A PLATFORM INDEPENDENT FRAMEWORK FOR STATECHARTS CODE GENERATION

L. Andolfato, G. Chiozzi, ESO, Munich, Germany
N. Migliorini, ENDIF, Ferrara, Italy
C. Morales, UTFSM, Valparaiso, Chile.

Abstract

Control systems for telescopes and their instruments are reactive systems very well suited to be modelled using Statecharts formalism. The World Wide Web Consortium is working on a new standard called SCXML that specifies XML notation to describe Statecharts and provides a well defined operational semantic for run-time interpretation of the SCXML models. This paper presents a generic application framework for reactive non real-time systems based on interpreted Statecharts. The framework consists of a model to text transformation tool and an SCXML interpreter. The tool generates from UML state machine models the SCXML representation of the state machines as well as the application skeletons for the supported software platforms. An abstraction layer propagates the events from the middleware to the SCXML interpreter facilitating the support for different software platforms. This project benefits from the positive experience gained in several years of development of coordination and monitoring applications for the telescope control software domain using Model Driven Development technologies.

INTRODUCTION

Statecharts [1] have been successfully applied at the European Southern Observatory (ESO) for modelling reactive systems [2] such as telescopes and their instruments. Currently all ESO software platforms (the Very Large Telescope (VLT) [10], the Alma Common Software (ACS) [11], and the Standard Platform for Adaptive optics Real Time Applications (SPARTA) [12]), provide different application frameworks based on Statecharts to build monitoring and control applications. For the VLT platform a code generation framework integrated with Rational ROSE and MagicDraw has been developed and successfully employed to create more than 30 C++ control applications for different VLT/VLTI projects [6]. The SPARTA platform offers a C++ library to build Statecharts based applications. The ACS platform provides a tool to transform Statecharts into Java applications using Xpand template language [8].

Table 1: ESO Platforms (for non real-time applications)

Platform	OS	Languages	Middleware
VLT	Linux	C++, TCL/TK	CCS
ACS	Linux	C++, Java, Python	CORBA
SPARTA	Linux	C++	CORBA, DDS

These tools, although implementing the same formalism, support different sets of Statecharts features and, even for the commonly supported features, their operational semantics is often different due to lack of standard semantics for Statecharts [4]. In addition the UML models are not exchangeable between different generators because they are built using different UML profiles. Finally, for all three tools the generated/created source code depends directly on the state machine model and any change in the state machine logic requires a recompilation of the code.

In order to solve the above mentioned problems, reduce maintenance costs, and promote model reusability on different platforms (Table 1), the Generic State Machine Engine (GSME) project started at ESO with the objective of building a platform independent code generation tool from Statecharts called COMODO.

REQUIREMENTS

The requirements for COMODO were derived from the existing tools and can be summarized in:

- Support for the main Statecharts features
- Support for Statecharts inheritance
- Support for graphical and textual representation of Statecharts
- Support for multiple software platforms
- Support for Statecharts interpretation

Looking at the existing telescope control applications based on state machines, the most used Statecharts features are: composite and orthogonal states, shallow and deep history state, guards, entry/exit/on-transition actions, do-activities, and initial/final pseudo-states.

In addition to these standard features a requirement on model inheritance has been introduced to have the possibility of creating models that extend existing Statecharts (for example by adding states and/or transitions). Note that only “extension inheritance” is required, while “refinement inheritance” is not considered [3].

Statecharts is a graphical formalism and since it is part of the UML standard, any UML tool provides the possibility to create, edit, and visualize Statecharts models graphically. However, textual representation is useful for model comparison as well as for scenarios where UML modelling tools are not available. Therefore both graphical and textual Statecharts notations are mandatory for this project.

At last, the ability to change at run-time the behaviour of control applications becomes an important feature for:

- Reducing the model complexity by splitting a large model into several smaller ones. Consider for example the case where a control software application has to support different type of HW components that can be modified at run-time. One possible implementation is to include all HW configurations in one large Statecharts model. A second solution is to have one model per HW configuration and load the correct model at run-time. The latter is usually a better choice in terms of maintainability since smaller Statecharts models are easier to understand.
- Reducing the time needed to apply changes to the state machine logic. For example, during testing/deployment of an instrument, a large amount of time is dedicated to tuning the calibration procedures (i.e. to define in which order to move the hardware and acquire the data). Changing the behaviour of the calibration procedure without the need of recompiling speeds up considerably the testing/deployment phases.

