILLING* specification. In fact, the kiltera simulation in Section 3.3 had already warned us regarding this problem. The prescription-related security requirements were not fully addressed in the initial system design. This resulted in the unsuccessful check of the *FILLED_PRES_FILLING* specification, and the following counter event trace was produced by FDR2 to reflect the execution of a sequence of events leading to this outcome:

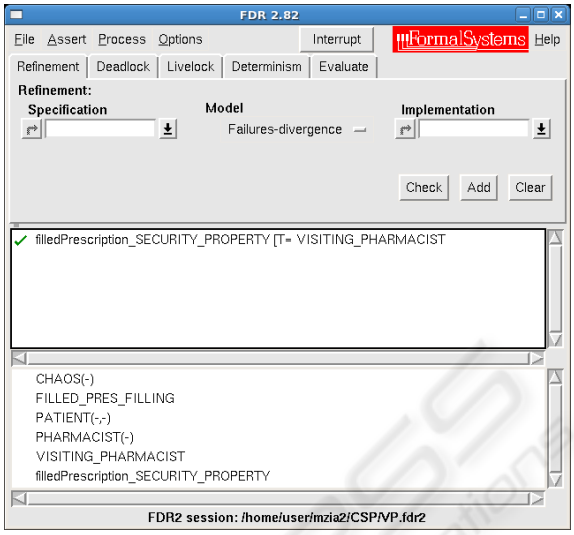

Figure 6: An FDR2. screenshot, reflecting the successful check of a security requirement. Here, *FILLED_PRES_FILLING* is the specification process, *filledPrescription_SECURITY_PROPERTY* is the security assertion, and the remaining are implementation processes.

```
pharmacistLocalAuthentication
patientLocalAuthentication
secureComm.patientCred
patientAccredited
secureComm.pharmacistCred
pharmacistAccredited
session.filledpres
prescriptionIsFilled
fillprescription
```

The counter-example shows that although a prescription was verified and found to have previously been filled, the system allowed a refilling of the prescription. A second implementation of this system is generated from a modified Statecharts design where the issue of filled prescriptions is dealt with by reporting the fraud. When FDR2 is run on this design specification, it outputs a successful assertion of the security requirement, as shown in Figure 6.

4 CONCLUSION

In most e-applications today, it is crucial to guarantee that security requirements are successfully implemented. Methods should be provided which can check if security has been attained in a system design. In this paper, we have presented a process based on multi-formalism modelling and model transformation

targeted towards verification of domain-specific security constraints, and have demonstrated it through the application on an e-health case study.

At the highest level, we started from a Sequence Diagram representation of software requirements, used these to obtain, through automatic transformation, an initial design of the system in Statecharts, transformed this model into a kiltera model, then extended and modified it to consider the system environment. Then, we further transformed the extended kiltera model into a CSP description and, finally, refined it with a CSP specification of the security requirements. The refinement checking tool FDR2 was used to verify that security was respected in the system implementation. A negative response by the checker indicated that the original Statecharts model neglected to address all security-related requirements. Note that after the above analysis (and possible modification of the design), the Statecharts model can be used for code synthesis of the final application.

We have shown how a multi-formalism approach can be useful for designing secure systems. The verification results gave us increased confidence that security requirements were effectively addressed. Productivity is increased as many of the steps in the process can be automated.

We are working on a toolchain which implements the modelling and transformation steps described above. Using the toolchain will allow us to investigate how well our approach scales to a full-blown application using the Belgian e-ID card. This work is done using our meta-modelling and model transformation tool AToM^3 (de Lara and Vangheluwe, 2002).

ACKNOWLEDGEMENTS

This work was made possible thanks to the support of the Flemish government (IWT-Vlaanderen) through the adapID project. Hans Vangheluwe greatfully acknowledges partial support for this work through his National Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant. We thank the anonymous reviewers for their pertinent and constructive comments.

