On the Benefits of Using Aspect-Orientation in UPPAAL Timed Automata

Jüri Vain[*] , Dragos Truscan[#], Junaid Iqbal[#], Leonidas Tsiopoulos[*]

[#]*Software Engineering Laboratory, Abo Akademi University*

Vesilinnankatu 3, FI-20500 TURKU, FINLAND

firstname.surname@abo.fi

[*]*Department of Software Science, Tallinn University of Technology*
Akadeemia tee 15A, 12618 Tallinn, ESTONIA

firstname.surname@ttu.ee

***Abstract*— We present an evaluation study on applying aspect-oriented modeling concepts in UPPAAL timed automata. The study is focusing on the modeling and verification effort that can be reduced when applying explicit aspect-oriented structuring principles in model construction. We discuss the drawbacks and benefits related to model update and verification effort. The approach suggested is benchmarked on a mission critical crisis management system case study. We demonstrate the usability of our approach by extracting the aspects such as resource authentication and mission execution. Finally, we demonstrate by experimental data how our approach is more efficient compared to verification and testing effort applied in the non-aspect-oriented model.**

***Keywords*— model-based testing, aspect-oriented modeling, compositional verification, UPPAAL timed automata, mission execution**

I. INTRODUCTION

Model-based testing (MBT) [1] has become one of the black-box testing solutions for reducing software testing effort [2]. MBT suggests the use of abstract behavioral models for specifying the expected behavior of the System Under Test (SUT) and for automatically deriving tests from it. According to Utting et al. [3] there are three distinct phases in MBT: test specification, test generation, and test execution.

Compared to traditional testing methods, in MBT, the testing effort is shifted from mere test purpose specifications to modeling the requirements which the SUT should conform to. However, the recent survey by Binder [2] showed that two of the main challenges of MBT are in updating the models and in handling their complexity. In addition, the models used in MBT are not always intuitive and usually only part of the system behavior is modeled to reduce the test generation effort.

Model-checking is one of the methods used for test generation [3]. Given a formal model of the SUT and a set of properties the model should satisfy, a model-checker will generate witness or counterexample traces that confirm or refute the validity of these properties. The resulting traces are used for creating tests. UPPAAL Timed Automata (UPPAAL TA) are one of the popular formalisms supported by mature model-checking and testing tools [4]. Uppaal TA are well-suited for specifying timing properties of the SUT and therefore widely used in real time and embedded systems design. However, model-checking tools may suffer from scalability problems due to the so called state space explosion problem. Therefore, in this paper we aim at the reduction of the search space by applying model-checking to partial specifications in the form of aspect models.

Aspect-Oriented Software Development (AOSD) is a paradigm, originating from Aspect-Oriented Programming (AOP) [5]. It addresses the effects of crosscutting concerns on software artefacts: *scattering* – specifications related to one concern are distributed over several units, and *tangling* – a given unit contains specifications related to several concerns.

The core principle of AOSD is to develop multiple concerns of a system in isolation (via aspects) and later on combine (weave) them into a complete system. The perceived benefits of AOSD are: *separation of concerns* (which improves the developers' comprehension of large systems), *ease of maintenance, evolution and customization*, and thus, greater flexibility in development [6].

Aspect-Oriented Modeling (AOM) [7] combines the ideas behind AOSD with those of model-based software development, where the main focus is placed on how different concerns of the system can be modeled independently and combined later on via composition mechanisms. Experiments confirm that using AOM techniques provides models of better quality [8] and improved readability [9].

In this paper, we present an evaluation of our AOM approach for UPPAAL TA. The approach has been originally presented in [10], where we defined weaving mechanisms for UPPAAL TA models and associated tool support for aspect weaving. In the current paper we evaluate the approach and discuss its benefits and drawbacks with respect to modeling and model update effort, as well as with respect to the verification and testing effort. The evaluation is performed on a partial implementation of the Crisis Management System (CMS) case study suggested as reference case study in [11].

II. RELATED WORK

Several authors have evaluated the benefits of using AOM. The vast majority of these works target the Unified Modeling

Language (UML) [12]. UML has been the *de facto* modeling language in AOM. For instance, authors of [13] evaluate several aspect modeling notations in order to evaluate them with respect to reusability and maintainability of the obtained models. They concluded that having Reusable Aspect Models performs well against the majority of the employed metrics.

In [8], the authors report a controlled experiment to assess the applicability of AOM from the perspective of the AspectSM UML profile. The study focuses on two aspects: the quality of the derived state machines and the effort required to build them. The study concludes that, even if it took more time to develop, the aspect state machines derived with AspectSM are more complete and correct compared to standard approach.

A similar study [9] evaluates the reliability of state machine developed with AspectSM. The results show that AOM enables better defect identification and fixing rates in both hierarchical and flat state machines.

