Validating Real-Time Behavioral Patterns of Embedded Controllers

Jagannath Aghav, Claude Petitpierre

School of Computer and Communication Sciences
Swiss Federal Institute of Technology
CH-1015 Lausanne Switzerland
{jagannath.aghav,claude.petitpierre}@epfl.ch

Abstract. The functional complexity of hardware and software systems is growing exponentially. This demands an integration of system validation in the design phase to guarantee that a concrete implementation conforms to the modeled requirements. System validation process involves use of advanced systematic techniques and tools. In this paper we present a method for validating the real-time behavioral patterns of control intensive applications. The real-time behavioral patterns of embedded controllers are modeled using Statechart diagrams of Unified Modeling Language (UML). Concrete implementation of the Statechart diagrams consists of active objects in Java programming language with synchronous communication. In the implementation using specified timed annotations, we validate the real-time constraints to be satisfied by embedded controllers. We shall illustrate the validation process with an example of a gear controller used in automobiles, and furthermore, provide an algorithmic solution.

Keywords

Unified Modeling Language, Statechart diagrams, Design Patterns, Embedded Systems, Reactive Systems, Validation and Verification.

1 Introduction

Growth in the technology of manufacturing microprocessors and decline in the prices have made a revolution of supplanting all electronic systems to single chip computers. The application-specific single chip computers or embedded systems [4] are often utilized in control intensive applications. Such systems or embedded controllers behaving with multiple functionality are fanned out in everyday life. The high cost of failures makes validation of system essential. These hardware-software co-designed systems are substantially increasing in complexity and size. Thus forming system validation a formidable challenge. The rapid development in this area makes it necessary to have systematic methods and tools for designing and validation.

Unified Modeling Language (UML) [7] is used by software developers in the early stages of software development for expressing object oriented models and designs. The generalized solutions are mined in form of architectural patterns and behavioral patterns [8] or broadly as design patterns [9]. Various real-time models [2] and their finite state formalisms [3] are employed for validation of real time systems specified in different programming languages. We consider the design and validation of controllers for timed systems [1, 15] in UML. The integration of validation methods in development phase of UML will assist in devising the design patterns of complex systems with high reliability.

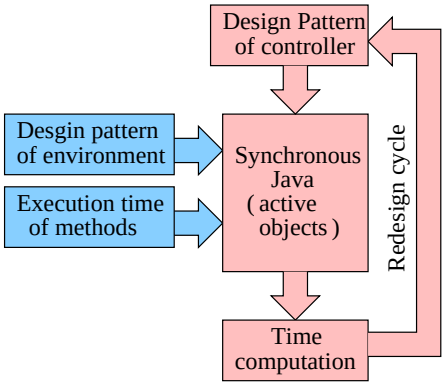

Fig. 1. Validation Process Cycle for Designing Controllers.

In this paper we describe a method for validating real-time behavioral patterns in the design of embedded controllers. The reactive nature of controllers allow the classification of the behavioral patterns into two categories: behavioral patterns of program controllers and of components in environment. The specification of environment as model for validation originates from Sifakis [6] where non-computing regions of program are specified using event relations such as *cumulative, separate* and *coalescent*. We continue to apply the environment modeling for identifying the length of execution sequences in the UML framework. Figure 1 shows the general view of validation process of our approach which is a cyclic process like different play-in-scenarios in [10]. The steps in validation process are as follows:

(i) Model the behavioral pattern of the program controller in Statechart diagrams of UML including time constraints.
(ii) Model the behavioral pattern of corresponding component being controlled in the environment.
(iii) Implement the code from Statecharts diagrams as active objects having synchronous communication.

(iv) Read execution times in Java code. The time values are based on selected target architecture. Either the compiler supports the reading of time values or the designer supplies. Labels are annotated based on notations of *TimeC* in [11] or in [6, 16] for synchronous programming language ESTEREL [5].

(v) Construct a finite state automaton with time annotated labels on transitions from the concrete realization.

(vi) Compute the longest response time on each stimuli of the environment.

