

Motorola WEAVR: Aspect-Oriented Modeling for Code Generation and Simulation

Thomas Cottenier

Software & System Engineering Research Lab
Motorola Labs
1303 E. Algonquin Rd, IL01, Annex 2B
60196, Schaumburg, IL, USA
www.motorola.com/labs/
thomas.cottenier@motorola.com

Concurrent Programming Research Group
Illinois Institute of Technology
10 W. 31st St., Stuart Building, Room 226E
60616, Chicago, IL, USA
www.iit.edu/~concur/
cotttho@iit.edu

ABSTRACT

This tutorial aims at familiarizing academia with the tools and development practices used for Model-Driven Engineering in a large industrial context. The development environment used in production is presented as to familiarize the audience with *translation-oriented* style of UML 2.0 modeling. This style of modeling emphasizes precise modeling of behavior, model simulation and early verification, followed by fully automated code generation and execution within a test harness. Aspect-Oriented Software Development technologies are discussed in this context and an Aspect-Oriented Modeling engine developed at Motorola, the Motorola **WEAVR**, is presented. The tutorial emphasized the importance of model simulation and verification, and presents a simulation environment for aspect-oriented models. A visualization engine highlights the joinpoints within a model and allows developers to assess the effects of an aspect on a model, set breakpoints on aspect instantiations and step through the execution of the model before full aspect weaving is performed.

KEYWORDS

Model-Driven Engineering, Simulation, UML 2.0, Aspect-Oriented Modeling, Model Transformation, Verification and Testing,

OUTLINE

1. Model-Driven Engineering (45 min)

This Session presents the Model-Driven Engineering tools and practices used in production at Motorola. This environment is representative of the distributed, embedded, real-time and critical system industry.

1.1. The UML 2.0 (10 min)

The UML is a collection of languages that leaves multiple semantic variations points open to interpretation. This section reviews the different uses of the UML and shows how precise semantics can be associated to the language to make models executable.

1.1.1. Models as sketches

By far, the most common use of the UML is as an informal description language for the documentation and communication of system designs.

1.1.2. Models as blueprints

Models are also used as an architecture description language, and for the design of the static structure of a system. These models can be used to generate deployment descriptors and code skeletons, which are then further elaborated by developers which refine the generated code skeletons. The development model fits the traditional waterfall development process well. The industry is moving away from this development model (adoption of agile methods).

1.1.3. Executable models

Executable models can be compiled into executable artifacts. The complete, precise behavior of the system is defined at the level of the model by embedding code fragments into the models.

1.1.4. Translatable models

Translatable models do not commit to a specific language at the model level but make use of an action language that is translated into the target programming language. In this tutorial, we use the Specification and Description Language (SDL) action language. We also use the SDL interpretation of the UML 2.0 as the semantic base for model translation. This is achieved by loading the SDL profile for UML 2.0. This is the modeling environment adopted by Telelogic TAU.

1.2. Model-Driven Engineering (15 min)

This section discusses the different interpretations of the OMG Model-Driven Architecture and emphasizes the importance of model simulation and testing.

1.2.1. Model-Driven ArchitectureTM

The terminology of the MDA is reviewed, including Platform-Independent Models (PIM) and Platform-Specific Models (PSM).

1.2.2. Model Elaboration

This interpretation of the MDA involves manual and semi-automatic model transformation, as well as elaboration of the generated code.

1.2.3. Model Translation

The focus is on fully automated model transformation and code generation. Neither the transformed models nor the generated code are manually inspected or refined. The PSM is typically hidden from the developers

1.2.4. Model Simulation and Verification

The section emphasizes the importance of model simulation for early verification of the system. This section also discusses modeling in the context of agile development methodologies.

1.2.5. Execution in a Test Harness

Simulation cannot do full test coverage of a system. This section discusses systematic testing of models in a test harness using the Test and Test Conformance Notation (TTCN-3) standard.

1.3. Translation-Oriented Modeling (20 min)

This Section goes over the UML 2.0 diagrams used for translation-oriented modeling.