Another study [14], points out that using AOM in an industrial context enables scalable modeling, reduced modeling complexity and facilitates model evolution.

In contrast to the works referred above, we perform an evaluation on applying AOM principles to modeling, verification, and testing using UPPAAL TA. We are not aware of similar studies in this context.

III. PRELIMINARIES

A. UPPAAL Timed Automata

UPPAAL Timed Automata (UPPAAL TA) [15] used for the specification are defined as a closed network of extended timed automata that are called *processes*. The processes are combined into a single system by synchronous parallel composition like that in process algebra CCS. The nodes of the automata graph are called *locations* and directed vertices between locations are called *edges*. The *state* of an automaton consists of its current location and assignments to all variables, including clocks. Synchronous communication between processes is expressed by synchronisation links called *channels*. A channel *ch* relates a pair of transitions in parallel processes where synchronised edges are labelled with symbols for input and output actions (denoted *ch*? and *ch*!, respectively).

Asynchronous communication between processes is modeled using *global variables* accessible in all processes. Let Σ denote a finite alphabet of actions $a, b, \ldots$, and C a finite set of real-valued variables x, y, z denoting clocks. A *guard condition* of the edge is a conjunctive formula of atomic constraints of the form $x \sim n$ for $c \in C$, $\sim \in \{\leq, \geq, =, >, <\}$ and $n \in N^+$. We use $G(C)$ to denote the set of clock guards.

A timed automaton A is a tuple (L, l_0, E, Inv) where L is a finite set of locations, $l_0 \in L$ is the initial location, $E \in L \times G(C) \times \Sigma \times 2^C \times L$ is the set of edges, and $Inv: L \rightarrow I(C)$ assigns *invariants* to locations (here we restrict to constraints in the form: $x \leq n$ or $x < n$, $n \in N^+$). Without the loss of generality we assume that guard conditions are in conjunctive form with conjuncts including besides clock constraints also constraints on integer variables. Similarly to clock conditions, the propositions on integer variables k are of the form $k \sim n$ for $n \in Z$, and $\sim \in \{\leq, \geq, =, >, <\}$. For formal definition of complete semantics of UPPAAL TA we refer to [16].

B. Aspect-Oriented Modeling with UPPAAL TA

The approach evaluated in this paper has been originally presented in [10] where we defined aspect-oriented concepts for UPPAAL TA and proposed weaving adapters for weaving. We will briefly summarize this work in the following.

1) Mapping aspect-oriented concepts to UPPAAL TA

According to aspect-oriented principles, different crosscutting concerns are modeled independently starting from requirements, either as a base or advice models. They are then woven into a composite model. Before we elaborate on AO verification, we interpret the AO terminology in the context of UPPAAL TA, as follows. A *base model* is a set of UPPAAL TA processes modeling the base (primary) functionality of the system. An *aspect model* is an UPPAAL TA process or a set of parallel processes implementing a crosscutting concern. *Joinpoints* are model fragments in the base model to which an aspect can be woven. For this paper, we limit our join points to model fragments composed of an edge with synchronization. However the approach can be generalized to larger model fragments. A *pointcut* is the set of joinpoints and conditions under which an advice can be woven. Pointcut expression is a logic condition which uniquely defines the model fragments (join points) where the weaving is applied. A *woven model*, sometimes referred in the literature as *augmented model*, is an UPPAAL TA in which the base model is woven with all intended aspect models. *Weaving* is the process of composing a base model with the aspects. *Weaving adapters* are model fragments that allow the execution of an advice model at the designated joinpoints. A weaving adapter encodes the pointcut expression, the advice type and the joinpoint. Our approach is based on several assumptions:

- We do not allow unique instances of the same advice model (defined by an UPPAAL TA template) to be shared by several join points of a base model.

- The execution of an advice is atomic w.r.t. its joinpoint. This means that once a joinpoint is reached, the control flow of the base model process containing the joinpoint will be passed to the aspect model and the base model process will wait for the aspect to complete and return to the same joinpoint. However, this does not restrict several joinpoints located in different processes of the base model to be enabled at the same time and their corresponding aspects to be executed simultaneously.
- An advice model has one entry point and one or several exit points which return to the same joinpoint.
- The base model and advice model can be woven using UPPAAL TA specific communication and synchronization constructs, e.g. synchronizing the entry and exit of the advice model with *wait* in the base model, sharing or refining data between base and advice model, etc.
- Joinpoint definitions cannot target the elements introduced by the weaving adapters in order to ensure that the weaving does not introduce or remove joinpoints for another adapter. However new joinpoints can be introduced via the advice models.

2) Weaving Adapters

In [10], we defined four types of weaving adapters. They provide support for weaving an advice *before*, *after*, and *around* a join point, similarly to the homonym advice types in AspectJ. The forth adapter type, *conditional*, has been suggested based on practical considerations. Due to space limit, in this paper we will only discuss the adapters used in our example, while we defer the others to [10].