Of course, interpreted Statecharts do not provide the same performances of compiled Statecharts and often require more memory. However, our target applications are monitoring and control applications running on workstations that do not require real-time reaction time.

STATECHART XML

COMODO is based on a StateChart XML (SCXML) engine and a model to text (M2T) transformation tool. Depending on the given software platform configuration, it transforms UML State Machine models into SCXML models and platform specific artefacts such as the application code needed to integrate the SCXML engine and the manually developed code (Figure 1).

SCXML [5] is a standard introduced by the World Wide Web Consortium (W3C) to describe Statecharts. The syntax is based on XML textual notation, and the operational semantics is well defined via pseudo-code. Furthermore, the SCXML language has been designed to be interpreted so that the dynamic behaviour of an SCXML application can be modified at run-time.

SCXML supports all standard Statecharts features required by the project. Moreover, it offers a possible implementation of the Statecharts inheritance via the XML inclusion mechanism.

The support for different ESO platforms is achieved by providing C++ and Java SCXML engine libraries that can be compiled on each of these platforms.

COMODO PROFILE FOR UML

A COMODO application is modelled with a subset of UML which consists of the state machine to specify its behaviour and some elements to describe the interface and its deployment. Applying domain specific stereotypes to the model elements, allows to customize and re-use the model for the different supported software platforms. Stereotypes were collected using the UML profile mechanism in the COMODO profile [13].

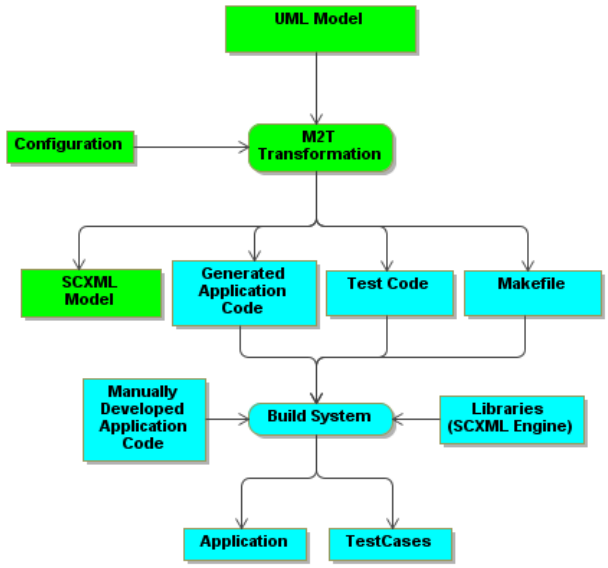

Figure 1: COMODO data flow: in green platform independent activities and artefacts, in blue the platform dependent ones.

MODEL TO TEXT TRANSFORMATIONS

The process to transform a UML model into the target application for a given platform consists of two parts:

- Transformation from UML model to SCXML
- Transformation from UML model to platform specific artefacts

The transformations are based on the mapping between the source model element and the generated artefact.

UML to SCXML Mapping

In order to transform UML Statecharts to SCXML, a mapping between UML model elements and SCXML syntax notation has been defined (Table 2).

The SCXML model is platform independent and even the implementation of entry/exit/on-transition actions and do-activities can be embedded in the model using SCXML scripting action language (or another scripting language supported by the target platform). However our target applications are normally developed in C++ or Java and therefore the interpretation has been limited to the state machine logic while actions and activities are compiled. This approach allows changing the state machine logic at run-time while leaving the advantages of compiled languages for the implementation of actions and activities.

