

Enabling a Roundtrip Engineering Process for the Modeling and Analysis of Embedded Systems*

Heather Goldsby, Betty Cheng,** Sascha Konrad, and Stephane Kamdoun
{hjpg,chengb,konradsa,kamdoumm}@cse.msu.edu

Software Engineering and Network Systems Laboratory
Department of Computer Science and Engineering
Michigan State University
3115 Engineering Building
East Lansing, Michigan 48824 USA

Abstract. Increasingly, object-oriented technology, specifically the Unified Modeling Language (UML), is being used to develop critical embedded systems. There have been several efforts to translate UML models into formal specification languages, thus enabling the models to be analyzed by model checkers. Unfortunately, the complexity and volume of the analysis results often prevents developers from fully taking advantage of the analysis capabilities. This paper introduces a generic visualization framework, Theseus, that provides developers with a model-based, visual interpretation of the analysis results in terms of the original UML diagrams. Within this framework, a playback mechanism displays the execution path that has led to a model checking violation in terms of the original UML state diagram and a newly generated sequence diagram that depicts the problem scenario. A Theseus prototype supporting the Spin and SMV model checkers has been applied to the analysis of UML models for embedded systems from industry.

1 Introduction

Embedded systems have become increasingly pervasive, particularly occurring in high-assurance systems, such as automotive systems, medical devices, and telecommunication systems. Given the critical nature of these embedded systems applications, it is important to use rigorous development techniques. Increasingly, object-oriented technology is being used to develop embedded systems [1, 2]. Furthermore, the Unified Modeling Language (UML) [3], the *de*

* This work has been supported in part by NSF grants EIA-0000433, EIA-0130724, CDA-9700732, CCR-9901017, Department of the Navy, Office of Naval Research under Grant No. N00014-01-1-0744, Eaton Corporation, Siemens Corporate Research, and a grant from Michigan State University's Quality Fund. Stephane Kamdoun completed a portion of this work while studying under an International Exchange Program through the University of Kaiserslautern.

** Please contact this author for all correspondences.

facto standard for object-oriented modeling, is the primary modeling notation for the recent movement towards model-driven development (MDD), such as in the model-driven architecture (MDA) by the OMG [4]. Using MDD, the models are refined successively from requirements to design and eventually to code. One drawback with the UML has been the lack of model analysis tools. To date, most of the UML analysis has been limited to syntactic-based analysis or simulation. Recently, there have been several efforts to translate UML behavior diagrams (e.g., state and sequence diagrams) to formal specification languages [5–7] to be analyzed by model checkers, such as Spin [8] and SMV [9]. A challenge with this approach to analysis is how to understand and then use the error descriptions from the analysis output to modify the original UML diagrams. This paper describes a generic visualization framework, Theseus, that interprets the analysis output from model checkers in terms of the original UML diagrams. Using Theseus, the developer is alleviated from the burden of deciphering the frequently cryptic and verbose trace output, which is often denoted in an analysis tool-specific language, including references to line numbers of the specification, internal process numbers, temporary variable names, etc.

In addition to the syntactic-based analysis tools, such as those provided with XDE [10], several CASE tools [11–15] provide visualization support for (UML) model simulation. Simulation provides information about a single execution path (e.g., a scenario) through a system model, where visualizations can be used to depict a scenario by displaying message traces in sequence diagrams or highlighting elements of a state diagram. Simulation-based analysis *validates* that a model conforms to a developer’s expectations. In contrast, the recent work of translating the UML diagrams to model checker specification languages is intended to support the *verification* of UML models. That is, does a UML model satisfy temporal properties, such as invariants and leads-to properties, for all possible execution paths. Particularly for high-assurance systems, it is important to be able to verify a UML model against critical properties *before* the models are refined to design and code. A notable feature of model checkers is that if a system model does violate a property, a counterexample depicting the sequence of events and/or states causing the violation is returned. Two challenges exist with using the analysis results. First, a developer needs to decipher the verbose and often non-intuitive representation of system elements specified in the counterexample. Second, the cause of the error must be mapped back to the original UML diagram in order to make the appropriate model refinements, particularly in the context of MDD.