In the current approach, we restrict a joinpoint to a model fragment composed of an UPPAAL TA edge labelled with a *guard expression*, *channel*, and *update* as depicted in Fig. 1. The channel represents a synchronization. It can be interpreted either as send or receive action denoted respectively by *channel*! and *channel*?. Optionally, the edge may carry a guard expression and an update expression. However, our approach can be easily extended to more complex model fragments, as long as weaving assumptions are fulfilled.

The main purpose of the weaving adapters is to allow a systematic and mechanized weaving of advice models at designated joinpoints in the base model. A weaving adapter has a base model side and an advice model side, specifying the model fragment to be included in the base model and, respectively, to the advice model, during weaving.

The *after adapter* (Fig.2 top) allows the execution of an advice after a channel synchronization. It refines the *End* location with two new locations *AspectStart* and *Call* , as well

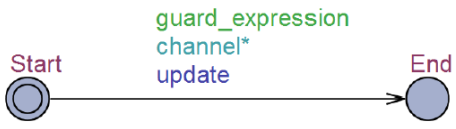

Fig. 1 Model fragment with channel synchronization

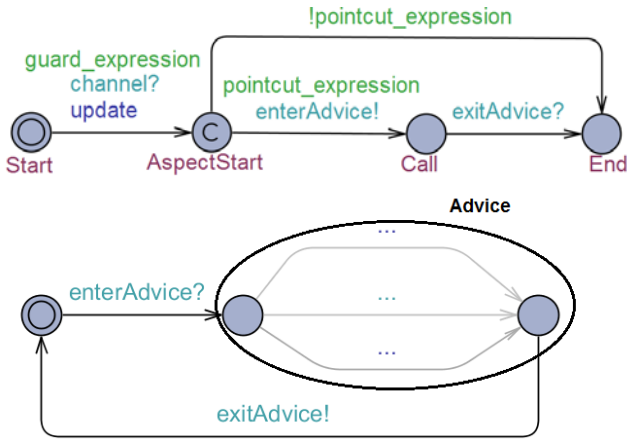

Fig. 2 Generic *after adapter* (top) and generic advice (bottom).

as with two new channels *enterAdvice!* and *exitAdvice?*. Whenever the *pointcut_expression* is true, the advice is executed, otherwise the advice is skipped.

The corresponding adapter introduced to the advice model during the weaving is shown in Fig. 4 (bottom). As one may notice, the execution of the advice model is triggered from the base model via the joinpoint by receiving the *enterAdvice?* synchronization and, after executing the advice functionality, it returns the control via the *exitAdvice!* synchronization [11].

3) Weaving Process:

In our approach, we assume that aspects can be designed independently from specifications and the aspects are woven incrementally. That is, for a given base model and a set of advices, we weave one advice at a time to all of its designated join points. We regard the weaving process as a model transformation that takes as input a base model, an advice model, and a selected weaving adapter.

The pointcut expression is used as model pattern which identifies joinpoints. The transformation inserts the adapter at the joinpoint and instantiates the UPPAAL template of the advice for each joinpoint.

We also assume that the weavings are applied to the class of *weakly-invasive* aspects and that the weaving is conservative with respect to this class. Weakly-invasive aspects may change the control flow and the values of non-local variables, as long as the state after returning the execution to the base model is reachable in the base model without the aspect woven [17]. Additionally, verification of aspect non-interference is another prerequisite for allowing us to take advantage of compositional verification and testing of the aspect-oriented models. That is, inferring the properties and test verdicts of the composition from verified properties or passed tests of components in separation. We presented guidelines for enabling compositional verification and testing of aspect-oriented UPPAAL models in [18].

IV. Case Study – Crisis Management System

We take as an example a specification of the Crisis Management System (CMS) proposed in [11] as reference case study. The idea behind CMS is to deal with different kinds of crisis situations, ranging from major to catastrophic accidents, by allocating resources to handle the crisis. The actors in the CMS are the *Coordinator* and the *Resources*. *Coordinator* receives calls from witnesses who are reporting an incident and initiates a new mission. Each mission has a type which tells the number of resources that have to be allocated. Three procedures of the CMS will be modeled, as follows:

ResourceAllocation: A mission needs to be completed within a specified mission time, *MT*. *Coordinator* initiates the mission by sending a message to *cSystem*. The latter is responsible for allocating resources and waiting for them to complete their task. The resources are allocated after performing a feasibility check by comparing if the resource's *time-to-arrive* (TTA) and *mission execution time* (MET) match. The feasibility check is performed twice, i.e., before sending the initial synchronization for allocation and later when the resource is allocated to the mission via a second synchronization. When all the allocated resources have completed their tasks within MT, *cSystem* informs *Coordinator* about the mission completion. However, if a resource cannot be allocated, *cSystem* will attempt to allocate the next available resource. Furthermore, if *cSystem* cannot allocate the required number of resources within MT, the mission will fail. Likewise, if all the allocated resources are unable to complete their task within MT, the mission will timeout and *Coordinator* is notified.