1.3.1. Reactive and Transformational Systems

The differences between reactive systems and transformational systems are discussed. This discussion is important with respect to the use of state machine diagrams.

1.3.2. System Specification and Architecture

The diagrams used for system specification and system architecture are reviewed. The focus is on the composite-structure architecture diagram and state-centric state machines. Figure 1 and Figure 2 show examples of those diagrams for a simple resource access server that implements the two-phase commit fault-tolerance protocol.

1.3.2.1. Class Diagrams

1.3.2.2. Composite-Structure Architecture Diagrams

1.3.2.3. State-Centric State Machine Diagrams

1.3.3. System Design and Implementation

System specifications such as state-centric state machine diagrams can be further refined into executable state machines, implemented as transition-centric state machines. The semantics of transition-centric state machines are detailed and the use of an action language is discussed. Figure 3 illustrates an implementation of the specification of Figure 2.

1.3.3.1. Transition-Centric State Machine Diagrams

1.3.3.2. Action Language

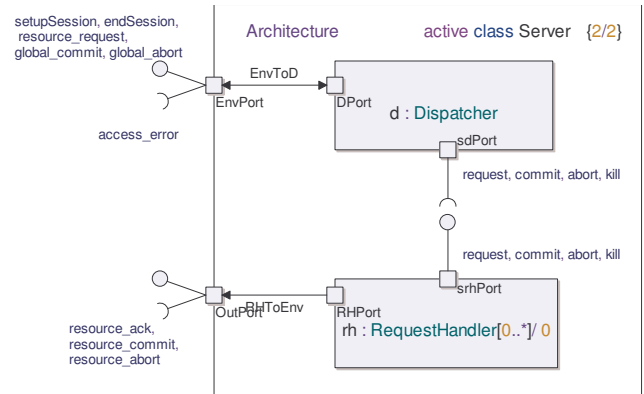


Figure 1. The Composite-Structure Architecture diagram for a simple server

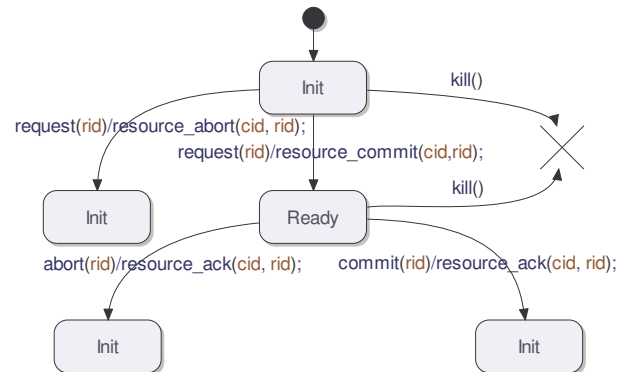


Figure 2. Specification of the observable behavior of a request handler as a state-centric state machine

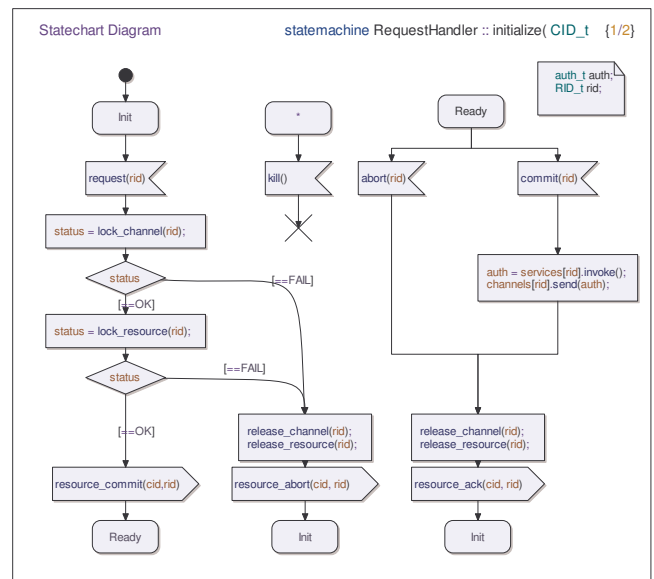


Figure 3. Implementation of a request handler as a transition-centric state machine

1.3.4. System Simulation and Verification

The simulation and verification environment is presented through a demo. The use of sequence diagrams is discussed in this context. Figure 4 shows a trace generated by the model verifier for the system defined in Figures 1, 2 and 3.