This paper describes a generic visualization framework, Theseus, that supports a model-driven, visual interpretation of analysis output from commonly used model checkers, including Spin and SMV. Three tasks were essential in the development of Theseus. First, based on numerous trace output files generated from each model checker, we constructed a grammar and a corresponding parser for each model checker to be supported by Theseus; the parser generates an abstract syntax graph (ASG) for a given trace file. Second, we developed a translator for each formal analysis tool that traverses the ASG to generate a

generic XML representation containing only UML-relevant model elements, such as state names, transition names, attributes, etc. The parser and translator are combined into an analysis tool-specific trace processor. Third, we developed a visualization engine that processes the XML representation of the counterexamples to support UML state diagram animation and sequence diagram generation. The combination of these three elements have been encompassed in the Theseus prototype that accepts as input a UML model and the trace file for a counterexample generated from a model checker (currently, we support Spin and SMV) for an error detected in the UML model, and produces a state diagram animation and sequence diagram depicting the counterexample. The user has the option of either stepping (single or multi-step) through the animation or running through the complete counterexample, where color changes are used to depict state and transition traversals.

Theseus has been developed to provide a critical piece of a larger project supporting a roundtrip-engineering approach to the construction of UML diagrams for modeling and analyzing embedded systems requirements. Specifically, we have previously developed several techniques and tools to provide a bridge between (semi-)informal and formal approaches to requirements engineering of embedded systems. First, in order to enable UML diagrams to be automatically analyzed by model checkers, we developed a meta-model based approach to mapping UML diagrams to target specification languages [5]. Hydra is a prototype tool that supports the automatic generation of specification languages, such as Promela, the specification language of the Spin model checker [8], from UML class and state diagrams. Second, in order to help developers create the UML diagrams, we developed a set of object analysis patterns for embedded systems [16], that provide sample structural and behavioral templates for modeling embedded systems. Third, in order to facilitate the specification of formally analyzable properties using natural language, we have developed a structured natural language grammar and the SPIDER tool [17, 18]. Using SPIDER, developers can create natural language specifications of properties that are automatically and transparently mapped to the property specification language of the targeted analysis tools, *e.g.*, linear-time temporal logic (LTL) [19, 20] for the model checker Spin [8]. Theseus provides the fourth component of the roundtrip-engineering process, that is, the visualization of the model checking analysis. Therefore, putting all four elements together, a developer can use the object analysis patterns to create a UML model for an embedded system, use Hydra to generate a formally analyzable model for a model checker, use SPIDER to specify properties to be satisfied by the UML model, use the model checker to analyze the UML model against the SPIDER-specified properties, and use Theseus to visualize counterexamples generated from the model checker in terms of the original UML diagrams, thus completing the roundtrip-engineering process.

In order to validate our work, Theseus has been instantiated to handle trace output generated from two different model checkers, Spin and SMV, and we have applied our roundtrip-engineering process to the analysis of several industrial embedded systems. The remainder of the paper is organized as follows. Section 2

guide developers in the construction of design models, object analysis patterns guide developers in the creation of conceptual models during the analysis phase preceding the design phase. Specifically, these patterns aid in the construction of conceptual models of the embedded systems focusing on functional aspects, where these models may later be refined in the design phase through the use of design patterns.

In addition to creating the model, the user can specify the properties of the UML model to be analyzed. In our approach, these properties are specified in natural language using a previously developed process for deriving and instantiating formally analyzable natural language properties based on real-time and qualitative specification patterns [17, 18], termed SPIDER (**S**pecification **P**attern **I**nstantiation and **D**erivation **E**nvi**R**onment). Briefly, the SPIDER process comprises three steps:

1. **Derivation:** Derive a natural language sentence from a structured natural language grammar.
2. **Instantiation:** Instantiate the natural language representation with model-specific elements.
3. **Mapping:** Map the instantiated natural language sentence to the temporal logic required by the targeted formal validation and verification tool and analyze.

An important component of this process is a structured natural language grammar. This grammar is used to derive natural language sentences that can be mapped to formal specifications structured in terms of a specification pattern system. In this paper, we use the qualitative portion of a previously developed structured English grammar [23] for the specification patterns by Dwyer *et al.* [24].

Using SPIDER, the developer specifies the property to be verified in natural language. SPIDER then translates the natural language property to a form that can be understood by the targeted analysis tool. Spin uses linear-time temporal logic (LTL) [19, 20] as its property specification language, while SMV requires the property to be specified in computational tree logic (CTL) [25]. Accordingly, SPIDER translates the natural language property to LTL to be used with Spin and to CTL to be used with SMV.

2.2 Step 2: Formalizing a UML Model

After using the object analysis patterns to create the UML model, the UML model is translated into the specification language for the targeted model checker. We briefly describe the process for formalizing a UML model in the specification languages accepted by Spin and SMV.

To translate a UML model to Promela for Spin analysis, we use a previously developed UML formalization framework termed Hydra [5]. The general UML-to-Promela formalization approach is to map objects to processes in Spin (*proctypes*) that exchange messages via *channels*. Nested and concurrent states are also formalized as processes. For the purposes of this paper, the formalization

framework is configured to read UML 1.4 [3] models specified in terms of XMI 1.1 [27] and generate Promela [8] specifications.