Authentication: Every *Resource* needs to authenticate itself before it can be allocated to a mission. The authentication process follows a simplified version of Needham-Schroeder protocol adopted from [19]. There are two participants in the protocol: *A* - the initiator, and *B* - the responder.

In the first step, the initiator *A* sends a message containing its identity to the responder *B* and after a sequence of encryption/decryption procedures with the information using public and private keys, the initiator is eventually authenticated. In case of an authentication failure a resource cannot be used for the current mission. The authentication process for every resource requires a fixed amount of time, including the time needed to generate the secret key of *B*, and to encrypt and decrypt the messages sent between *A* and *B*.

Execute Critical Mission (ECM): The execution of each resource begins when its allocation is completed. It starts to *travel* to the mission site and it must arrive after TTA time units. After reaching the location, each resource requires MET time units to complete its task. Upon successful execution, it notifies *cSystem*.

A. Modeling the CMS

For evaluation purposes, we developed two versions of the CMS specification, referred in the following as *flat* and, *aspect-oriented* (AO) model, respectively. The former is developed using traditional, non-aspect-oriented methods, while the latter is developed using our aspect-oriented approach. To ensure that the two versions are specifying the same observable behavior, we use bisimulation relation as detailed later on.

1) Traditional approach: Fig. 3 depicts the flat version of CMS specified using the traditional modeling approach, containing a *Coordinator*, an *AuthResponder* and a resource (*flatResource0*) process (only *Coordinator*, one *Resource* and corresponding *AuthResponder* are shown in order to save space, while *cSystem* is shown in Fig. 4). *cSystem* initially allocates the number of required resources. The feasibility check is performed before sending an alloc synchronization and later when the resource is allocated to the mission.

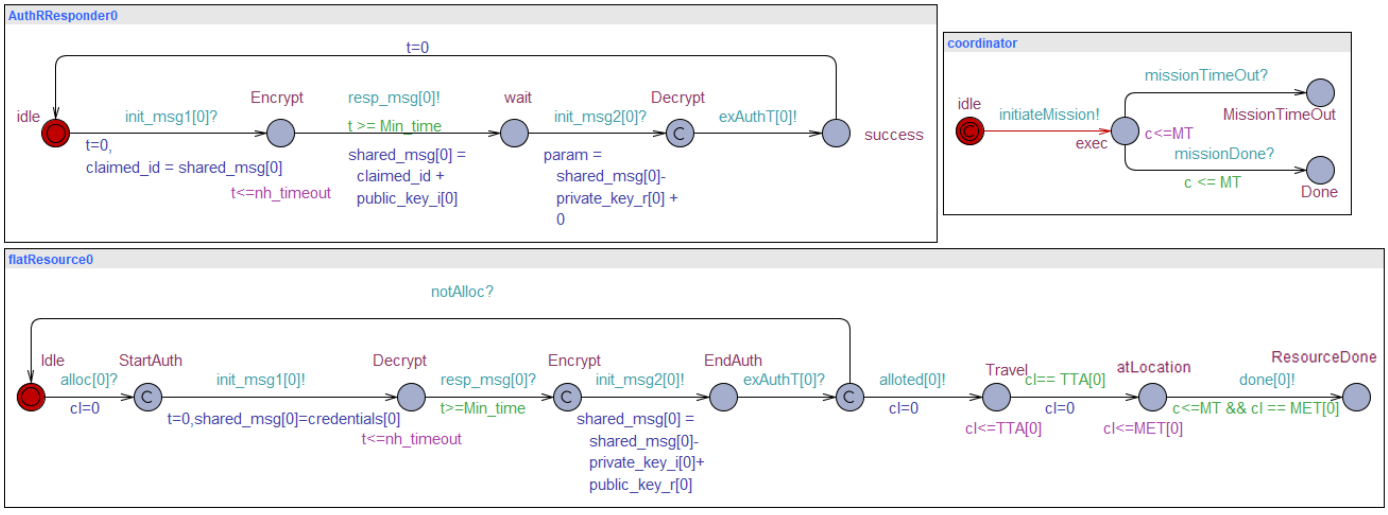

Fig. 3 Flat model with one resource including the Authentication and ECM features

After authentication, resource proceeds to *Travel* and spends *TTA* time units to reach the crisis location (*atLocation*). The mission execution progresses while staying at *atLocation* and after *MET* time units, the resource notifies *cSystem* about the successful completion of its task via sending a *done!* synchronization and reaching *ResourceDone* location.