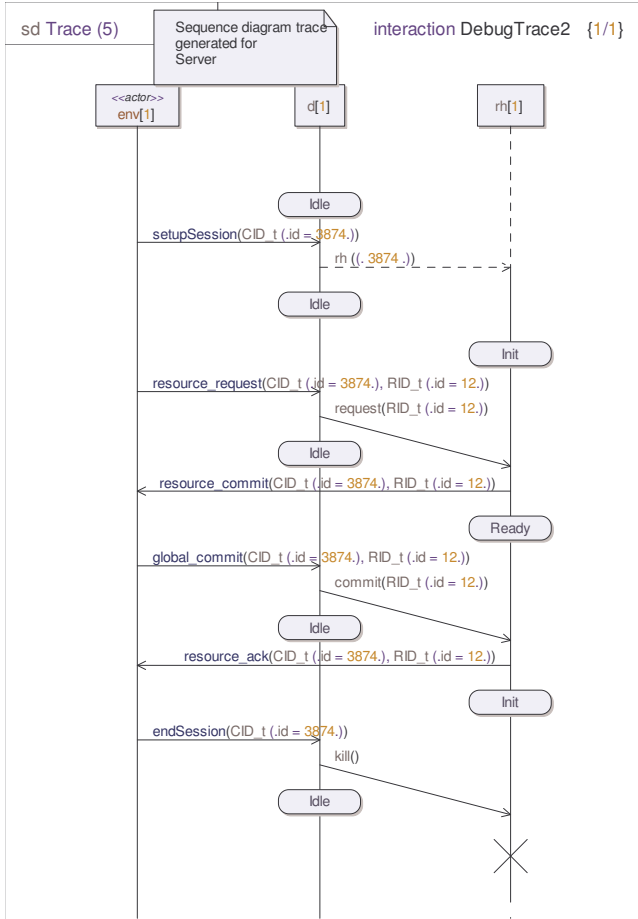


Figure 4. Sequence Diagram trace generated by the model simulator for a successful resource access, for the models of Figure 1 and 3.

2. Aspect-Oriented Modeling and Simulation (75 min)

This Session focuses on the languages constructs used by the Motorola *WEAVR* engine to capture aspects, perform simulation of Aspect-Oriented models and full model weaving.

2.1. Crosscutting Concerns in Models

(10 min)

The characteristics of crosscutting concerns in detailed behavior models is discussed and illustrated through real world examples.

2.1.1. State Machine Decomposition Units

The decomposition and reuse mechanisms of the Harel statechart [] are presented, including virtual state machines and behavioral inheritance.

2.1.2. Crosscutting Concerns in Transition-Centric State Machines

The structure of crosscutting concerns appearing in state machine diagrams is discussed with respect to the modularity units (state and sub-state machines, behavioral inheritance) used in transition-state machine diagrams. Crosscutting is illustrated through real world examples from production models.

2.2. Aspect-Oriented Modeling with the Motorola WEAVR (30 min)

The language constructs used for Aspect-Oriented Modeling are presented through examples. In particular, a new joinpoint model for transition-centric state machine is motivated and presented.

2.2.1. Aspects

We go over the Aspect-Oriented Modeling profile and the stereotypes used to define, deploy and compose aspects. Inter-type declarations are also discussed, including the introduction of signals and ports in active classes. Figure 5 illustrates the definition and deployment of a simple tracing aspect within the server defined in Figure 1.