To use the SMV model checker [9, 26], Tanuan and Atlee [7] have developed a set of rules to translate a UML model into SMV’s specification language. Currently, there does not exist a tool that automatically translates UML models to SMV specifications. Therefore, we manually translate a UML model into an SMV specification using these rules.

2.3 Step 3: Analyzing a UML Model

Next, the developer uses the model checker to analyze the translated UML model for adherence to the previously specified property. If the formal analysis tool finds a violation of the property, then a violation trace is returned. The violation trace contains the sequence of steps performed by the system that lead to the violation of the property.

3 Theseus Visualization Framework

As shown in the shaded region of the activity diagram in **Fig. 1**, the Theseus visualization framework supports visually interpreting the analysis results generated by model checkers in terms of the original UML diagrams. For example, **Fig. 2(a)** depicts a state diagram that has been analyzed for the user interface element for our adaptive light controller case study that will be described in detail in Section 4. **Fig. 2(b)** is an excerpt of the corresponding violation trace generated by Spin. From the trace files, Theseus extracts four types of dynamic behavior to animate: (1) A state is visited; (2) A transition is taken; (3) A message is sent; and (4) A message is received. The Theseus visualization framework comprises two key components to depict this behavior: the Theseus trace processor and the Theseus visualization engine. The *Theseus visualization engine* takes the XML intermediate representation of the dynamic behavior from the trace output and the original UML model as inputs and produces the UML state diagram animations and UML sequence diagram generation. Next, we describe in more detail the Theseus trace processor and visualization engine.

3.1 Theseus Trace Processor

The objective of the violation trace processor (depicted in **Fig. 1**) is to identify the dynamic behavior within the violation trace file and to specify this behavior in an intermediate XML representation. The *Theseus trace processor* comprises a parser and a translator, where a grammar must be constructed to generate a parser for each syntactically unique trace file format. Note that different trace file formats will be generated by different model checkers or by the same model checker with different output options or instrumentation, which means different parsers will need to be generated for each unique format. However, a parser is reusable across trace output files generated from the analysis of different UML

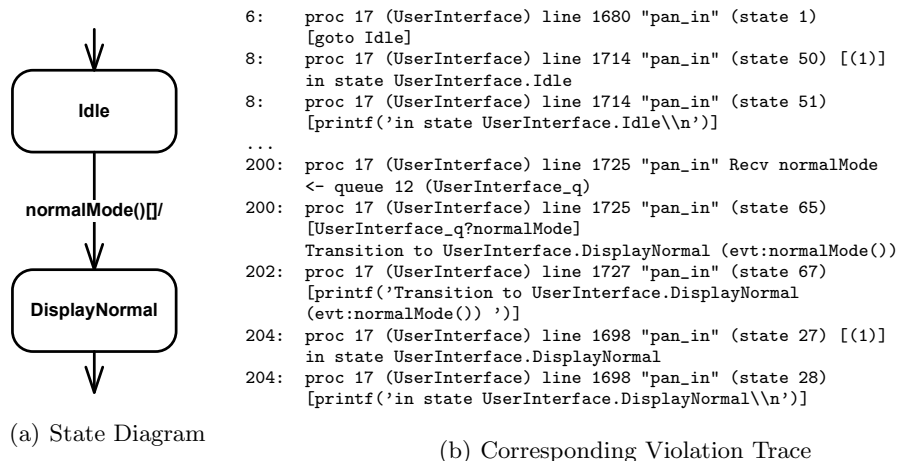


Fig. 2. Sample State Diagram and Violation Trace

models and/or different properties by the same model checker with the same output options selected. The parser constructs an ASG representation of the dynamic behavior specified by the trace file. The translator then traverses the ASG and creates an intermediate XML representation of the dynamic behavior.

An excerpt of a violation trace generated by Spin is depicted in **Fig. 2(b)**, which contains information from four different sources: the UML model, the Promela specification, from any instrumentation added by Hydra, and internal Spin information (e.g., line numbers, process number, Spin states, etc.). The Theseus parser extracts the information corresponding to the four dynamic behaviors of interest as an ASG (abstract syntax graph). We give examples for each as follows:

1. **A UML state is visited:**

6: proc 17 (UserInterface) line 1680 "pan_in" (state 1) [goto Idle]

This statement specifies that the `UserInterface` visits state `Idle`. The italicized part of the statement specifies Spin internal information that is irrelevant for visualization purposes. Specifically, *6: proc 17* represent the execution step and internal Spin process number, respectively. *line 1680 "pan_in" (state 1)* are the line number within and the file name of the trace file, and the Spin internal state, respectively.