(vii) Check the real-time constraints in all possible execution paths for satisfaction. The longest time response of all paths is displayed on the transition of Statechart diagram of the controller.

The main contributions of the paper are, a method for validating real-time behavioral patterns of embedded controllers and an algorithmic solution to the validation process. We provide an illustration of our method with an industrial case study example of a gear controller. The rest of the paper is structured as follows: In the sequel we present briefly an example of a gear controller used in automobiles. In Section 3 we describe the modeling of architectural, behavioral patterns of the example and demonstrate timed annotations in code implementation. Section 4 presents the validation of real-time constraints and an algorithmic solution for computing time responses. In Section 5 we present the conclusions of the paper.

2 Gear Controller

We take an example of a gear controller (for more details see [12]) proposed by Mecel. The gear controller has five main components: Clutch, Gear Box, Engine, Interface and Controller apart from the linking and plug in components. The main controller is composed in parts or modules. These modules together with corresponding components forms the complete gear controller as shown in Figure 2. For the validation we categorize the composed system into two parts:

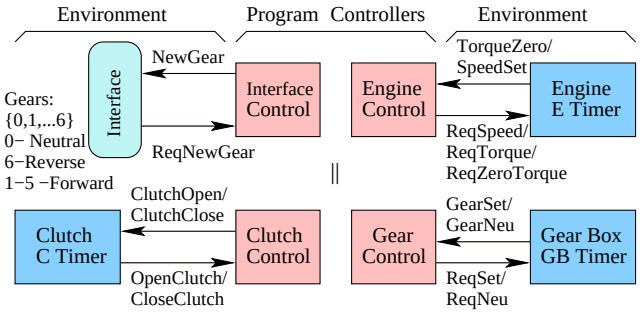

Fig. 2. Composition of Mecel's Gear Controller.

controller part as *program* and component part as *environment*. The components have time requirements to be satisfied for the correct functioning of the gear controller. For example the time requirements on clutch operation is as follows. The clutch should change state from close to open or vice versa within 100 to 150 time units.[†] Next we shall present the modeling and concrete realization of the example.

3 Implementation

UML provides stereotype of classes, protocols and Statechart diagrams to model event driven systems. Using capsule as a stereotype class, we model each independent control element of gear controller forming the architectural class diagram. The dynamic behavior of each element is modeled using a Statechart diagram. In the following we present architectural patterns, behavioral patterns and timed annotations of gear controller example.

3.1 Architectural Patterns

This is the static structure of the full gear controller that has a capsule for each control part and protocol stereotype for communication. The architecture is composed with main capsule `MainGearController` which comprises capsules of all other components and controllers. Figure 3 shows architectural class diagram of gear controller.

3.2 Behavioral Patterns

The dynamic behavior of program controller part and its corresponding component is specified using Statechart diagrams. The component or environment behavior is specified to assist the validation. Figure 4 shows the behavioral pattern of clutch controller or program part. Figure 5 shows behavioral pattern of clutch component or environment. The behaviors of other components and program controllers are modeled in a similar way. Figures are not shown due to lack of space. The interactions between different program controllers are modeled using collaboration diagrams.

3.3 Annotating Synchronous Active Objects

From the behavioral patterns the concrete implementation is generated as follows: For each controller and its corresponding component active objects are created. These active objects communicate synchronously and such objects are known as synchronous active objects [13, 14]. The communication between the controllers and its components is synchronized by having a common method name in Java. We annotate the syntax of synchronization calls with following label structure within the comments beginning with special symbol '!'.

[†] For validation purpose measurement is in time units instead of milliseconds. Based on target architecture exact times are computed.

