SOFTWARE SYNTHESIS FOR EMBEDDED SYSTEMS USING EXECUTABLE AND TRANSLATABLE UML

Caraulean Valeriu

Technical University of Moldova, Faculty of Radioelectronics and Telecommunication

Email: caraulean@gmail.com

Secrieru Nicolae

Technical University of Moldova, Faculty of Radioelectronics and Telecommunication

Email: secrieru@rgg.md

Abstract: In this paper are described Executable and Translatable UML and how it can be used in developing of Embedded Systems. Also, is evidentiated all benefits offered by automatic translation of UML models in source-code and verification of models.

Keywords: UML, XTUML, application model, software design, model translator.

INTRODUCTION

Synthesis is the process of taking a high-level description and turning it into a lower-level description that, in the case of software, can be compiled directly. Synthesis involves usage of Automatic Code Generation (ACG).

ACG tools based on the Unified Modeling Language (UML) allow programmers to create control and calculation programs graphically by using boxes to represent input, output, and processing algorithms [1, 5, 6]. Some UML tools generate code based on these diagrams. But, standard features and abilities of UML language is not enough for description of objects which can be translated in source code ready to be compiled for target. Especially for this, was created a subset of UML language - Executable and Translatable UML (XTUML), which separates models from design. It offers possibility to test the model before you have a target, then generate an optimal target-specific design.

EXECUTABLE AND TRANSLATABLE UML

XTUML is a subset of the UML endowed with rules for execution. With an executable model, you can formally test the model before making decisions about implementation

technologies, and with a translatable model, you can retarget your model to new implementation technologies.

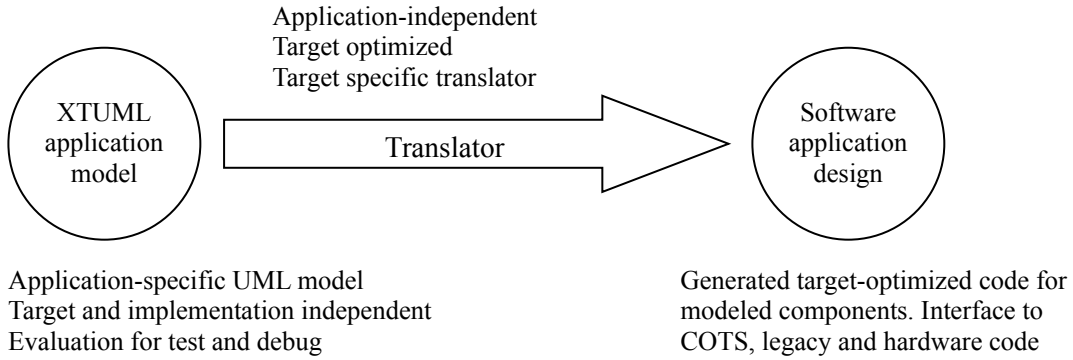

Figure 1: Separation of application models and software architecture

XTUML separates application models from software architecture design, weaving them together through a translator at deployment time, as shown in Figure 1. There are three components of an XTUML design:

- *Application models* capture what the application does. The models are executable, which enables you to validate that your application meets requirements early on. Application models are independent of design and implementation technologies.
- *Software architecture designs*, defined as design patterns, design rules, and implementation technologies, are incorporated by a translator that generates code for the target system. The software architecture designs are independent of the applications they support.
- The *translator* applies the design patterns to the application models according to the appropriate design rules.