2. **A UML transition is taken:**

Transition to UserInterface.DisplayNormal (evt:normalMode()) ^ Display.showNormMes)

This statement is produced by the instrumentation (from Hydra) added to the Promela specification. Spin can provide this information, but only by activating specific flags to generate even more verbose and cumbersome output. Therefore, since we have the ability to extend Hydra, for convenience we have added instrumentation to obtain this information. This statement denotes that the `UserInterface` transitions to state `DisplayNormal` as a result

of the `normalMode` event occurring. In addition, as a result of this transition being taken, the message `showNormMes` is sent to `Display`.

3. **A UML message is sent:**

201: proc 17 (UserInterface) line 1726 "pan_in" Send showNormMes → queue 13 (Display_q)

This statement specifies that `UserInterface` sends the message `showNormMes` to `Display`.

4. **A UML message is received:**

206: proc 11 (Display) line 1248 "pan_in" Recv showNormMes ← queue 13 (Display_q)

This statement specifies that `Display` receives the message `showNormMes`.

The Spin translator translates the ASG representation of the dynamic behavior generated by the parser into an XML intermediate format. Specifically, there is an intermediate XML specification for each of the four types of dynamic behavior. For example, **Fig. 3(a)** shows a sample XML element specifying that state `Idle` in class `UserInterface` is visited. **Fig. 3(b)** specifies that object `Display` sent a message named `showNormMes` to object `UserInterface`.

<pre> <Expression> <Process name="UserInterface"/> <Goto> <Read_location> <Process name="UserInterface"/> <State name="Idle"/> </Read_location> </Goto> </Expression> </pre>	<pre> <Expression> <Process name="UserInterface"/> <Send_Message> <Message name="showNormMes"/> <End_Transition> <Queue name="Display"/> </End_Transition> </Send_Message> </Expression> </pre>
--	---

(a) Visited State

(b) Sent Message

Fig. 3. Sample XML Elements

3.2 Visualization Engine

The visualization engine has been implemented in the ArgoUML [21] CASE tool as a plugin. ArgoUML was selected because of its open source application programming interface (API) that allows the creation of visualization plugins. Theseus provides two animation options: automatic playback and incremental playback. Automatic playback animates the complete violation trace; whereas, incremental playback animates the animation trace in a stepwise fashion (single or multi-step). The multi-step option is useful when there are a large number of steps in the violation trace and the developer suspects the first several steps may not be relevant to the violation. After skipping to a specific step, the developer is able to automatically play the remaining steps, or incrementally play the next step.

Theseus provides two mechanisms for visualizing violation traces on the UML model, *state diagram animation* and *sequence diagram generation*. Specifically, state diagram animation depicts that a state is visited (colored red when visited and turns yellow upon departure) and that a transition fires (in red). The generated sequence diagram is animated to depict that a message is sent (arrow in red) and received (arrow in blue). As such, Theseus depicts all four types of dynamic behavior useful for understanding a violation trace.

Both state diagram and sequence diagram animations help a developer to better understand the cause for a property violation. While the state diagram animation is better suited for understanding the behavior of an individual object, the generated sequence diagram helps a developer to understand the context for a property violation in terms of object interaction. Note that typically a UML diagram may have several state diagrams, each of which represents the behavior of a particular object in the system. Currently, Theseus displays the state diagram of a particular object, depending on the part of the counterexample being traversed. As events and messages communicate among objects, the corresponding object's state diagram is displayed. In future versions, we plan to display more than one state diagram at a time in addition to the sequence diagram.

4 Case Study

This section describes an industrial case study we performed to validate our visualization framework. Specifically, we used the object analysis patterns to create a UML model of an embedded system application, then used Hydra to generate a formal specification of the UML model, used Spin to verify critical system properties specified with SPIDER, and used Theseus to visualize the analysis results in terms of the original UML diagrams. Due to space constraints, we do not include a case study for the SMV visualization, but a description may be found in [28].

4.1 Adaptive Light Control System

The adaptive light control system (ALCS) is responsible for moderating the lights in a room. A class diagram depicting the structure of this system is depicted in **Fig. 4**. The class attributes and operations have been elided due to space constraints.