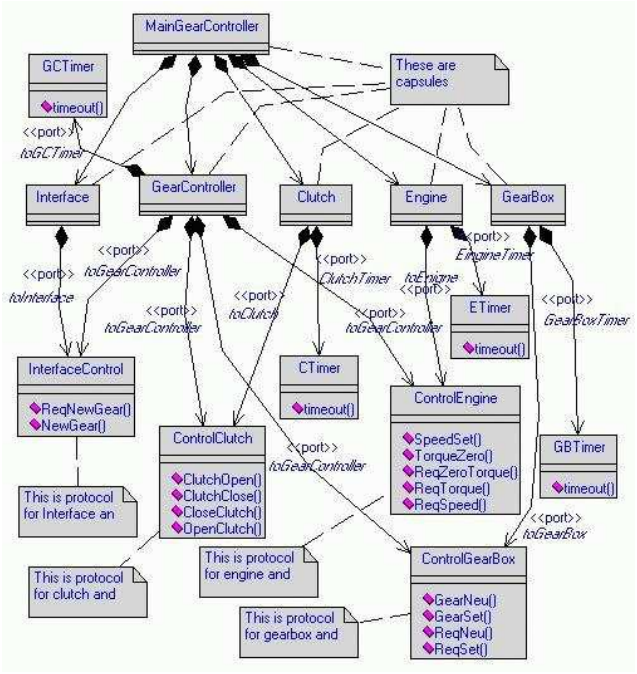

Fig. 3. Class Diagram of Gear Controller.

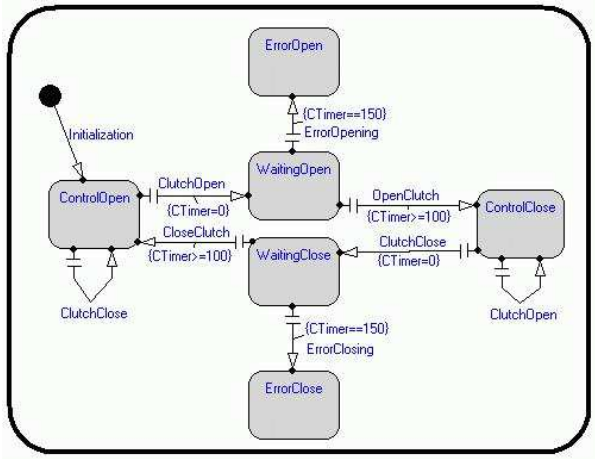

Fig. 4. Statechart Diagram of Clutch Controller.

//! { *Calling active object number, receiving active object number, method name, time units* }

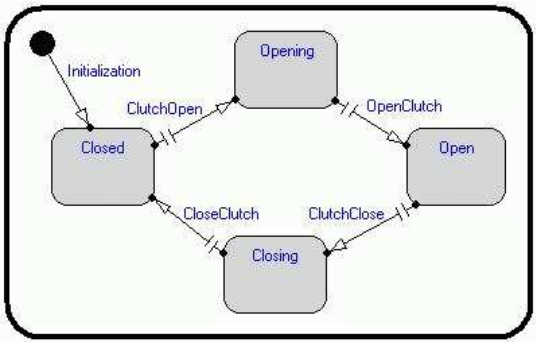

Fig. 5. Statechart Diagram of Clutch Component

The structure of label is composed with the following information: (i) *Caller* active object number. (ii) *Receiver* active object number. (iii) The name of the method that is executed and (iv) The execution time of the method. The two categories of composition, the program controller and components of *environment* are specified in annotations as even and odd numbers respectively. For the example of simple clutch controller two active objects are created: `ClutchController` and the component `Clutch` with synchronous communication. The annotated source with the timing information appears as shown below:

```
public class Gear {
...
active class ClutchController{
...
accept OpenClutch;
//! {1, 0, OpenClutch, 35}
...
}
active class Clutch{
...
accept ClutchOpen;
//! {0, 1, ClutchOpen, 30}
...
}
```

As the code depicts the clutch controller is numbered with 0 and clutch component with 1.

4 Validation

In this section we present the validation of the finite state model generated from the java source implementation and from the timed annotations on synchronous active objects. The generated finite state model is a directed graph that

represents sequential execution of active objects. The time annotated labels on synchronization becomes the labels of the edges in the finite state model. For validation we detect the transitions that have communication from *environment* component to *Program* controller. Following algorithm computes time responses for the input finite state model.