Moreover, the successfully deployed resources, i.e., the ones that reached the location and have performed their task within *MT* time units, contribute to the overall success. The mission is completed when all required resources have successfully reached the *ResourceDone* location in the model.

2) Aspect-oriented approach: For the aspect-oriented version of the CMS, the starting point is modeling the resource allocation procedure which will be considered as the base model (see Fig. 4). Each resource is modeled as one process, but in order to save space we only show one instance of it. The additional Authentication and ECM features are modeled as separate aspects, which are incrementally woven into the base model in Fig. 5 (the *cSystem* automaton is omitted since it is identical to the one in Fig 4).

The ECM aspect refines the time behavior of how a resource executes a mission as well as the functional steps involved. Beside the number of required resources, a mission requires that the resource arrives at the location and executes its task within MT. Thus, we characterize each resource availability by its distance to the incident site and time for task execution, i.e., bounded by the TTA and the MET clock constants. This exemplifies one possible option of weaving aspect models, as a refinement of an abstract model. The ECM aspect is woven in the base model at the *alloted[i]!* join point using an *after* adapter which allows the control flow to return to a location before the join point. The execution of each resource specified by the aspect advice begins when receiving a synchronization on channel *eExec[i]?* from the base model and proceeds to mission site (location *Travel*) which consumes TTA time units. After reaching the mission site (location *atLocation*), each resource requires MET time units to complete its task. After successful execution, the control is returned back to the base model of a resources via the *exExecT[i]!* synchronization.

The second aspect to be modeled is the *Authentication* (*Auth*). Whenever a resource is initialized (requested to be allocated), it attempts to authenticate itself via *eAuth[i]* by providing its credentials. The authentication aspect is modeled as two parallel automata, one for the initiator and one for the responder. The initiator automaton, modeled as *AuthI0*, receives the credentials of a given resource from the *Resource0* process (Fig. 5), while the responder process (*AuthR0*) will provide the verdict of authentication back to the base model via the channel of the adapter *exAuthT!* for successful authentication.

