Automating Design and Verification of Embedded Systems Using Metamodelling and Code Generation Techniques

What is Metamodelling and Code Generation All About

Wolfgang Ecker & Michael Velten, Infineon
Automating Design and Verification of Embedded Systems Using Metamodeling and Code Generation Techniques

Wolfgang Ecker, Michael Velten - Infineon Technologies AG
Rainer Findenig - Intel Corp.
Daniel Müller-Gritschneider - Technical University of Munich
Wolfgang Mueller – Heinz-Nixdorf Institut University of Paderborn
What is Metamodelling and Code Generation All About

- Motivation, Technology and Terminology

Well known Metamodels in EDA and Design

- UML/SysML
- IP-XACT

Coffee Break

Metamodelling in Action using Eclipse Modeling Framework

- Generate a code generation framework for IP-XACT
- Specify an IP-XACT model and generate code out of it
Motivation for Using Meta-Modeling and Code Generation

Infineon Designers on Single Design Tasks
- Up to 95% reduction in SW header generation
- Savings of about 1PY / year through test file generation
- Savings of about 4-5PYs / year through efficient solutions handling test programs

Infineon Designers on Full Chip Implementation
- 60% effort reduction and 2 months project time savings from specification to implementation
- 80% code of digital design part generated

MetaCase
- Up-to 20x speed and productivity improvement in using MetaEdit (A Metamodelling Framework) for SW Development
A Well Known Scenario:
Scripts Supporting Design Productivity

Problems: Starts easy, gets more and more complex (and harder to maintain) and ends up in spaghetti-code due to …

- increasing requirements,
- more output formats and alternatives, and
- more complicated import formats

The good aspect: Content is automatically copied, code is generated, and nothing is retyped
Model-View Separation is a good, well-known and powerful SW Concept (Pattern)

MyScript is separated in 3 pieces

- An **API** that controls access to structured data called **Model**
- A **Reader** that takes abstract data and fills the model
- A **Writer** that extracts data from and generates code
**1st Improvement:**
Model-View Separation

**Benefit**

- New and more complicated input and output formats can be supported by local changes
- Existing parts can be used further on
Tasks of a reader:
- Parse a description that is more abstract than the target code (e.g. specification items, domain specific languages)

Building blocks of a reader:
- Libraries as XML Readers, document readers as MS-Office or OpenOffice readers, PDF-parser, ...
- HDL Readers (Verific), compilers with open API (e.g. clang)
- Generated Parsers (e.g. via ANTLR) ...
Model-View Separation
Writer

Different approaches to implement writers:

- **Sequence of prints, each taking values from the model**
  
  ```python
  print("entity %s is\n", model.name);
  ```

- **Systematic model traversal (mostly breath first or depth first) and registration of prints as actions when entering/leaving a node**

- **Template Engines, e.g.: FreeMarker (Java, EMF), Mako (Python, used by IFX), xsd:template as part of XSLT**
Model-View Separation
Templates (Mako)

render directive

library IEEE;
use IEEE.std_logic_1164.all;

entity ${component.getName()} is
    port(
    % for port in component.getPorts()
        :{port.getName()} :...
    % endfor
    )
end ${component.getName()};

target code

library IEEE;
use IEEE.std_logic_1164.all;

entity LU is
    port(
        A : in std_ulogic_vector (15 downto 0);
        L : in std_ulogic_vector ( 1 downto 0);
        Y : out std_ulogic_vector (15 downto 0))
end LU;
A Template Engine translates visible or under the hood a template to a writer and then controls execution of the writer.
2nd Improvement: Generation of Tool’s Code from Metamodel

Structure definition by Metamodel:
- Reader / Writer has to comply to Metamodel’s structure and types
- API can be generated
- API generator offers to be structured similarly:
  - Reader, API (Model), Writer
What is a simple Metamodel composed of

- Composite Data
  - Typed or Un-Typed Attributes
  - Typed or Un-Typed Children
  - Typed or Un-Typed Links
- Optional multiplicity or other constraints

