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

What is Metamodeling and Code Generation All About

Wolfgang Ecker & Michael Velten, Infineon

SYSTEMS INITIATIVE

Tutorial

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-Gritschneder *- Technical University of Munich*
Wolfgang Mueller *– Heinz-Nixdorf Institut University of Paderborn*

Outline and Schedule

What is Metamodeling and Code Generation All About

- Motivation, Technology and Terminology

Well known Metamodels in EDA and Design

- UML/SysML
- IP-XACT

Coffee Break

Metamodeling 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

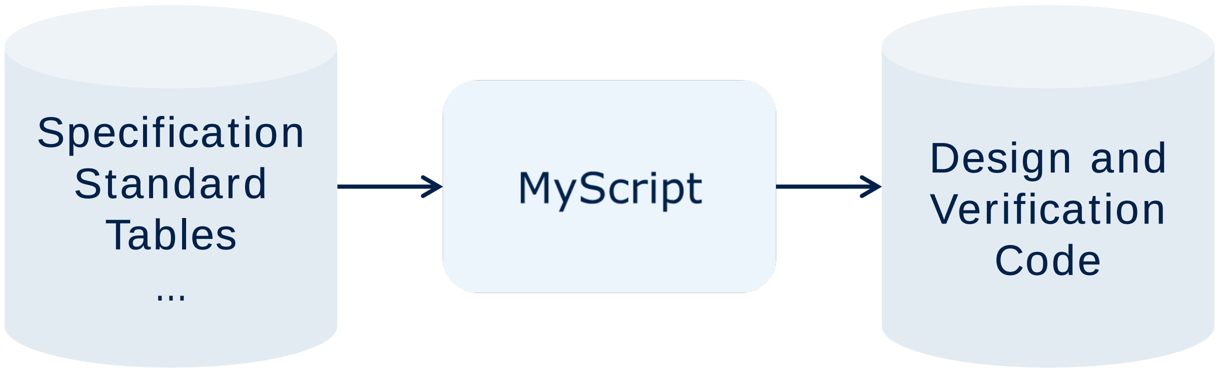
Infineon Designers on Full Chip Implementation

- 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 Metamodeling Framework) for SW Development

A Well Known Scenario:
Scripts Supporting Design Productivity

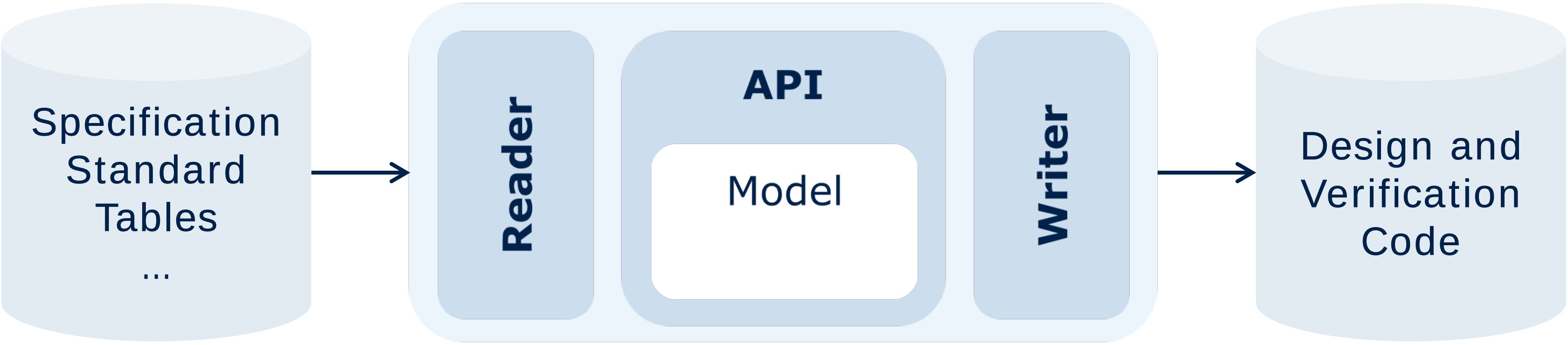


<u>Problems:</u> Starts easy, gets more and more complex (and harder to maintain) and ends up in spaghetti-code due to ...