Table 2: UML to SCXML Mapping

UML	SCXML
State Machine	<scxml>
State	<state>
Composite state	<state> (the initial sub-state must be defined)
Region	<parallel> (the initial sub-state must be defined)
Initial pseudo-state	<initial> or initial attribute of <state>
Final pseudo-state	<final>
Entry / Exit action	<onentry> / <onexit>
Transition (trigger [guardExpression] / action)	<transition event="trigger" guard="guardExpression">
Internal transition	<transition> with no target specified
Deep History	<history type="deep">
Shallow History	<history type="shallow">
Activities	<invoke>
Actions	Custom actions
Event (signal)	String
Timer Event	Send command with timeout

UML to Platform Specific Artefacts Mapping

The platform specific artefacts generated from the UML model are:

- Action and Activities stubs (generated only once the very first time)
- The code to handle the platform specific events and propagate them to the SCXML engine
- All artefacts to build the generated application
- Some basic test code used by the automatic test infrastructure provided by the software platforms

Each action and activity defined in the Statecharts model is mapped to a C++/Java class. Activities are started in separate threads while actions are executed via method invocation. It is foreseen to have also an action-to-method mapping (i.e. mapping of a group of actions to methods of a class) to avoid proliferation of classes, facilitate data sharing among actions, and reduce compilation time.

SCXML defines events as simple strings therefore a translation of platform specific events to the SCXML events has to be specified. For example a CORBA call has to be mapped to the corresponding SCXML event string and injected in the SCXML engine. Table 3 shows different types of events supported by the software platforms and their representation in UML.

Table 3: Events Mapping

UML	VLT	ACS	SPARTA
Signal	Command/Reply callback	CORBA method	CORBA method
Time event	VLT Timer callback	ACS Timer	Timer
Signal <<attribute>>	Database change notification callback	CORBA attribute change	CORBA attribute change
Signal <<fileio>>	UNIX file I/O event.	UNIX file I/O event.	UNIX file I/O event.
Signal <<internal>>	Internal event	Internal event	Internal event

IMPLEMENTATION

The UML to SCXML and UML to platform specific artefacts mappings are implemented using EMF [7] and Xpand tool set [8] which include a workflow executor (MWE), the Check constraint language, the Xpand template language, and the Xtend language. The code generator structure is shown in Figure 2.

The MWE workflow is used to drive different steps of the model to text transformations. Four workflows have been created: one per platform and one for the SCXML transformation. The steps of the workflow are:

- Load the UML Profile
- Load the model
- Validate the model by applying Check rules
- Run Xpand templates to generate artefacts

The step validating the model is important since normally UML tools do not impose any rule in the specification of Statecharts while SCXML, being an operational specification, requires well formed models. For this reason a set of Check rules have been written to verify that the model complies with SCXML specifications (for example that the initial state in composite states is always specified or that a transition from the history state is provided).

After the model has been validated, Xpand templates are executed to generate the artefacts defined in the mapping. A set of functions have been developed in Xtend language to help navigating the model to retrieve the model elements properties.

Note that veto strategy or round-trip code generation is not supported by the tool. Customization of the generated code is done via subclassing. Implementation of actions and activities is therefore performed by inheritance from abstract classes generated by the tool.

GENERATED APPLICATION

The applications generated by COMODO use an SCXML engine to process incoming events and trigger transitions.