The primary function of the ALCS is to ensure that if the room is occupied, then the room is sufficiently illuminated, either by natural light or by the lamps. The ALCS comprises a switch for manually turning on the lights, a display for communicating messages to a user, a motion sensor for detecting that the room is occupied, a brightness sensor for detecting the current illumination level of the room, and a dimmer that controls the brightness of the lamps. The *Controller Decompose*, *Actuator-Sensor*, *User Interface*, *Computing Component*, *Fault Handling*, and *Detector-Corrector* object analysis patterns have been used in the specification of the structure and behavior of the ALCS. For additional

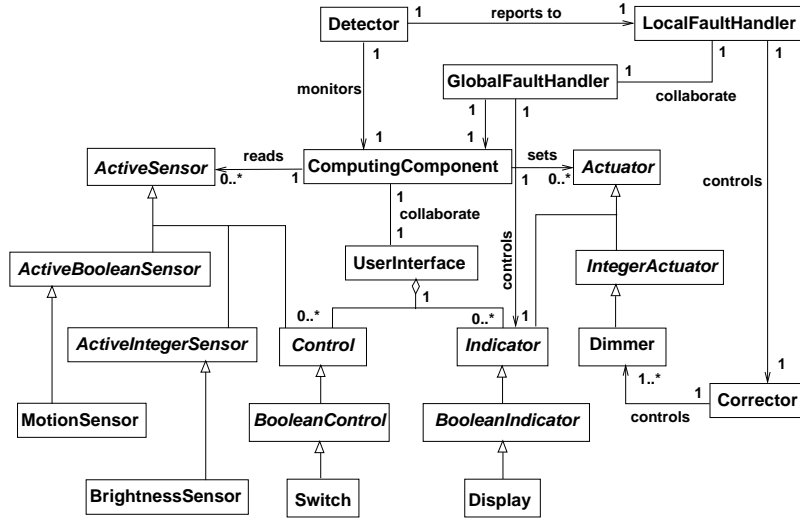


Fig. 4. Class Diagram of the ALCS

details about these patterns, please refer to [16]. Note, **Fig. 2(a)** describes a portion of the behavior of the `UserInterface`.

4.2 Property Specification and Analysis

We analyzed the UML model for the ALCS using the Spin model checker. First, we used Hydra to translate an XMI representation of the UML model into Promela. Second, we used SPIDER to formally specify properties to be satisfied by the model. For example, using SPIDER, we created the following natural language property:

“Globally, it is always the case that if the initialization has succeeded, then eventually the display shows the initialization succeeded message.” (1)

SPIDER then extracts UML model elements from the ALCS model to instantiate the free-form text the initialization has succeeded and the display shows the initialization succeeded message with model-specific elements. The initialization in the ALCS has succeeded if the `lightStatus` of the `ComputingComponent` is set to value 1. Therefore, the initialization has succeeded is replaced with `ComputingComponent.lightStatus=1`. Similarly, the text the displays shows the initialization succeeded message is replaced with `call(Display.showNormMes())` to denote that the message `showNormMes()` of the `Display` is called. Thus, we obtain the following instantiated natural language (NL) property:

“Globally, it is always the case that if `ComputingComponent.lightStatus=1`, then eventually `call(Display.showNormalMes())`.” (2)

From this specification, SPIDER automatically creates the formal specification of the property in LTL:

$$\begin{aligned} & \Box((\text{ComputingComponent.lightStatus}=1) & (3) \\ & \rightarrow \Diamond(\text{call}(\text{Display.showNormMes()})) \end{aligned}$$

At this point, SPIDER invokes Spin with the Promela model of the ALCS and the LTL property. In this case, model checking detected a violation and Spin generated a violation trace.

4.3 Property Visualization

Theseus processed the violation trace and visualized the counterexample in terms of the original UML state diagrams and a sequence diagram. A screen shot of a state diagram animated to depict one step of the violation trace is depicted in **Fig. 5**, where the key thing to note is the different colors of the states and the transitions. In this case, we are viewing the state diagram for the `UserInterface` object. (The intent of these figures is not to read the individual names of states or transitions, but to note the color changes – or the levels of shading in gray scale.) A screen shot of the sequence diagram generated by Theseus depicts the violation trace shown in **Fig. 6**. Using the Theseus visualizations of the violation path, we were able to locate the source of the error and revise the UML model accordingly. Rerunning the overall process yielded no further violations. Without Theseus, we would be forced to understand the syntax and semantics of the trace output, determine the relationship between the output and the UML model, and then locate within the UML model the corresponding error.

5 Related Work

Numerous CASE tools [11–15, 29] provide visualization support for UML model simulation. To the best of our knowledge, they do not allow the visualization of violation traces gathered during model checking analysis in terms of the original UML diagrams. Most formal analysis tools, in contrast, offer visualization capabilities in terms of the analysis models, such as Spin [8] and UPPAAL [30]. However, this visualization is on the level of the description language of the formal analysis tool and not at a more abstract level, such as a UML model.

