Reusable state machine code generator

ARTURO A. HOFFSTADT[1], HEIKO SOMMER[2], LUIGI ANDOLFATO[2], CECILIA REYES[1]
[1]Computer Systems Research Group, Universidad Técnica Federico Santa María (UTFSM), Valparaíso, Chile
[2]European Southern Observatory, Garching b. München, Germany

ABSTRACT

The State Machine model is frequently used to represent the behaviour of a system, allowing one to express and execute this behaviour in a deterministic way. A graphical representation such as a UML State Chart diagram tames the complexity of the system, thus facilitating changes to the model and communication between developers and domain experts.

We present a reusable state machine code generator, developed by the Universidad Técnica Federico Santa María and the European Southern Observatory. The generator itself is based on the open source project architecture, and uses UML State Chart models as input. This allows for a modular design and a clean separation between generator and generated code. The generated state machine code has well-defined interfaces that are independent of the implementation artefacts such as the middle-ware. This allows using the generator in the substantially different observatory software of the Atacama Large Millimeter Array and the ESO Very Large Telescope. A project-specific mapping layer for event and transition notification connects the state machine code to its environment, which can be the Common Software of these projects, or any other project.

This approach even allows to automatically create tests for a generated state machine, using techniques from software testing, such as path-coverage.

Problem

Description

Design and specification of discreet events involved in software, is a complex task for software architects and developers. Several visual formalisms can tackle this problem, and allows to specify the workflow without much trouble. But when software reach a medium size, UML sequence diagrams explodes in complexity, and the straightforwardness of its representation is lost in confusion.

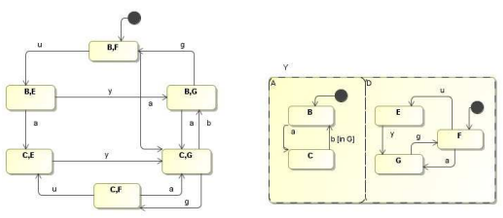
State machines has the same problem, that is why Harel, in [1], proposes *State charts*, an extension to the visual formalism for state machine, that allows to reduce visual complexity, and keeps the mathematical determinism of state machines. This new formalism was later standarised and included as part of UML2.0.

In other front, state machines entities (understanding entity as a component of the specification of a language) does not map directly to program language entities. There are several libraries, like *boost* that currently have state machine implementations. So, to allow model driven development using state charts models, a special transformation and code generation has to be implemented.

Reusability

There is another important problem that this project attacks. Reusability. ALMA Common Software aims to be a generic distributed control platform, and many applications has the same problems described before, not only in astronomy, but also in other areas. Speaking in a more specific manner, re implementing the state machine logic through several ALMA projects is far from ideal, and in the case of VLT Common Software, several applications has been created using their own state machine code generation framework [4].

This project aims to solve this problems, and integrate the better qualities of each ACS and VLT implementations of code generation.



Example presented in [1] by Harel, showing the visual complexity reduction of Statecharts.

State Machine Code Generator

Or SMCG, is a reusable state machine code generator. It is based on openArchitectureWare (oAW), and uses UML2 XMI 2.1 state chart models as input, to generate compilable and executable code. The current implementation is based on work from Niaz and Tanaka ([2] and [3]), a previous implementation of code generator from Heiko Sommer created for the ALMA project, and the *WorkStation Framework* from VLT Common Software [4].

The generator is primary based on oAW. Through its workflow engine, the code generator is configured, setting input model, output code directory, and parameters such as language used for code output (Java or C++), and also which framework used as event engine (VLT or ACS). Templates for code generation were created keeping in mind that new languages or engines can be added to this project, so further capabilities can be added.

Workflow

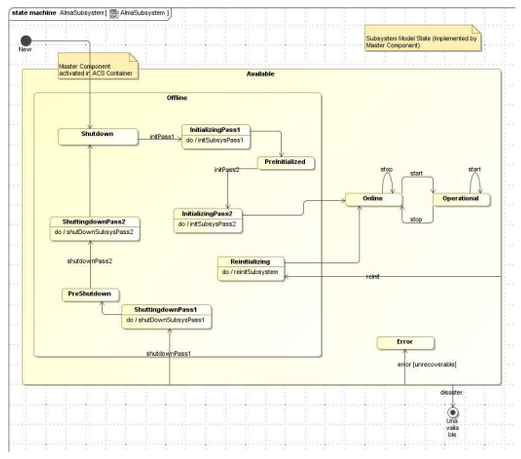
1. Generation starts with the indication of the input model, in UML2 XMI 2.1 format.
2. Several checks are ran over the model, to ensure compatibility, such as naming convention (presence and uniqueness of names), warn about missing or experimental features, and so on.
3. Later, a model-to-model mapping takes place, mapping the specific instance of UML2 statechart used as input, to the UML2 statechart metamodel.
4. Next step is to generate the engine to allow execution of the state machine. This engine allows event handling, logging, and control of the state machine. Engines are platform dependant.
5. Continuing the workflow, is the code generation for the state machine, which creates the logic for the state machine. This implementation is only language dependant. Specific interfaces allow changes in engine or state machine logic.
6. Last step, is creation of tests. This tests are preset, and provides a minimum set for testing known cases.