2.2.1.1. Aspect Definition

2.2.1.2. Binding Diagram

2.2.1.3. Inter-type Declarations

2.2.1.4. Aspect Deployment Diagram

2.2.1.5. Aspect Composition

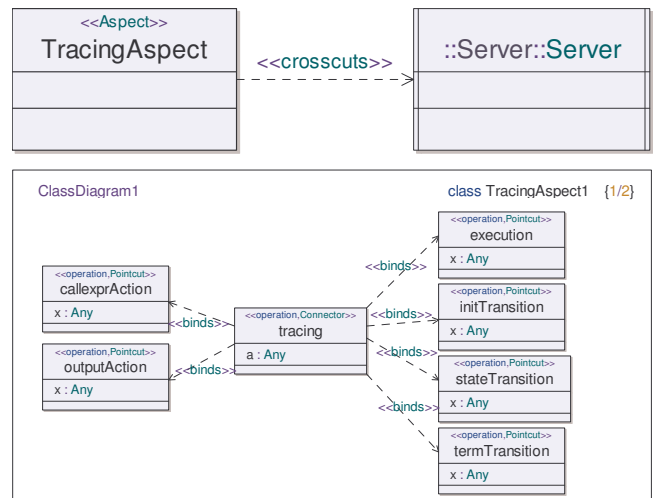


Figure 5. Deployment of an aspect that traces call expression actions, output actions and transitions in the scope of the server active class

2.2.2. Pointcuts

The joinpoint model of the *WEAVR* is based on actions and transitions. Action joinpoints refer to the action language used. They include call expression actions, create expression actions and output actions. Transition joinpoints refer to the selection of execution paths in the system that match a transition pointcut. This joinpoint model is discussed and the structure of pointcuts and pointcut designators is presented. The composition mechanisms of pointcuts are also presented. Figures 6, 7 and 8 show the pointcut designators of the pointcuts defined in Figure 5.

- 2.2.2.1. Action Pointcut Designators
- 2.2.2.2. Transition Pointcut Designators
- 2.2.2.3. Composition of Pointcuts

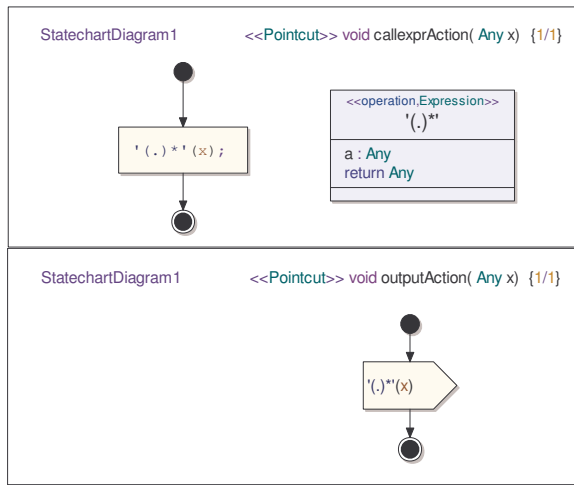


Figure 6. Action pointcut designators for a call and an output action, with one argument