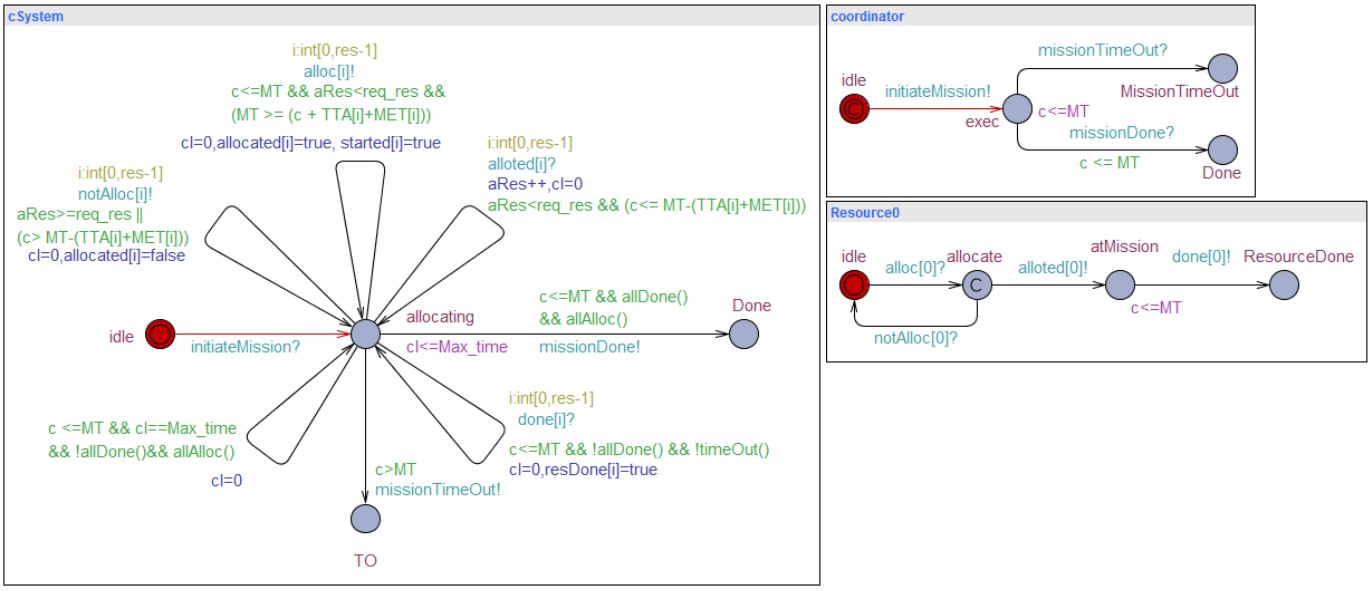

Fig. 4 The Base model of CMS

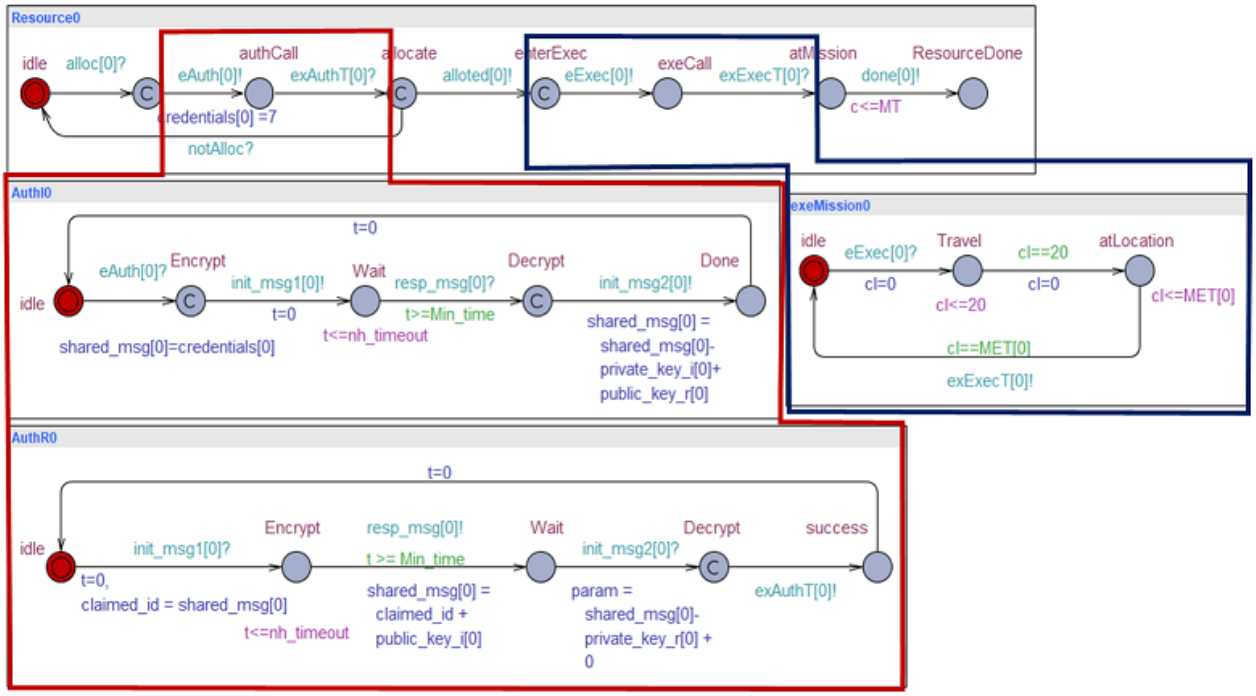

Fig. 5 Base model woven with Authentication (surrounded by bold line in the left) and ECM aspects (surrounded by bold line in the right)

V. Evaluation of the Method

In our evaluation, we aim at answering two questions regarding the proposed approach:

1. Are *aspect-oriented UPPAAL TA models easier to update compared to models built using the traditional method?*
2. Is *the compositional verification and test generation effort of UPPAAL TA aspect models smaller than the verification and testing effort of the traditional model.*

A. Modeling Update Effort Evaluation

In order to evaluate the modeling and update effort, we conducted an experiment with the two different developed versions of the authentication feature. The first version used a simpler specification, in which the resource provided its credentials (authentication token) and the CMS checked if they correspond to a list of known credentials. The CMS specification was developed both as flat and aspect models and the two models were checked for equivalence via bisimulation. In the second version, we updated the authentication feature to use the Needham-Schroeder protocol which resulted in the models described in this paper.

During the update we measured the number of changes (statement additions, updates, removals) that we had to perform on the flat model. It resulted that modifying the flat models required editing approximately 40% of the model, versus 20% in the case of the aspect model. The effort of updating the models was much smaller in the case of aspect models, since it was easier to identify which elements had to be changed, and these elements had in general a local scope, without affecting the specification of the other features. This was an expected result, according to different studies, which confirm that AOM reduces the scattering and tangling of requirements.

B. Verifying equivalence between flat and aspect models

As referred above, AOM imposes explicit structure to the models which in their flat form do not provide clear separation of design concerns and do not support compositional verification and testing. When transforming flat models to aspect-oriented ones or comparing their complexity the same behavior has to be maintained by both. To assure that aspect-oriented models present the same observable behavior as the non-aspect models, a relevant notion of equivalence between models is needed. For that, we employ bisimulation relation between models. Informally, two UPPAAL TA are bisimilar if they accept the same timed language, i.e., they perform exactly the same observable (i/o-) actions and reach bisimilar states while satisfying same time constraints. In other words, these two system models cannot be distinguished by an external observer. Bisimilarity is a symmetrical relation.