There are several techniques out that support Metamodeling and Code Generation. Examples are:

- XML with XSD (XML Schema)
- UML based on (E)MOF
- EMF based on (E)CORE
- METAGEN based on MMANALYZE (IFX-proprietary)

The elements of a Metamodel are defined in a so called Meta-Metamodel (we will see its usefulness later)
Metamodelling Technology: **Modeling** Is About Structuring and Formalizing Things

- **car**
  - Type: BMW
  - Name: i6
  - Voltage: 24

- **electronics**
  - Version: 1.2.3
  - Voltage: 24

- **battery**
  - Capacity: 200
  - Voltage: 24

- **wheel**
  - Position: Right
  - Type: 190

- **pressure sensor**
  - Provider: IFX

- **chip**
  - Name: SP37
Metamodeling Technology: Metamodeling is about structuring and formalizing models.

**car**
- Type: string[1]
- Name: string [1]

**wheel**
- Position: positionEnum[1]
- Type: int[1]

**pressure sensor**
- Provider: string [1]

**electronics**
- Version: 1.2.3
- Voltage: 24

**battery**
- Capacity: 200
- Voltage: 24

**chip**
- Name: SP37

**car**
- Type: BMW
- Name: i6
Metamodelling Technology: Metamodelling is about structuring and formalizing models.

Elements of a Metamodel:
- Compositions
- Typed Attributes
- Typed Children
- Multiplicity constraints
- Other constraints (not shown)
Some Known Metamodels
UML and IP-XACT

- Graphical formalism (primarily) to describe/model SW Systems
  - Formalisms describe structure, behavior and interaction
  - Examples are class diagrams, object diagrams, state diagrams, activity diagrams
  - UML is based on a superstructure (MOF, EMOF) that defines the formalism
  - OCL (object constraint language) is used to defined further constraints
- Stereotypes as [ML][Marte] support embedded systems

IP-XACT
- Defines data that support automation in IP-integration. Includes
  - Busses, components with their registers, connectivity
- Does not model IP-Internals
Metamodeling Technique
Additional Features of a Core Model

Wide range of products (IFX Examples Shown) require flexibility in Metamodelling

- Extendibility
- Constraints
- Interaction
- Composition

Power/Analog
MEMS/Sensors
CMOS
RF/Bipolar
Examples

- Analog types and their properties
- Register protection mechanisms
- Clocked State Diagrams

Constructs for extendibility in different notations

- Supported e.g. by inheritance in core model
- UML uses profiles or OCL
- XML provides restrictions and complex datatypes
Metamodelling Technique
Interaction and Composition

Examples
- Registers or State diagrams manipulate ports
- Constructs for extendibility
  - Link mechanism e.g. XML XPATH
  - Model-to-Model translation
Metamodeling Technology
Layers in Structuring Data

- **Meta-Meta Model**
  - (meta-meta model)
  - Defines Structure of Metamodel

- **Metamodel**
  - Defines Structure of Model
  - Generate Metamodel Infrastructure and Metamodels

- **Model**
  - Defines content of view language independently
  - Generate Model Infrastructure Models

- **View**
  - Generate View(s)
  - Implementation of content
Meta-Metamodel: Is About Structuring Metamodels, i.e. Metamodel of Metamodel
Meta-Metamodel: Is About Structuring Metamodels, i.e. Metamodel of Metamodel

Shown in 3rd part of the tutorial building an IP-XACT to target code translation
It’s All About Structuring
Summary and Retrospect

All is not new! Metamodelling has a >25-year history

- Formally called Express Information Model
- Further developed in Jessi Common Framework Initiative (CFI)
- Formal foundation for EDIF (Electronic Design Interchange Format)
Metamodeling and Code generation is

- an industry proven technology to efficiently build domain/problem specific tools following a specific structure

Modeling in the context of Metamodeling is about

- structuring things in a design context

Metamodeling is about

- Structuring Models
Thank you!