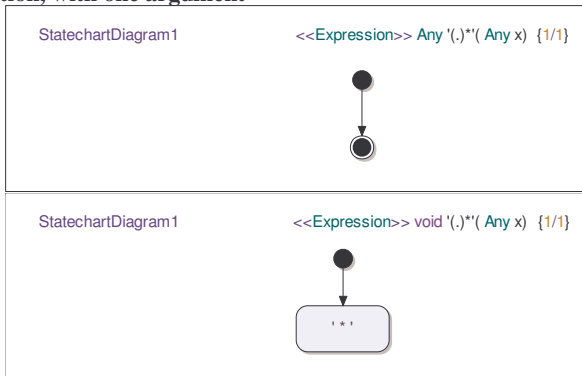


Figure 7. Expressions for the execution and initTransition pointcuts

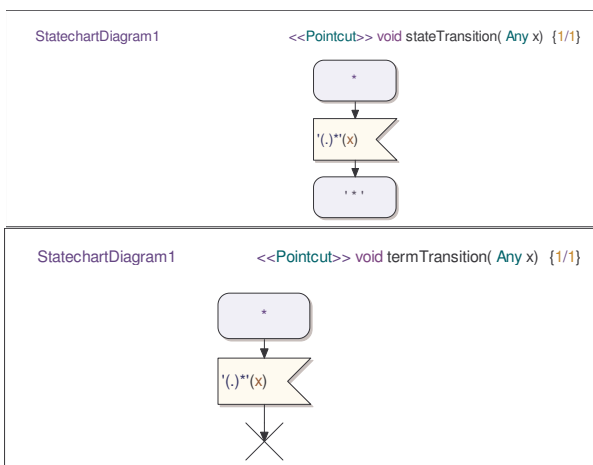


Figure 8. A transition pointcut designer as a triggered transition

2.2.3. Connectors

Connectors correspond to advices in AspectJ. The implementation of aspect connectors is discussed, including context passing and the use of the thisJoinPoint reflective API. Figure 9 shows the implementation of the tracing connector defined in Figure 5.

- 2.2.3.1. Context Passing
- 2.2.3.2. Use of the thisJoinPoint API

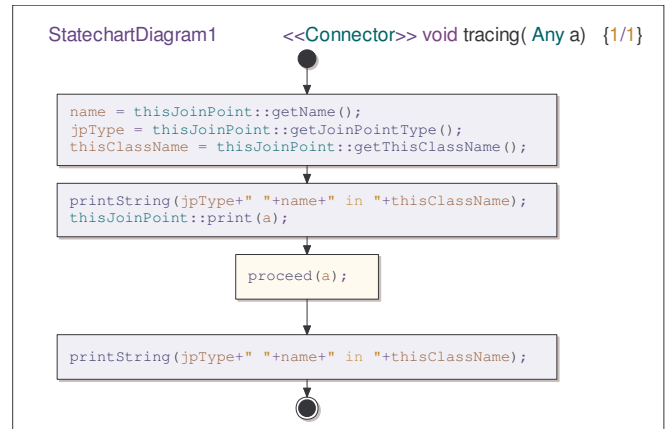


Figure 9. The tracing connector

2.3. Simulation of Aspect-Oriented Models (20 min)

For simulation purposes, the aspects defined in a model need to maintain a modular structure because woven models do not have a user-friendly representation. It is therefore required to provide the ability to simulate an Aspect-Oriented model in a way that is has semantics that are equivalent to the resulting woven model, without performing full aspect weaving.

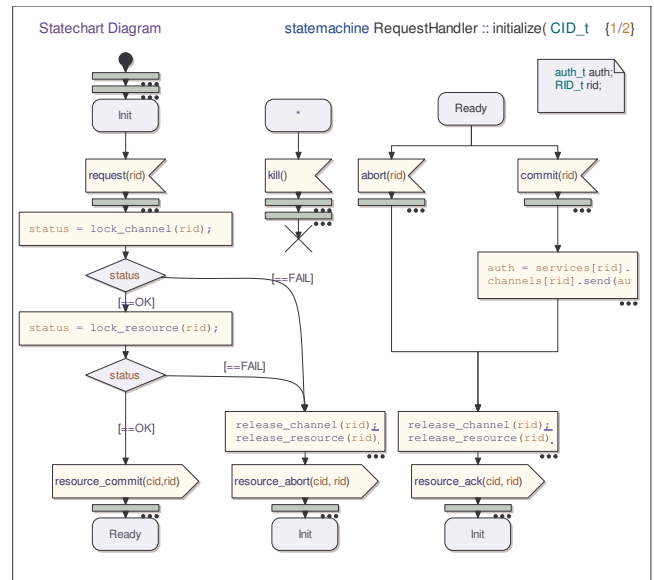


Figure 10. The request handler as annotated by the visualization engine after the tracing aspect has been applied

2.3.1. The WEAVR Visualization Engine

The joinpoints and the transition selections that match pointcut designators are annotated and highlighted in the models. This enables developers to assess the correctness of the pointcuts. Figure 10 shows the model of Figure 3 as annotated by the visualization engine. The symbols containing action joinpoints are highlighted in pink, while the transition selected by transition pointcuts are delimited by green marks.

