OMER-2 Workshop
Daimler-Chrysler Modeling Contest

Modeling S-Class Car Seat Control with AnyLogic

Alexei Filippov alf@xjtek.com,
Dr. Andrei Borshchev andrei@xjtek.com

St. Petersburg State Technical University,
XJ Technologies
http://www.xjtek.com
Fax: +7 (812) 2471639
21 Polytechnicheskaya street
St. Petersburg
Russia

Abstract: In this paper we give an overview of the car seat model that was developed for Daimler-Chrysler modeling contest in year 2001 and was awarded the 1st prize. We outline the OO UML-RT based modeling approach that was used and the simulation tool AnyLogic that supports it, and discuss their main advantages with respect to automotive area.

1 Introduction

Many different object-oriented methods for the development of embedded real-time systems have been made public during the last years. Some of them are already supported by off-the-shelf tools, others are still research prototypes. From the viewpoint of industry research, Daimler-Chrysler decided to identify the best development method. The comparison was performed in the form of a contest, where different methods were applied to the same problem. The real-life working specifications of the S-class car seat control were offered as the problem definition.

With over a decade experience in OO modeling and real-time systems, St. Petersburg Technical University and XJ Technologies took part in the contest. The model of the car seat was developed with the modeling and simulation tool AnyLogic from XJ Technologies that supports the extended UML-RT as the modeling language.

The result of that development work – an executable animated model of the car seat controller, conforming all specifications and supporting the predefined interfaces – demonstrated that the modeling approach supported by AnyLogic is highly applicable to the automotive area. The model appeared to be very compact, precise, well-structured and intuitive; it was awarded the 1st prize at the contest.

2 The Modeling Language of AnyLogic

The Unified Modeling Language (UML) that originally was proposed to handle complexity in software systems, has proven to have a strong set of concepts applicable across domains. AnyLogic utilizes the power of UML-RT (UML for Real Time, an extension of UML that roots to ROOM language) in simulation applications. AnyLogic supports those constructs of UML-RT that are necessary to construct fully executable models of high expressive power in multiple application areas. AnyLogic extends UML-RT so that the resulting language is sufficient to construct such models. Detailed information about the modeling language can be found in [BKR97], [BKS00] and [Bo01].

When developing a model in AnyLogic you develop classes of "active objects" representing real objects that have activities and states inside and interact with their surroundings. Active objects interact solely through interface elements - ports and variables. Active objects may inherit properties and encapsulate each other to any desired depth, so that the model is structured as a hierarchy (a tree) of instances. Behavior specification is strictly separated from structure.

Behavior in AnyLogic is specified in terms of statecharts, timers, events, and ports. Statecharts (enhanced UML state machines) are used to characterize event and time ordering of operations. They specify the states of the active object and transitions between them. Messages are used to model various information units passed between active objects. They also can inherit properties and encapsulate other messages. Messages are sent, received, and routed at ports. One can define arbitrary queuing and routing policies.

Interface variables are a useful extension to UML-RT supported by AnyLogic. They may be exposed at the active object interface and connected to other objects. One can define algebraic and differential equations over entities to model e.g. the physics of the evironmnet the control system is embedded. Moreover, one can embed sets of equations into the statechart states to capture compex interactions of discrete logic and continuous behavior. This enables hybrid simulation – the feature highly demanded by embedded system developers.

3 The Car Seat Model

The car seat has two groups of adjustments, three adjustments in each, memory for two seat positions, courtesy seat adjustment for climbing in and out, and two heating stages. There is a number of implementation restrictions, for example only one of two motors can be activated at a time, heating should be turned off when the motor is on, the motor should be turned off before reaching the end of movement range, voltage conditions, etc.

One of the primary contest requirements to models was a clear and natural division of the model into components. The model of the car seat is partitioned into several active object classes according to the original functional specification [DC00]. The structure of the main model class Controller is shown in the Figure 1.
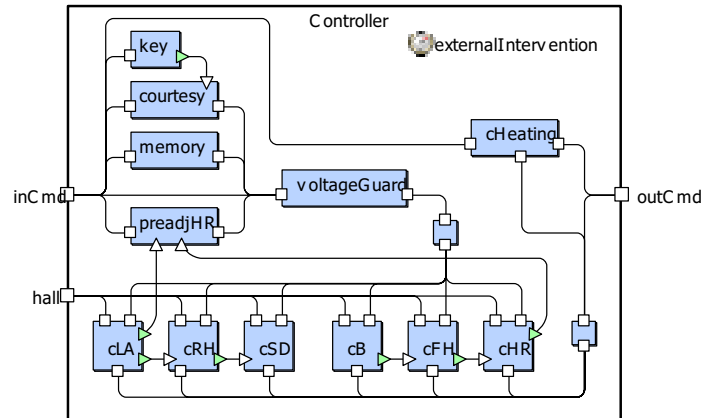
Figure 1 Structure of the Controller Class

The Controller object is composed of several sub-controllers:

- Ignition key sub-controller handles ignition key operations, enables and disables controller circuits according to the key state;
- Memory sub-controller handles position memory functions, allowing storing and restoring car seat positions;
- Courtesy adjustment sub-controller provides automatic movement of the seat to the backward position to help driver climb in and out of the car;
- Head restrain pre-adjustment sub-controller automatically adjust head restrain depending on the seat position;
- Heating sub-controller handles heating device functionality, switching between heating modes 1 and 2; temporarily turns heating off during seat movement operations to reduce power consumption.
- Motor sub-controllers. These six sub-controllers correspond to six possible movements of the seat. They also handle motor priorities.