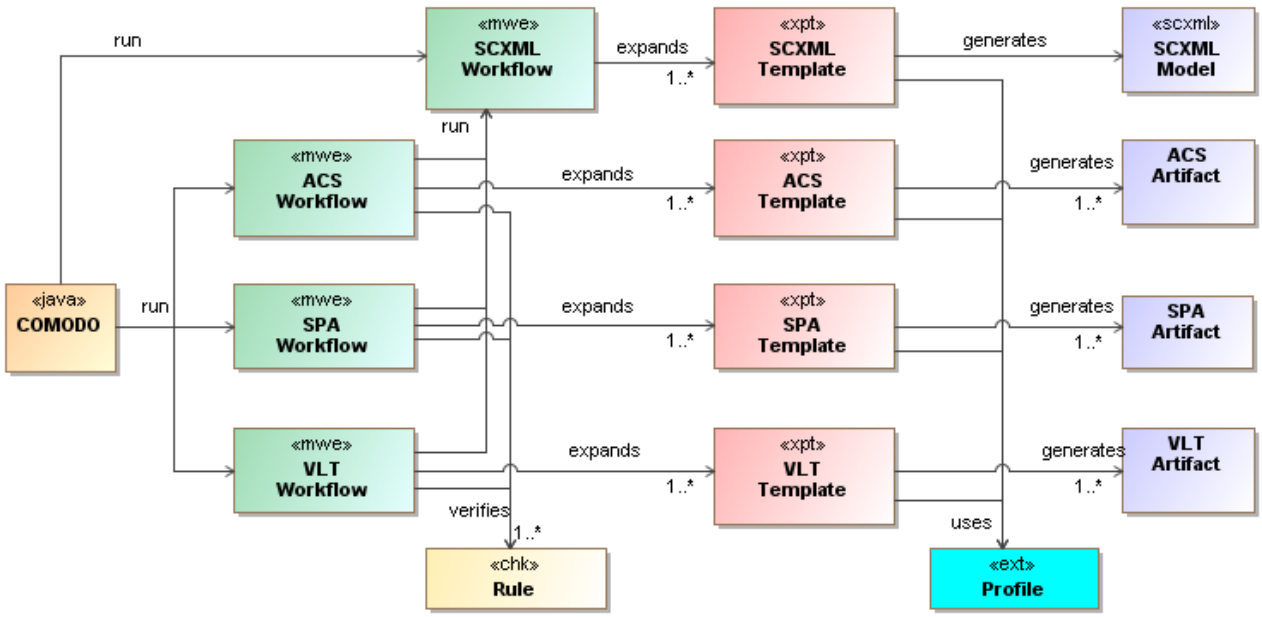

Figure 2: Code generator structure.

Apache Commons SCXML engine [9] has been selected to be the SCXML engine for Java applications. A C++ simplified prototype of the Apache Commons SCXML engine is being developed in house for the C++ applications.

Events are propagated to the SCXML engine via platform specific adapter classes generated by the tool. Actions and activities are executed by the SCXML engine using invoker classes, also provided by the tool. Access to platform services such as logging is provided to actions and activities via injection at creation time.

CONCLUSIONS

In this paper a cross-platform code generator tool from Statecharts was presented. The tool relies on the SCXML notation which provides a standard syntax and semantic specification to describe and execute Statecharts. Since SCXML models are interpreted, generated applications can change behaviour by modifying the state machine logic at run-time without the need for recompilation, reducing therefore the development time.

A UML to SCXML mapping is proposed to benefit from the graphical nature of Statecharts models.

The project is still at prototype level but it provides a complete UML to SCXML transformation and the generation of Java applications for the ACS platform. The next important step is to develop a C++ SCXML engine that is needed to support the other two software platforms.

REFERENCES

[1] D. Harel, "Statecharts: A visual formalism for complex systems", Journal Science of Computer Programming, vol. 8, issue 3, pp. 231-274 (1987).

[2] D. Harel, M. Politi, "Modeling Reactive Systems with Statecharts", McGraw-Hill, (1998).

[3] M. Stumptner, et all, "Behavior Consistent Inheritance with UML Statecharts", Proc. ECOOP'04, p. 59 (2004).

[4] M. L. Crane, J Dingel, "UML vs. classical vs. rhapsody statecharts: not all models are created equal", Software and Systems Modeling, vol. 6, issue 4, pp. 415-435 (2007).

[5] "State Chart XML (SCXML): State Machine Notation for Control Abstraction", W3C Working Draft, April 2011, (2011).

[6] L. Andolfato, R. Karban, "Workstation Software Framework", Proc. SPIE 2008, vol. 7019, (2008).

[7] D. Steinberg et al., "EMF: Eclipse Modeling Framework", 2nd Edition, Addison-Wesley Professional, (2008).

[8] B. Klatt, "Xpand: A Closer Look at the model2text Tranformation Language" 12th European Conference on Software Maintenance and Reengineering, (2008).

[9] Apache Commons SCXML; http://commons.apache.org/scxml/.

[10] K. Wirenstrand, "VLT telescope control software: status, development, and lessons learned", Proc. SPIE 2003, vol. 4837, p. 965 (2003).

[11] G. Chiozzi et all, "ALMA Common Software (ACS), status and development", Proc. ICALEPCS 2009, (2009).

[12] E. Fedrigo et all, "SPARTA: the ESO standard platform for adaptive optics real time applications", Proc. SPIE 2006, vol. 6272, (2006).

[13] G. Chiozzi et all, "A UML profile for code generation of component based distributed systems", this conference, (2011).