2.3.2. Joinpoints Annotations and Connector Instances

When the user clicks one of the colored symbols, a state machine implementation diagram pops up and displays the instantiation of connectors applied to those locations.

Figure 11 shows an instance of the connector defined in Figure 9, in the context of the triggered transition from state Ready to state Init, triggered by the commit signal. This transition matches the pointcut of Figure 8.

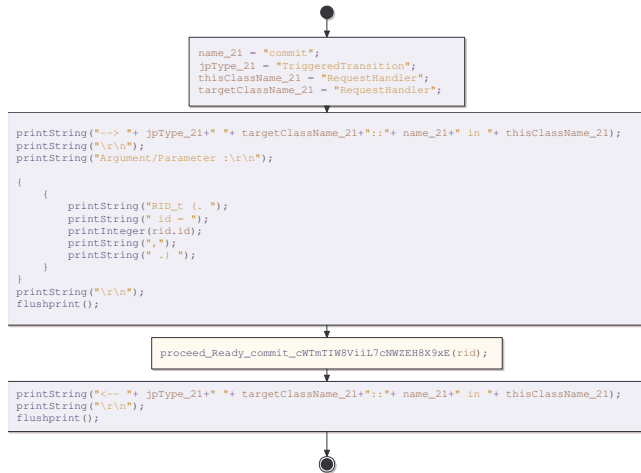


Figure 11. An instance of the connector of Figure 9, as instantiated in the context of the transition from Ready to Init, triggered by the commit signal

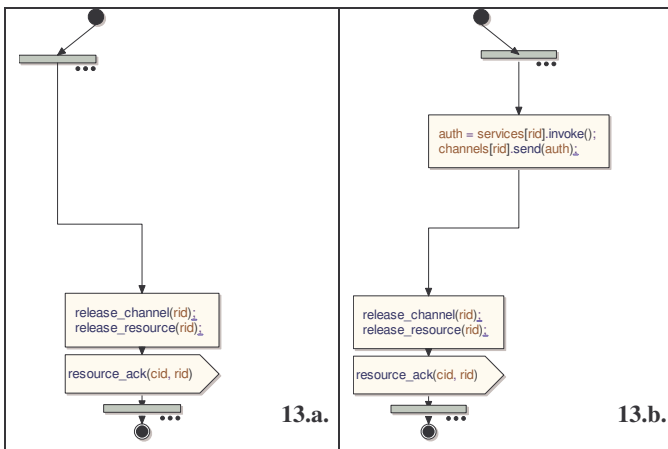


Figure 12. Representations of the matched transitions from state Ready to state Init, triggered by the abort and commit signals, respectively

2.3.3. Visualization of Transition Selections

The selection of transitions that match transition pointcuts has non-trivial semantics. It is therefore important to be able to visualize the transition selections. The visualization engine therefore generates state machine implementations that represent those selections. Figure 12 illustrates two execution paths within the state machine of Figure 3 that correspond to different joinpoints.

2.3.4. Simulation of Aspect-Oriented Models

Before model simulation, the WEAVR performs a series of transformations that allows the Aspect-Oriented model to be simulated using the views of the visualization engine, in a way that is semantically equivalent to final woven model. When the model simulator hits a joinpoint, it “jumps” to the corresponding connector instance, performing context passing, and returns control to the joinpoint after the execution of the connector instance. The engine also decomposes the generated sequence traces according to the action occurrence that correspond to aspects. Figure 13 shows a decomposed trace generated in the presence of the tracing aspect.