Bisimulation for timed automata has been originally introduced by [16] and, as shown in [20], it is decidable for parallel timed processes.

In order to show bisimilarity between the flat and aspect-oriented models of the same system, we follow the steps below:

1. we compose the flat model and its aspect-oriented counterpart in parallel;

2. we define an observable interface (set of i/o actions) between the system and its environment models, with respect to which the equivalence has to be shown;

3. we add additional, side-effect free, synchronization channels between the edges (of compared models) that model same observable i/o-actions defined in step 2;

4. Finally, we verify that the synchronous composition of compared models never deadlocks in states where the models verified separately would not deadlock.

Before benchmarking the case study with respect to verification effort, we proved that different versions of the CMS are bisimilar.

C. Evaluation of the Verification Effort

The verification and testing in UPPAAL are in the same complexity class since verification traces are later used as test sequences. In order to compare the verification and testing effort and exemplify the difference between the traditional and aspect-oriented approach, we execute the same verification queries and benchmark them against the three models: *flat* model, *Base* ⊕ *Auth*, and *Base* ⊕ *ECM*, where ⊕ denotes the *weaving* composition operator.

In our evaluation, we rely on two characteristics of our models: a) compositionality - since the individual aspect models are weakly-invasive by construction, a verification property that holds on two intermediate aspect models i.e., *Base* ⊕ *Auth* and *Base* ⊕ *ECM* will also hold on the complete aspect model and b) bisimilarity - the complete (augmented) aspect model is bisimilar to the flat model. Thus, any local property of an aspect that will pass the verification on the individual aspect models, is expected to also pass the verification on that flat model.

Since the *Base* ⊕ *Auth* ⊕ *ECM* model and the corresponding version of the *flat* model have been proven bisimilar and each intermediate weaving, i.e., *Base* ⊕ *Auth* and *Base* ⊕ *ECM*, preserve their local properties if they are satisfied in the full aspect model. Thus, any local property of an aspect that will pass the verification on the flat model, is expected to also pass the verification of that aspect model in isolation.

For instance, we use the following property "*The system should not be in a deadlock state except when the mission is completed or there was a mission timeout.*" which corresponds to UPPAAL query:

A[] deadlock imply (cSystem:Done || cSystem:TO)

For each model we recorded the number of stored symbolic states, the number of explored symbolic states and the CPU time. We repeated each benchmark 10 times and calculated the average values shown in TABLE 1.

The results show that the verification effort for the above query on *flat* model was 862 stored and 2991 processed symbolic states, respectively. The benefit of compositionality becomes evident when verifying the aspect models independently. The verification search space resulted in 1380 stored symbolic states for the *Base* ⊕ *Auth* model, and in 230 stored symbolic states for the *Base* ⊕ *ECM* model. Similarly, the number of steps required to verify the query was 2235 and 230 explored symbolic states, respectively.

TABLE 1
EXPERIMENTAL RESULTS

Model	States Explored	CPU time (ms)	States Stored
Flat	2991	108	862
Base ⊕ Auth	2235	52	1380
Base ⊕ Exec	230	5	230

The effort of verifying the query on the individual aspect model is, as expected, much smaller than verifying the query on the *flat* model. In this particular case, the sum of the effort of the two verification tasks is also smaller than verifying the same query on the *flat* model, both time and space wise. However, the *flat* model has less structural elements as compared to *aspects* model, the verification is achievable with fewer verification steps. One may notice that the explored states for verification of both *flat* and sum of explored states of the *aspect* models is almost the same.

VI. Conclusions

In this work, aspect-oriented modeling concepts have been applied to UPPAAL timed automata and studied from the perspective of improving the modularity and reducing complexity of model-based verification and testing.

By developing simultaneously non-aspect oriented and aspect-oriented models we found that besides improving human comprehension and traceability of requirements the modularity of aspect-oriented models allows one to reduce the verification effort.

The experiments built upon the Crisis Management System case study revealed also some limitations stemming from the weaving adapters. Even though the weaving adapters are not introducing additional interleaving in the parallel composition, they introduce additional model elements which increase the structural complexity of models and indirectly affect the state space. Though, the larger the structural complexity of the advice models, the more effect the compositional testing and verification have, since the complexity of the advice model compensates for the structural complexity introduced by the adapters.

The efficiency of aspect-oriented verification and testing, depends on whether these activities can be done compositionally, i.e., if it is possible to infer from verified properties or passed tests of components in separation the properties and test verdicts of the augmented model as a whole. The experiments also confirmed that separate aspect

models can support compositionality if there is no interference between them.