Using State Machine Code Generator

First task, is to model the system as a finite state machine. States has to clearly defined, with no overlying functions, and transitions needs to have clear events, and guards is necessary. Names for states, transitions and events has to be defined, and to be unique. Orthogonal regions are supported, as also composite states. Every composite state needs to have an initial node. Also, at least one final node has to be determined. Names in this cases are not needed.

Once the logic of the software has been created, is time to configure SMCG. Configuration is done in the main workflow file. Selection of framework, language and input model, are necessary. Execution of workflow file, will start code generation, and then the specified workflow will take into place, as in the previous section.

The recommended approach to create software in SMCG, is model driven development, but other approaches can be used.



Examples of Usage: Model of an ALMA Subsystem's lifecycle.

Architecture

As this is a code generator, two aspects of architecture were analysed. First, is the architecture of the generator itself, and second but not less important, is the architecture of the generated code.

Code Generator

The generator has a flexible structure, based mainly on the openArchitectureWare's workflow file, and templates. Configurations are taken at this time directly from variables set in the workflow file. Selection of framework and language are done in this manner. State machine implementation, regardless of framework or language, is platform independent, so this can be reused, even to the degree of having no underlying framework. A mapping layer in the state machine engine allows to easily change from ACS to any other middleware. ACS mapping layer is complete, VLT CS is under development.

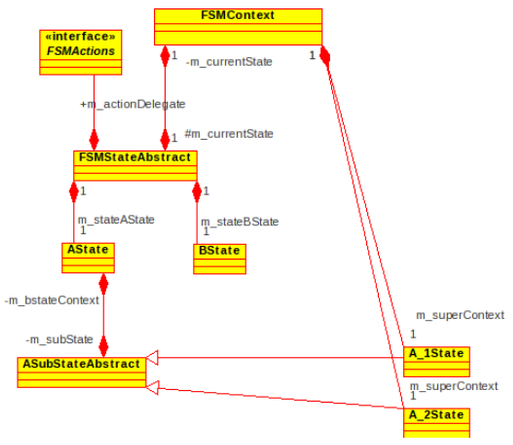
State Machine Code

The generated code architecture is based on [1] and [2], which propose that each state is represented by a class, using the state pattern. Transitions are represented by method in the corresponding state, and actions are implemented in a separate context class. Delegation is used to process events in the corresponding state object. To implement hierarchy (composite state) composition was used. This composition is defined in runtime, during state construction.

Some advantages of this implementation are the lower number of generated classes, and that only one file is needed to be modified: the context class, in which the actions are implemented.
Current implementation supports:

- Composite States ("or" states)
- Orthogonal States ("and" states)
- Guards
- Actions
- Activities
- Events
- Java Implementation
- ACS Framework

Next version, to be released and the end on November 2009, will include support for VLT CS, C++ Implementation, and other functionalities. Further work will be conducted by the CSRG group at UTFSM, and ESO, to enhance and add more characteristics.



Generated Code Architecture.

Conclusions and Future Work

Code generation enhances reusability, and model drivel development allows expert domains to have a better grasp on software specifics. UML State chart has been a great addition to UML, standardised this visual formalism, and also puts at hands of developers, another powerful tool for software design.

Development in this project continues. C++ implementation will be available for next release, but most important, is the current state charts specification that W3C is writing [5], that allows to specify a complete state machine, including code for actions, and other new characteristics, such as inheritance. Further work will concentrate on integration of the current available state chart engines that Apache and Qt Software have implemented, and the adaption of the code generator to accept SCXML specification of state machines as input.

Acknowledgements

This work was supported by ALMA-CONICYT Fund project #31060008 *"Software Development for ALMA: Building Up Expertise to Meet ALMA Software Requirements within a Chilean University"*, under development at Universidad Técnica Federico Santa María.

References

[1] David HAREL, "Statecharts: A Visual Formalism for Complex Systems" in [*Science of Computer Programming Vol. 8, No. 3*], (June 1987), pp. 231-274.
[2] I.A. Niaz and J. Tanaka, "Code Generation from UML Statecharts", in [*Proceeding (397) Software Engineering and Applications*], 2003.
[3] I.A. Niaz and J. Tanaka, "Mapping UML Statecharts to Java Code", in [*Proceeding (418) Software Engineering*], 2004.
[4] L. Andolfato, "Very Large Telescope Workstation Software Framework Design Description", VLT-MAN-ESO-17210-3389, Mar. 2006.
[5] J Barnett et. al., "State Chart XML (SCXML): State machine notation for control abstraction ", W3C Working Draft, 2007

Atacama Large Millimeter Array

Contact e-mail: ahoffsta@inf.utfsm.cl / For more information about ALMA development and research at UTFSM Computer Systems Research Group, please visit our web site http://alma.inf.utfsm.cl