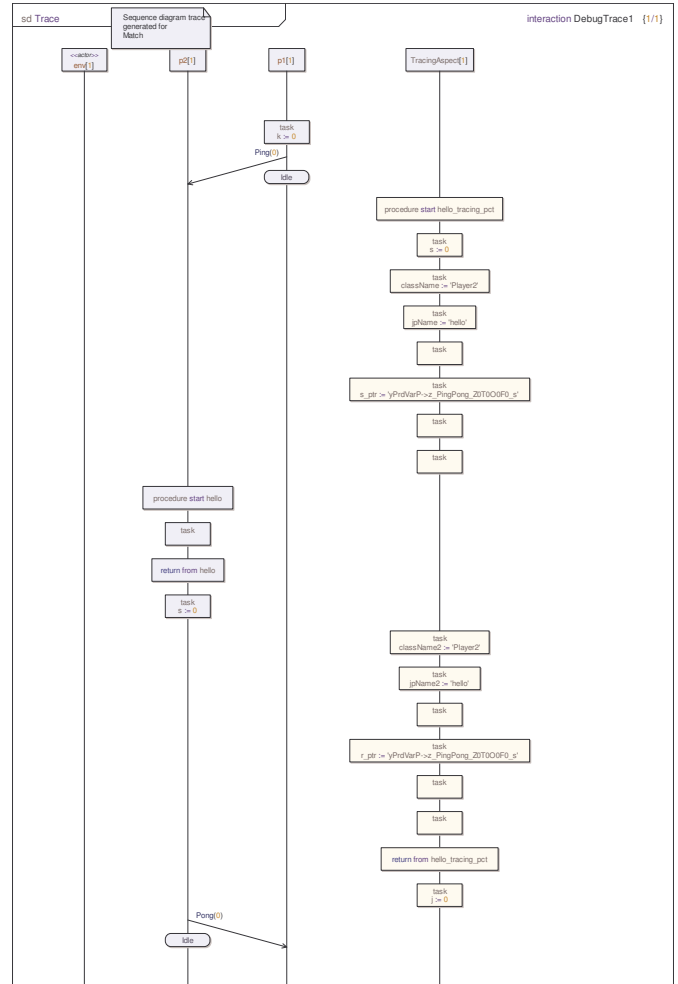


Figure 13. Aspect-Oriented decomposition of the sequence diagram generated by the model verifier

2.3.5. *Setting breakpoints on Connector Instances*

The simulation environment allows breakpoints to be set on aspect connectors and connector instances.

2.4. Model Weaving, Execution and Testing (15 min)

We discuss the full weaving of aspects at the modeling level, execution on the platform and testing of Aspect-Oriented models.

2.4.1. *Model Weaving*

The final weaving is performed right before code generation. Developers are never supposed to manually inspect a woven model. Various optimizations can therefore be performed while the presentation elements of the model can be disregarded

2.4.2. *Execution of Aspect-Oriented Models*

The execution of the code generated from an aspect-oriented model is demonstrated using a real work example

2.4.3. *Testing of Aspect-Oriented Models*

We discuss how aspect-orientation affects the test definitions in TTCN-3.

3. Examples and Hands-on Session (50 min)

This session will demonstrate the constructs and techniques presented in Session 2 through different examples.

3.1. Connecting to the Development Environment (5 min)

The attendees will have the opportunity to login on a server running the development environment. The tools and example models will be setup in advance to minimize the connection and setup procedures.

3.2. Overview of the Telelogic TAU Environment (5 min)

The user interface of Telelogic TAU will be briefly reviewed to familiarize the attendance with the development environment.

3.3. Model Simulation (20 min)

The attendance will be encouraged to browse through small, medium size and real (sanitized) models used in production, compile those models and run them in the simulator.

3.3.1. *Ping Pong Model*

This is a small model that illustrates key language feature.

3.3.2. *Simple Router Model*

The simple router model is a realistic implementation of a simple router. This model will be used in Section 3.4.2 in the context of timeout and quality of service concerns.

3.3.3. *Resource Access Server Model*

This model features a realistic implementation of an application server. It intends to show that these types of applications, which are typically written in Java, feature a lot of reactive behavior and can advantageously be implemented using modeling techniques.

3.3.4. *Subscriber Manager Model*

This model is sanitized version of a large model used in production. It intends to show that the techniques presented in the tutorial are scaleable.

3.4. Aspect-Oriented Model Simulation and Aspect Weaving (20 min)

The attendance will be encouraged to browse through the aspect definitions, deploy them and run Aspect-Oriented models in the simulator. The aspects will also be woven at the modeling level and the generated artifacts executed.