Other tools visualize analysis results from model checkers in terms of UML. vUML [6] translates UML diagrams into Promela and uses Spin for analysis purposes. Violation traces revealed by formal analysis may be displayed in terms of UML sequence diagrams. To keep the model checking process transparent,

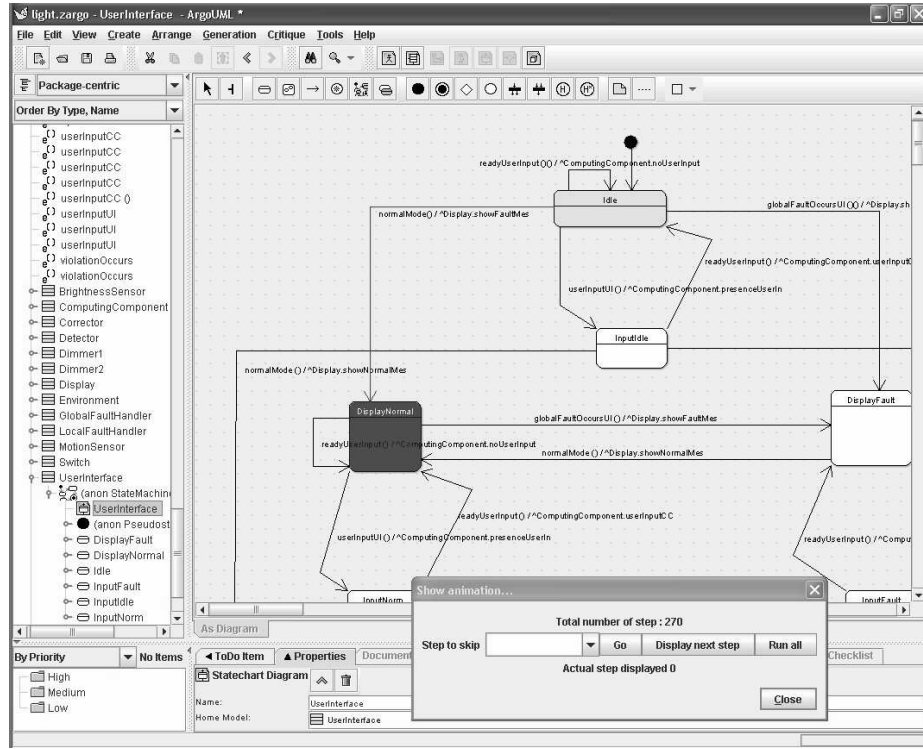


Fig. 5. Theseus Animation of Adaptive Light Controller Violation Path

vUML focuses on the analysis of more general properties, such as deadlocks and livelocks. Differing from our work, vUML only supports the translation of UML models to Promela, does not support the construction of property specification in terms of natural language or formal specification languages, and does not offer state diagram animation capabilities. MOCES [31] translates Statemate [13] state charts into Promela. The semantics of the Statemate state charts differs from the semantics for UML state diagrams [32]. In addition, MOCES only allows the analysis of a single state chart, while our tool analyzes behavior captured in a collection of collaborating state machines. Hugo/RT [33] supports the analysis of UML diagrams using Spin or UPPAAL [30]. In addition, Hugo/RT can translate a violation trace produced by these analysis tools to a representation in terms of UML elements. However, Hugo/RT provides a proprietary textual UML representation and does not interactively display the violation trace in terms of a graphical UML representation in a CASE tool.

In summary, none of the aforementioned tools combines the capability of displaying analysis results in terms of UML sequence and state diagrams and the customizability towards numerous formal analysis tools.