The other essential feature of embedded and real-time systems in general, and the car seat in particular, is event- and time-ordering of operations. To efficiently specify that type of behavior we have intensively used the statechart notation. It is very powerful and flexible for describing various classes of algorithms, especially for reactive systems, when reaction on an event depends on the current system state. As a statechart example we could consider the memory controller algorithm show in the Figure 2.

The functionality implemented by this statechart is the following. The user has three buttons M, M1, and M2. It is possible to store up to two seat positions in the car memory. When the user presses button M and then within 2 sec either button M1 or button M2, the system stores the current seat position to the corresponding memory slot. If no button is pressed within 2 sec time interval after button M, the storing process is aborted. On the other hand the seat position restoring process is initiated when the user presses

either M1 or M2 buttons. Then the seat starts moving to the stored position. The movement stops when the final position is reached or if the user releases the button.
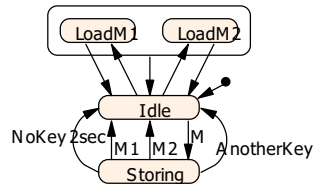

Figure 2 Memory Controller Main Statechart

Another functionality structuring technique that was used in the model is command pipelining. The user commands go though several processors and filters before they reach the main controller as shown in Figure 3. This pipelined architecture naturally lies on the UML-RT paradigm of messages, ports and statecharts.

commands / messages flow

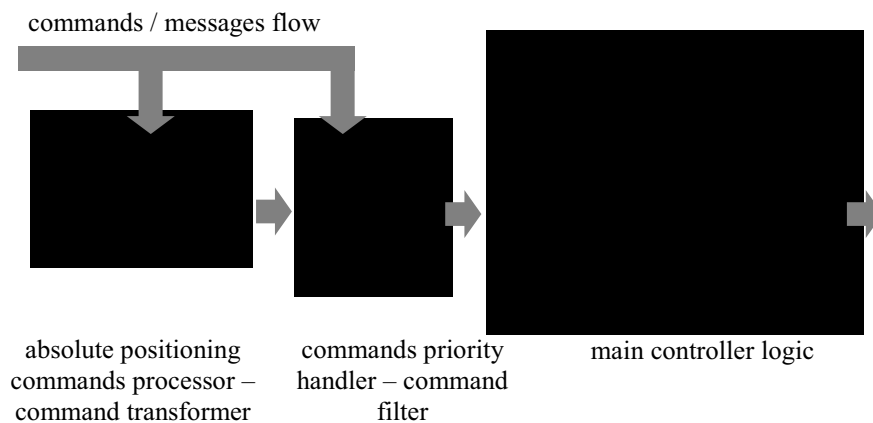| absolute positioning commands processor – command transformer | commands priority handler – command filter | main controller logic |

Figure 3 Command pipelining

An interactive animation was developed to enable visual testing, debugging and demonstration of the executable model. The animation displays the seat position, the motor and heating states, and has a number of controls via which the user can issue adjustment commands, use the position memory buttons, turn the ignition key, model the car speed, door opening, and voltage conditions. As AnyLogic generates 100% Java models, the animated model can be run over the Internet in a web browser. The model is available from XJ Technologies site at http://www.anylogic.com/applications/?model=carseat.

4 Conclusion

The main result of the modeling work is the proof of applicability of AnyLogic / UML-RT modeling language to the automotive area. The model developed is very compact and concise; its structure naturally reflects the problem logic. The two main diagram notations of UML-RT were extensively used: the structure/collaboration diagrams and statechart diagrams. Ports and message passing mechanism were used for command pipelining, and statecharts – for definition of the controller logic, including timing and ordering of operations. Inheritance enabled implantation of the functionality common to e.g. several subcontrollers in a base subcontroller class. Moreover, the ability of AnyLogic to model continuous and hybrid processes made it possible to put the controller into the experimental environment within using just one tool on one computer. In general, the approach that we used allowed us to spend very little time on adjusting the tool to our needs and to concentrate more on the essence of the problem.

Bibliography

[BKR97]   A. Borshchev, Yu. Karpov, V. Roudakov. Systems modeling, simulation and analysis using COVERS active objects. Proceedings of the 1997 Workshop on Engineering of Computer-Based Systems (ECBS '97) Monterey, CA, March 1997.

[BKS00]   A. Borshchev, Yu. Kolesov, Yu. Senichenkov. Java Engine for UML Based Hybrid State Machines. Proceedings of the 2000 Winter Simulation Conference, WSC'2000, Orlando, FL, December 2000.

[Bo01]   A. Borshchev. AnyLogic 4.0: Simulating Hybrid Systems with Extended UML-RT. To appear in: Simulation News Europe, Issue 31, April 2001.

[DC00]   Daimler-Chrysler, Detailed Functional Specification of the Car Seat Model, http://www.automotive-uml.de/mc/index.html

[XJ00]   Experimental Object Technologies. AnyLogic 4.0 User's Manual. Can be downloaded from http://www.xjtek.com/products/anylogic within the preview version of the tool itself.