Time Computation Algorithm Let us denote *program* controller with P and *environment* with E. $(P \uparrow E)$ denotes a transition or edge that has communication from *program* controller to *environment* component. The input to the algorithm is a file containing the description of finite state model. Output is the sum of execution times for all possible paths that has starting transition $(E \uparrow P)$ and a terminating transition of either $(P \uparrow E)$ or $(P \uparrow P)$. Here $(P \uparrow P)$ denotes transition type within a program controller that has no outgoing edges. The steps for computing time are as follows: Read the description of labels, edges, and vertexes from the input file. Search the new edge that has communication from environment to program controller, $(E \uparrow P)$. For the selected new edge, find the corresponding sinking vertex. From the sinking vertex find all possible paths ending on next new edge of type $(E \uparrow P)$. All the paths are terminating with either $(P \uparrow E)$ or $(P \uparrow P)$ type of edge. Compute the time on all the paths by summing up the execution times specified on the labels. Display the transition that takes longest time response into Statechart diagram of controller.

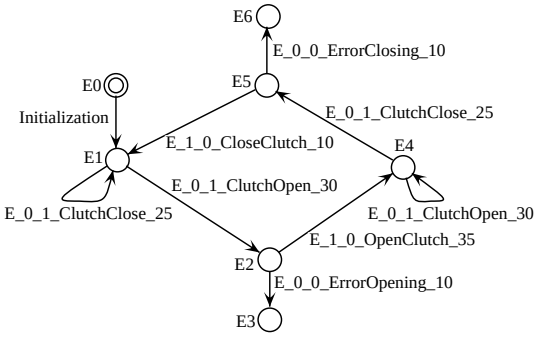

Fig. 6. Finite State Model of Clutch Controller

Figure 6 shows the finite state model representing execution of two synchronous active objects with timed annotated labels for the simple clutch controller example. Composition with other controllers is not taken in account for this simple case. The behavioral patterns of this model are shown in Figure 4 and Figure 5. The result of time computation for the model is as following:

```
E4→E6: Time(E_1_0_OpenClutch_35 + E_0_1_ClutchOpen_30 +
   E_0_1_ClutchClose_25 + E_0_0_ErrorClosing_10 ) = 100 units
```

The longest time taken by the controller for $(E \uparrow P)$ transition having `OpenClutch` method of Java is 100 units. The result is displayed in the Statechart diagram of controller.

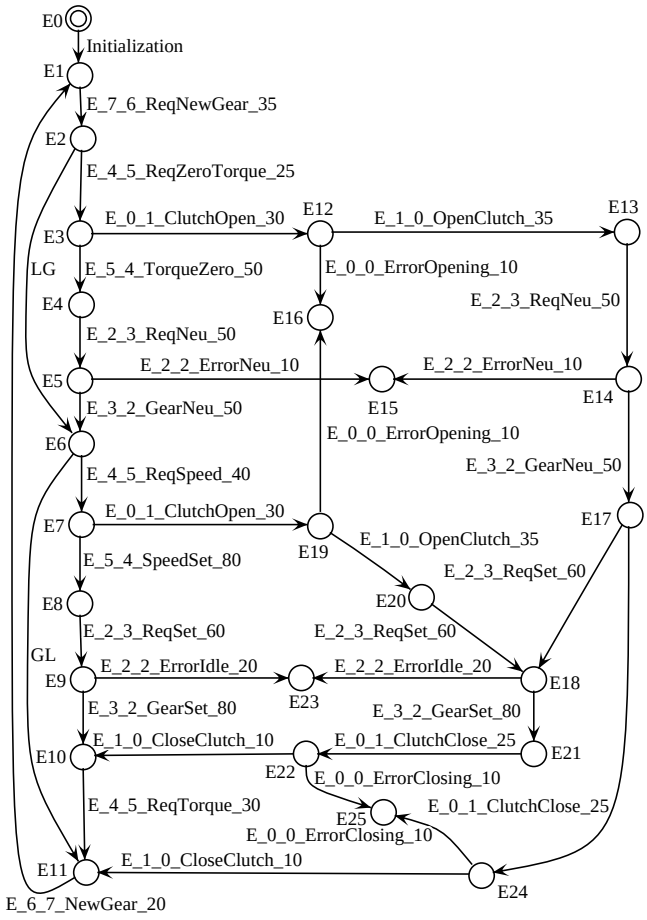

Fig. 7. Finite State Model of Gear Controller.