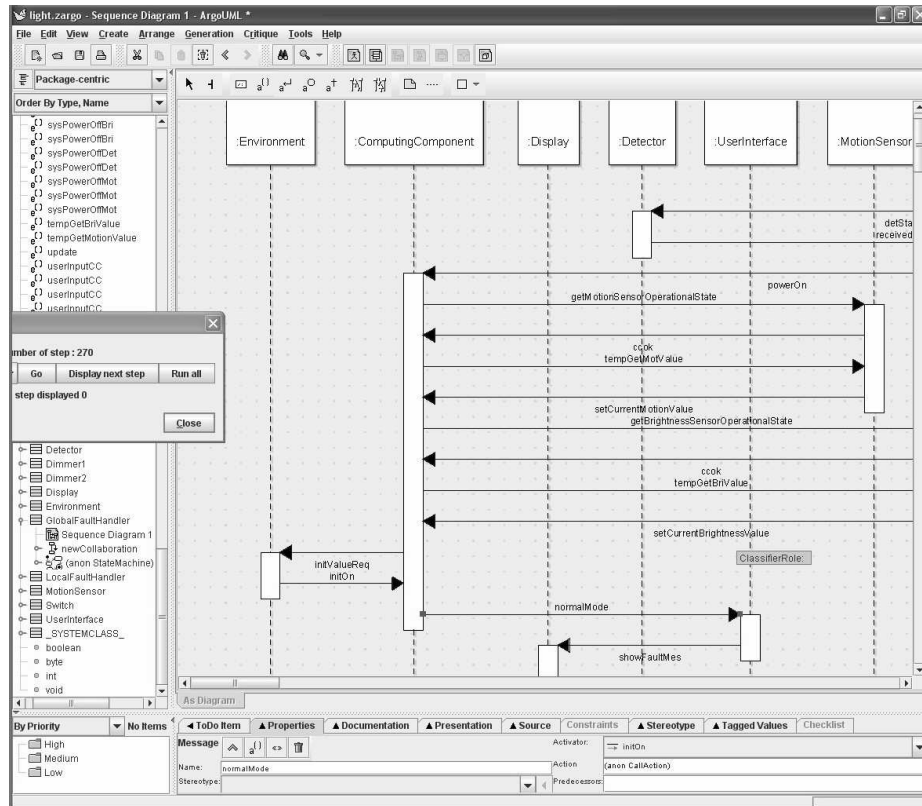


Fig. 6. Theseus Generated Sequence Diagram depiction of Adaptive Light Controller Violation Path

6 Conclusions

This paper has described a generic visualization framework that provides a critical link in a roundtrip-engineering process for modeling and analyzing embedded systems. The prototype of this visualization framework, offers three key benefits to UML modelers who want to model check their UML diagrams. First, Theseus supports modelers who are not proficient in interpreting the verbose and often cryptic analysis results generated by model checkers, by locating the source of the error identified by the violation trace. Theseus visually animates the violation trace on the UML state diagrams and a generated sequence diagram. Second, Theseus is extensible to other formal analysis tools beyond the ones mentioned in this paper. To extend Theseus to visualize output from other analysis tools, a specific Theseus trace processor needs to be created. The parser of the trace processor depends on the formalization rules (i.e., the rules for mapping UML to the target specification language of a given analysis tool) and the violation trace output options used in the model checker (including any instrumentation added

to the trace output). The translator of the trace processor, however, depends only on the formalization rules, and is potentially reusable for different violation trace output options. Currently, we have developed trace processors that support output generated by the SMV and Spin model checkers. Independent of the model checker, the formalization rules, and the output options, the Theseus visualization engine is reusable across the trace output for different state-based analysis tools. Third, Theseus completes the roundtrip modeling and analysis process for embedded systems by enabling a developer to automatically formalize a UML model, specify natural language properties that the model must satisfy, analyze the model for adherence to these properties, and visualize property violations in terms of the original UML diagrams.

Future work will include extending Theseus in many different ways. First, we are extending Theseus to view multiple state diagrams side-by-side during animation. Additionally, we are investigating how to extend the Theseus framework to visualize the analysis results from complementary model checkers, such as the real-time model checkers Kronos [34] and UPPAAL [30]. Finally, we are exploring a more seamless integration between the tools and the steps of our roundtrip-engineering process for modeling and analysis. The biggest challenge has been the lack of a universally accepted standard for data interchange between tools from third parties.

Bibliography

- [1] Douglass, B.P.: *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks and Patterns*. Addison-Wesley (1999)
- [2] Douglass, B.P.: *Real-Time Design Patterns*. Addison-Wesley (2003)
- [3] Object Management Group: *UML Specifications, Version 1.4 (2002)* <http://www.omg.org/cgi-bin/doc?formal/04-07-02>.
- [4] Object Management Group: *Model driven architecture*. <http://www.omg.org/mda/> (2005)
- [5] McUmbler, W.E., Cheng, B.H.C.: A general framework for formalizing UML with formal languages. In: *Proc. of the IEEE Int. Conf. on Software Engineering (ICSE01)*, Toronto, Canada (2001)
- [6] Lilius, J., Paltor, I.P.: vUML: A tool for verifying UML models. In: *Proc. of the 14th IEEE Int. Conf. on Automated Software Engineering*, Washington, DC (1999)
- [7] Tanuan, M.C.: *Automated Analysis of Unified Modeling Language (UML) Specifications*. Master's thesis, University of Waterloo, Canada (2001)
- [8] Holzmann, G.: *The Spin Model Checker*. (2003)
- [9] McMillan, K.L.: *Getting started with SMV* (1999)
- [10] IBM: *Rational Rose XDE Developer*. <http://www-306.ibm.com/software/awdtools/developer/rosexde/> (2005)
- [11] Telelogic: *ObjectGEODE*. <http://www.telelogic.com/> (2005)
- [12] I-logix: *Rhapsody*. <http://www.ilogix.com/> (2005)
- [13] I-logix: *Statemate*. <http://www.ilogix.com/> (2005)
- [14] ARTiSAN Software: *Real-time Studio*. <http://www.artisansw.com> (2005)
- [15] Ho, W.M., Jézéquel, J.M., Guennec, A.L., Pennaneac, F.: UMLAUT: an extendible UML transformation framework. In: *Proc. of the Automated Software Engineering Conf.*, Florida (1999)