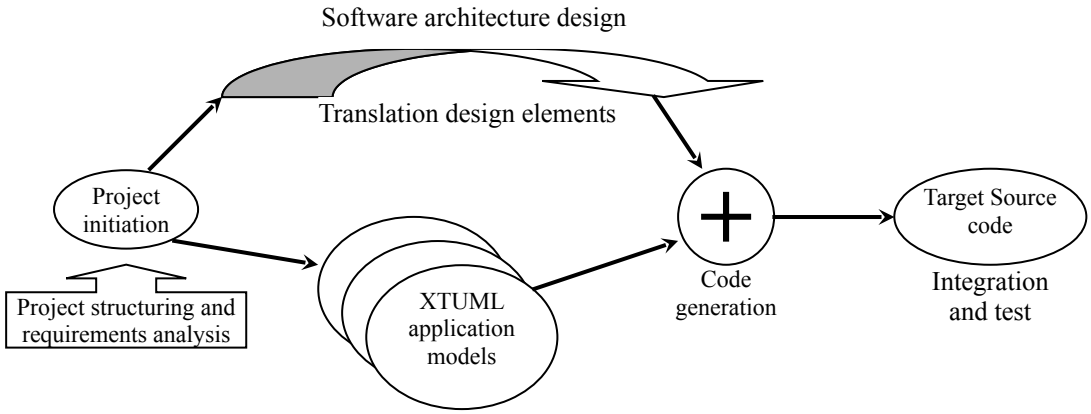

Figure 2: Concurrent design and modeling

The separation of the software architecture design from the application models supports concurrent design and application analysis modeling, as illustrated in Figure 2. Using XTUML, you can iteratively and incrementally construct both the application and the software architecture design.

UML IN EXECUTION

XTUML incorporates well-defined execution semantics. Objects execute concurrently, and every object is in exactly one state at a time. An object synchronizes its behavior with another object by sending a signal interpreted by the receiver's state machine as an event. When the event causes a transition in the receiver, the procedure in the destination state of the receiver executes after the action that sent the signal, thus capturing the desired "cause and effect" in the system's behavior.

The application model therefore contains the decisions necessary to support execution, verification, and validation, independent of design and implementation. No design decisions need be made nor code developed or added for model execution, so formal test cases can be executed against the model to verify that application requirements have been properly addressed. At system construction time, the conceptual objects are mapped to threads and processors.

TRANSLATION

Translators generate code from models automatically. The translator (Figure 1 again) is made up of three elements:

- A set of design patterns ("archetypes") to be applied in code generation together with rules for when a given archetype or model component will be used to build code.
- A translation engine that extracts application model information and applies the archetypes and rules to generate complete code.
- A run-time library comprising pre-compiled routines that support the generated code modules.

When generating code, the translator extracts information from the application model, then selects the appropriate archetype for the to-be-translated model element. The result is a coded implementation component. This approach is excellent for real-life applications [3,4].

AUTOMATION

XTUML cries out for automated support for execution and translation.

Capabilities provided by XTUML automation include:

- rapid project ramp-up resulting from a streamlined UML subset and a well-defined process
- concurrent application analysis and design to compress project schedules modeling
- reduced defect rates from early execution of target-independent application models and test of application-independent designs

- customizable translation generating complete, target-optimized code
- performance tuning and resource optimization
- effective, practical reuse of target-independent application models and application-independent designs
- reduced maintenance costs and extended product lifetimes

CONCLUSION

This approach was used in various embedded systems implemented by RGG s.r.l., such as Thermo and Electro Power Stations [4]. Design patterns [3] and XTUML modeling demonstrates his efficiency and this approach accelerates development and improves the quality, performance, and resource use of real-time embedded systems.

REFERENCES:

[1] Booch G, *Object-Oriented Analysis and Design with Applications.* Addison-Wesley, 1993.
[2] Fowler M. and Kendall S.. *UML Distilled: Applying the Standard Object Modelling Language.* Addison-Wesley-Longman, 1997.
[3] Caraulean V., *Fault Tolerance Techniques For Distributed Systems*. – In: Proceedings of SIELMEN 2003.
[4] Caraulean V., Secrieru N., *Designing Distributed Control System For Thermo And Electropower Stations Using Statecharts*. UTM, Anniversary Conference, 2004, Vol. 2
[5] Secrieru N., Caraulean V., *UML Designing Distributed Microcontrollers Network*. – In: Proceedings of ICMCS 2002, Vol. 2
[6] *UML Notation Guide* Version 1.1 , September, 1997. Rational Corp.