REFERENCES

Bengtsson, J., Larsen, K. G., Larsson, F., Pettersson, P., and Yi, W. (1995). UPPAAL — a Tool Suite for Automatic Verification of Real–Time Systems. In *Proc. of Workshop on Verification and Control of Hybrid Systems III*, number 1066 in LNCS, pages 232–243. Springer.

de Lara, J. and Vangheluwe, H. (2002). AToM^3: A tool for multi-formalism and meta-modelling. In *FASE '02: Proceedings of the 5th International Conference on Fundamental Approaches to Software Engineering*, pages 174 – 188. Springer.

Feng, H. (2004). DCharts, A Formalism For Modeling and Simulation Based Design of Reactive Software Systems. Master's thesis, McGill University.

Formal Systems Europe Ltd. (2003). ProBE User Manual. Technical report.

Gardey, G., Lime, D., Magnin, M., and Roux, O. H. (2005). Romeo: A Tool for Analyzing Time Petri Nets. In Etessami, K. and Rajamani, S. K., editors, *Computer Aided Verification, 17th International Conference*, pages 418–423. Springer.

Germeau, F. and Leduc, G. (1997). Model-based Design and Verification of Security Protocols using LOTOS.

Harel, D. (1987). Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3):231–274.

Hoare, C. A. R. (1985). *Communicating Sequential Processes*. Prentice-Hall.

ISO (1989). LOTOS — a formal description technique based on the temporal ordering of observational behaviour. ISO IS 8807.

Lowe, G. (1996). Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055, pages 147–166. Springer.

Milner, R., Parrow, J., and Walker, D. (1989). A Calculus of Mobile Processes, Parts I and II. Reports ECS-LFCS-89-85 86, Computer Science Dept., University of Edinburgh.

Muller, P.-A., Fleurey, F., and Jézéquel, J.-M. (2005). Weaving Executability into Object-Oriented Meta-Languages. In Briand, L. and Williams, C., editors, *MODELS'05*, pages 264–278. Springer-verlag.

Mustafiz, S., Sun, X., Kienzle, J., and Vangheluwe, H. (2006). Model-Driven Assessment of Use Cases for Dependable Systems. In *MoDELS'06*, pages 558–573.

Naessens, V. (2006). *A Methodology for Anonymity Control in Electronic Services using Credentials*. PhD thesis, K.U.Leuven.

Posse, E. and Vangheluwe, H. (2007). kiltera: a simulation language for timed, dynamic-structure systems. In *Proceedings of the 40th Annual Simulation Symposium. SpringSim'07*, pages 293 – 300.

Reed, G. M. and Roscoe, A. W. (1986). A Timed Model for Communicating Sequential Processes. In Kott, L., editor, *ICALP*, volume 226 of *Lecture Notes in Computer Science*, pages 314–323. Springer.

Roscoe, A. W. and Wu, Z. (2006). Verifying Statemate Statecharts Using CSP and FDR. In Liu, Z. and He, J., editors, *ICFEM*, volume 4260 of *LNCS*, pages 324–341. Springer.

Rumbaugh, J., Jacobson, I., and Booch, G., editors (1999). *The Unified Modeling Language reference manual*. Addison-Wesley Longman Ltd., Essex, UK.

Sheyner, O., Haines, J., Jha, S., Lippmann, R., and Wing, J. M. (2002). Automated Generation and Analysis of Attack Graphs. In *SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy*, Washington, DC, USA. IEEE Computer Society.

Sun, X. (2007). A Model-Driven Approach to Scenario-Based Requirements Engineering. Master's thesis, McGill University.

Whittle, J. and Jayaraman, P. K. (2006). Generating Hierarchical State Machines from Use Case Charts. *Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06)*, 0:16–25.

Whittle, J. and Schumann, J. (2000). Generating statechart designs from scenarios. In *ICSE*, pages 314–323.

Yeung, W. L., Leung, K. R. P. H., Wang, J., and Dong, W. (2005). Improvements Towards Formalizing UML State Diagrams in CSP. In *APSEC*, pages 176–184. IEEE Computer Society.