As future work, we plan to define and evaluate the aspect-oriented coverage criteria for model-based testing, which will allow better addressing aspects related properties of systems. We also plan to conduct a set of larger case studies to evaluate the scalability of the approach as well as its advantages from the point of view of incremental test suite updates.

ACKNOWLEDGMENTS

This research is partially supported by Estonian Science Foundation Project IUT33-13 "Strong warranties software methodologies, tools and processes" and by ECSEL JU MegaM@Rt2 project under grant agreement No 737494.

REFERENCES

[1] M. Utting and B. Legeard, *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann, 2010.

[2] R. V. Binder, "2011 Model-based Testing User Survey: Results and Analysis." Jan-2012.

[3] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing approaches," *Softw. Test. Verif. Reliab.*, vol. 22, no. 5, pp. 297–312, 2012.

[4] G. Behrmann, A. David, and K. Larsen, "A Tutorial on Uppaal," in *Formal Methods for the Design of Real-Time Systems*, vol. 3185, M. Bernardo and F. Corradini, Eds. Springer Berlin Heidelberg, 2004, pp. 200–236.

[5] Kiczales and Others, "Aspect-Oriented Programming," in *ECOOP '97 - Object-Oriented Programming*, 1997, vol. 1241, pp. 140–149.

[6] M. Badri, L. Badri, and M. Bourque-Fortin, "Generating unit test sequences for aspect-oriented programs: towards a formal approach using UML state diagrams," in *Information and Communications Technology, 2005. Enabling Technologies for the New Knowledge Society: ITI 3rd International Conference on*, 2005, pp. 237–253.

[7] S. Clarke and E. Baniassad, *Aspect-Oriented Analysis and Design. The Theme Approach*. Addison-Wesley, 2005.

[8] S. Ali, T. Yue, and L. Briand, "Assessing Quality and Effort of Applying Aspect State Machines for Robustness Testing: A Controlled Experiment," in *Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on*, 2013, pp. 212–221.

[9] Shaukat Ali, Tao Yue, Lionel C. Briand, "Does Aspect-Oriented Modeling Help Improve the Readability of UML State Machines?," *Software & Systems Modeling*, vol. 13, no. 3, pp. 1189–1221, 2014.

[10] D. Truscan and Others, "A Tool-supported Approach for Introducing Aspects in UPPAAL Timed Automata," in *Software Technologies - The 9th International Conference, ICSOFT 2014, Vienna, Austria, 2014, Revised Selected Papers*, 2014.

[11] N. G. Jörg Kienzle and S. Mustafiz, "Crisis Management Systems: A Case Study for Aspect-Oriented Modeling," *Transactions on Aspect-Oriented Software Development*, vol. 7, pp. 1–22, 2010.

[12] Omg, "Unified Modeling Language Infrastructure Specification, version 2.1.2." Oct-2007.

[13] A. Mehmood and D. N. A. Jawawi, "A quantitative assessment of aspect design notations with respect to reusability and maintainability of models," in *Software Engineering Conference (MySEC), 2014 8th Malaysian*, 2014, pp. 136–141.

[14] S. Ali, L. C. Briand, A. Arcuri, and S. Walawege, "An Industrial Application of Robustness Testing Using Aspect-Oriented Modeling, UML/MARTE, and Search Algorithms," in *Lecture Notes in Computer Science*, 2011, pp. 108–122.

[15] A. Hessel and Others, "Testing Real-Time Systems Using UPPAAL," in *Formal Methods and Testing*, Springer-Verlag, 2008, pp. 77–117.

[16] J. Bengtsson and W. Yi, "Timed Automata: Semantics, Algorithms and Tools," in *Lectures on Concurrency and Petri Nets*, vol. 3098, J. Desel, W. Reisig, and G. Rozenberg, Eds. Springer Berlin Heidelberg, 2004, pp. 87–124.

[17] E. Katz and S. Katz, "Incremental Analysis of Interference Among Aspects," in *Proceedings of the 7th Workshop on Foundations of Aspect-oriented Languages*, Brussels, Belgium, 2008, pp. 29–38.

[18] J. Iqbal, L. Tsiopoulos, D. Truscan, J. Vain, and I. Porres, "The Crisis Management System – A Case Study in Aspect-Oriented Modeling Using UPPAAL," Turku Centre for Computer Science, 2016.

[19] M. Rong, Z. Li, and G. Zhang, "Model Checking of Needham-Schroeder Protocol Using UPPAAL," in *2010 6th International Conference on Wireless Communications Networking and Mobile Computing (WiCOM)*, 2010, pp. 1–4.

[20] K. Čerāns, "Decidability of bisimulation equivalences for parallel timer processes," in *Computer Aided Verification: Fourth International Workshop, CAV '92 Montreal, Canada, June 29 -- July 1, 1992 Proceedings*, G. von Bochmann and D. K. Probst, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 302–315.