- [16] Konrad, S., Cheng, B.H.C., Campbell, L.A.: Object Analysis Patterns for Embedded Systems. *IEEE Trans. on Software Engineering* **30**(12) (2004) 970 – 992
- [17] Konrad, S., Cheng, B.H.C.: Facilitating the construction of specification pattern-based properties. In: *Proc. of the IEEE Int. Requirements Engineering Conf. (RE05)*, Paris, France (2005)
- [18] Konrad, S., Cheng, B.H.C.: Automated analysis of natural language properties for UML models. In Bruel, J.M., ed.: *Satellite Events at the MoDELS 2005 Conf.: MoDELS 2005 Int. Workshops*, Montego Bay, Jamaica, October 2-7, 2005, Revised Selected Papers. Volume 3844 of *Lecture Notes in Computer Science.*, Springer-Verlag GmbH (2006) 48–57
- [19] Emerson, E.A., Halpern, J.Y.: "Sometimes" and "Not Never" revisited: On branching versus linear time temporal logic. *Journal of the ACM* **33**(1) (1986) 151 – 178
- [20] Manna, Z., Pnueli, A.: *The temporal logic of reactive and concurrent systems.* Springer-Verlag New York, Inc. (1992)
- [21] Tigris.org: ArgoUML: The project home. <http://argouml.tigris.org/11> (2005)
- [22] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley (1994)
- [23] Konrad, S., Cheng, B.H.C.: Real-time specification patterns. In: *Proc. of the Int. Conf. on Software Engineering (ICSE05)*, St Louis, MO, USA (2005)
- [24] Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: *Proc. of the 21st Int. Conf. on Software Engineering*, IEEE Computer Society Press (1999) 411–420
- [25] Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. on Programming Languages and Systems* **2** (1986) 244 – 263
- [26] McMillan, K.L.: *Symbolic Model Checking.* PhD thesis, Carnegie Mellon University (1993)
- [27] Object Management Group: *OMG-XML metadata interchange (XMI) specification, v1.1* (2000) <http://www.omg.org/cgi-bin/doc?formal/00-11-02>.
- [28] Kamdoun, S.: *Facilitating the roundtrip engineering of model-driven software architecture.* Master's thesis, Michigan State University (2006)
- [29] Nickel, U., Niere, J., Zündorf, A.: The FUJABA environment. In: *Proc. of the 22nd Int. Conf. on Software Engineering*, New York, NY, USA, ACM Press (2000) 742–745
- [30] Pettersson, P., Larsen, K.G.: UPPAAL2k. *Bulletin of the European Association for Theoretical Computer Science* **70** (2000) 40–44
- [31] Mikk, E., Lakhnech, Y., Siegel, M., Holzmann, G.J.: Implementing statecharts in promela/spin. In: *WIFT '98: Proc. of the Second IEEE Workshop on Industrial Strength Formal Specification Techniques* (1998) 90
- [32] Crane, M.L., Dingel, J.: UML vs. classical vs. Rhapsody statecharts: Not all models are created equal. In: *Proc. of the ACM/IEEE 8th Int. Conf. on Model Driven Engineering Languages and Systems.* (2005)
- [33] Knapp, A., Merz, S., Rauh, C.: Model checking timed UML state machines and collaborations. In Damm, W., Olderog, E.R., eds.: *7th Int. Symposium Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT 2002)*. Volume 2469 of *Lecture Notes in Computer Science.*, Oldenburg, Germany, Springer-Verlag (2002) 395–414
- [34] Bozga, M., Daws, C., Maler, O., Olivero, A., Tripakis, S., Yovine, S.: *Kronos: a model-checking tool for real-time systems* (1998)