3.4.1. *Tracing Aspect*

This is a simple out of the box tracing aspect deployed in production at Motorola. It illustrates the different types of pointcuts supported by the *WEAVR*.

3.4.2. *Timeout Aspect*

This aspect encapsulates quality of service concerns. It can be deployed in any of the models used in section 3.3.

3.4.3. *Fault Tolerance Aspect*

This aspect *enforces* the specification of the two-phase commit protocol. It is intended to be deployed in the resource access server.

3.4.4. *Security Aspects*

These aspects perform authentication and encryption. They are intended to be deployed in the resource access server.

4. Discussion and Conclusions (10 min)

The focus of the discussion will be the new opportunities for Aspect-Oriented Software Development within a Model-Driven Engineering environment.

DURATION AND STYLE OF THE TUTORIAL

Half-day tutorial

The tutorial includes a 50-min demo and hands-on session.

LEVEL AND REQUIRED EXPERIENCE

Advanced

Familiarity with the UML notations and Aspect-Oriented Programming concepts is required.

EXPECTED AUDIENCE

This tutorial targets both an academic and industrial audience. One of the main goals of the tutorial is to bring awareness of the capabilities of state-of-the-art modeling and code generation tools among academia, discuss and demonstrate Aspect-Oriented Software Development technologies in this context.

SPEAKER'S PROFILE

Thomas Cottenier is a researcher for the Software and System Engineering Research Lab at Motorola Labs and is a PhD student at the Computer Science department of the Illinois Institute of Technology. Thomas holds an Electrical Engineering degree from the Université Libre de Bruxelles, Belgium and a MS in Computer Engineering from the Illinois Institute of Technology. Thomas' interests include Aspect-Oriented Software Development, Model-Driven Engineering and Service-Oriented Architectures. At Motorola, Thomas has been developing an industrial strength Model-Driven Engineering engine, the Motorola *WEAVR*, which combines Aspect-Oriented Modeling

and Automated Code Generation technologies. The engine is now reaching maturity and is being deployed in production within the Motorola Networks and Enterprise business unit. At the Illinois Institute of Technology, Thomas has been working on platform for defining adaptive and decentralized service compositions. He also worked on the modularization of Grid middleware concerns within the Globus Toolkit for Grid computing developed at Argonne National Lab.

SPEAKER'S PROFILE FOR THE ADVANCE PROGRAM

Thomas Cottenier is a researcher for the Software and System Engineering Research Lab at Motorola and is a PhD student at the Illinois Institute of Technology. Thomas' interests include Aspect-Oriented Software Development, Model-Driven Engineering and Service-Oriented Architectures.

TUTORIAL RESUME

This tutorial will be presented at the International Conference on Aspect-Oriented Software Development, in Vancouver, Canada on the 12th of March 2007.

Earlier versions of this tutorial have been presented to software engineers of the Motorola Networks and Enterprise Business Unit (about 60 developers). These sessions will be pursued and extended in 2007.

The tutorial has also been presented to a user group of graduate students at the Illinois Institute of Technology (about 10 students)

An early draft of the tutorial is available at:

<http://www.iit.edu/~concur/weavr/documentation.html>

The latest version used contains proprietary information and needs to be sanitized before it can be published.

EQUIPMENT

No particular equipment is needed for the presentation except for a LCD projector.

Attendees that wish to login on a server running the development environment need, in a addition to a Wifi connection:

- a SSH client such as Putty
<http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>
- a VNC client, such as RealVNC
<http://www.realvnc.com/cgi-bin/download.cgi>

PRESENTATIONS

External presentations on the *WEAVR* tool are available at:

<http://www.iit.edu/~concur/weavr/presentations.html>

REMARK

This material does not overlap with the series of tutorials presented by Markus Voelter on Model-Driven Software Development and Aspect-Oriented Software Development. Markus' material focuses on a particular generative programming technique that involves C++ template instantiation.