Now consider the finite state model of the composition for all four controllers.The environment statechart diagram of the respective components are provided for the validation process. The active objects for controllers of clutch, Gearbox, Engine and Interface are even numbered starting with 0, whereas the corresponding environment components are odd numbered starting from 1. Figure 7 shows the finite state model representing the execution of all synchronous

active objects with time annotated labels for gear controller example. The result of time computation for the model is as following:

$E_8 \rightarrow E_{23}$: `Time(E_5_4_SpeedSet_80 + E_2_3_ReqSet_60 + E_2_2_ErrorIdle_20) = 160 units`

The longest time taken by program controller for path is 160 units. There are two paths $E_{10} \rightarrow E_1$ and $E_{10}{}^* \rightarrow E_1$ showing the same sinking vertex E_{10} with two incoming $(E \uparrow P)$ type of transitions. The sum of execution time for the two paths is 130 units and 60 units respectively.

5 Conclusions

We have presented a method for validating real-time behavioral patterns of embedded controllers specified in statecharts diagrams of UML. The validation process is illustrated with an example of a gear controller. The method aims for the integration into the design phase of UML to assist in predicting the run time of behavioral patterns for a given target architecture and also in simplifying the constructions of timed systems. The method validates only real-time behavioral patterns and not the architectural patterns or internal inconsistencies.

References

1. E. Asarin, O. Maler, A. Pnueli and J. Sifakis. Controller synthesis for timed automata. In *Proc. System Structure and Control.* Elsevier, 1998.
2. R. Alur and T.A. Henzinger. Logics and models of real time: a survey. In J.W. de Bakker, K. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Real Time: Theory in Practice*, LNCS 600, pages 74–106. Springer-Verlag, 1992.
3. Rajeev Alur and David L Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126:183–235, 1994.
4. F. Balarin, P. Giusto, A. Jurecska, C. Passerone, E. Sentovich, B. Tabbara, M. Chiodo, H. Hsieh, L. Lavagno, A. Sangiovanni-Vincentelli, and K. Suzuki. *Hardware-Software Co-Design of Embedded Systems, The POLIS Approach.* Kluwer Academic Publishers, April 1997.
5. Gérard Berry and G Gonthier. The Esterel sychronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, 1992.
6. Valérie Bertin, Michel Poize, and Joseph Sifakis. Towards validated real-time software. In *Proceedings of the 12 th Euromicro Conference on Real Time Systems*, pages 157–164, Stockholm, June 2000.
7. Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide.* Addison-Wesley, Reading, Massachusetts, USA, 1 edition, 1999.
8. Bruce Powel Douglass. *Doing Hard Time: Developing Real-time Systems with UML, Objects, Frameworks, and Patterns.* Object Technology Series. Addison-Wesley, 1999.
9. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns.* Addison-Wesley, 1995.

10. David Harel. From play-in scenarios to code: An achievable dream. *Computer*, 34(1):53–60, January 2001.
11. Allen Leung, Krishna V Palem, and Amir Pnueli. TimeC: A time constraint language for ILP processor compilation. In K A Hawick and H A James, editors, *The 5^{th} Australian Conference on Parallel and Real Time Systems, Australia*, pages 57–71. Springer Verlag, 1998.
12. Magnus Lindahl, Paul Pettersson, and Wang Yi. Formal design and analysis of a gear controller. In *4th International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, pages 281–297. Springer Verlag, March-April 1998.
13. Synchronous Java. available on web site http://ltiwww.epfl.ch/sJava/.
14. Claude Petitpierre. Synchronous C++, a Language for Interactive Applications. *IEEE Computer*, pages 65–72, September 1998.
15. P. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–89, January 1989.
16. R K Shyamasundar and J V Aghav. Validating real-time constraints in embedded systems. In *Proceedings of IEEE Pacific Rim International Symposium on Dependable Computing, PRDC*, pages 347–355, Seoul, Korea, December 2001.