- 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

1st Improvement:
Model-View Separation

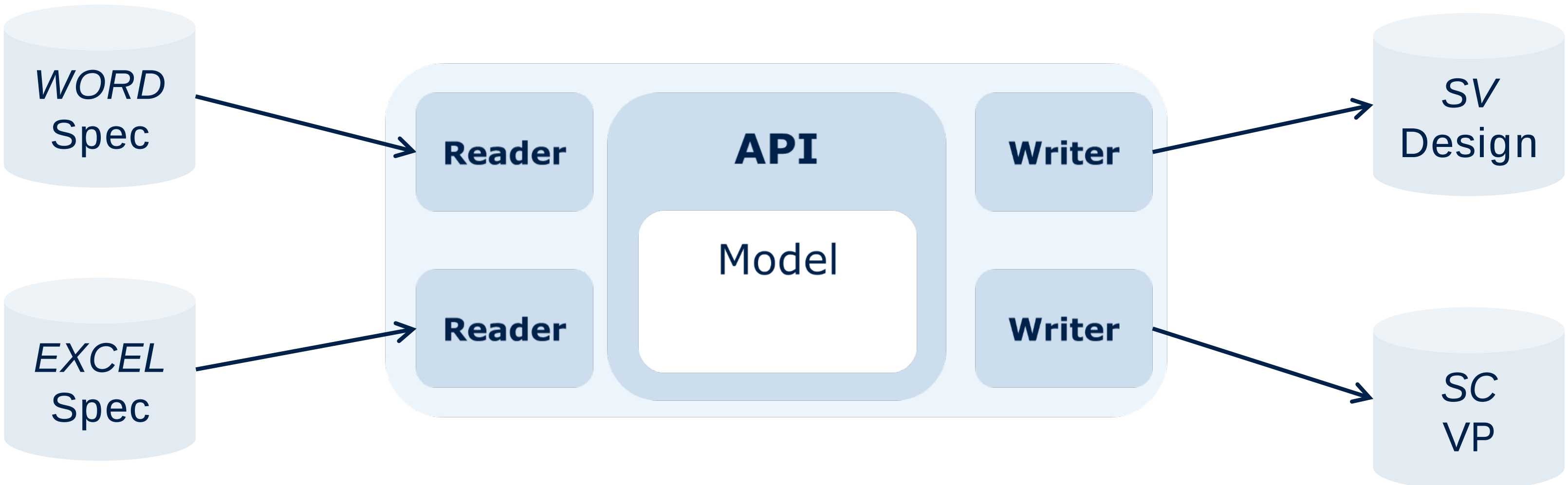


Model-View Separation is a good, well-known and powerful SW Concept (Pattern)

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

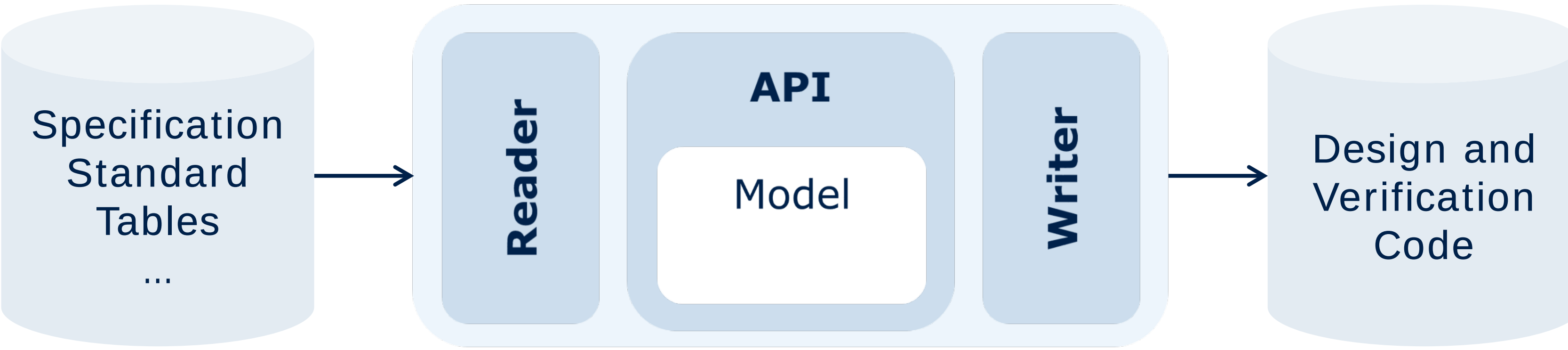
1st Improvement:
Model-View Separation



Benefit

- New and more complicated input and output formats can be supported by local changes
- Existing parts can be used further on

Model-View Separation
Reader



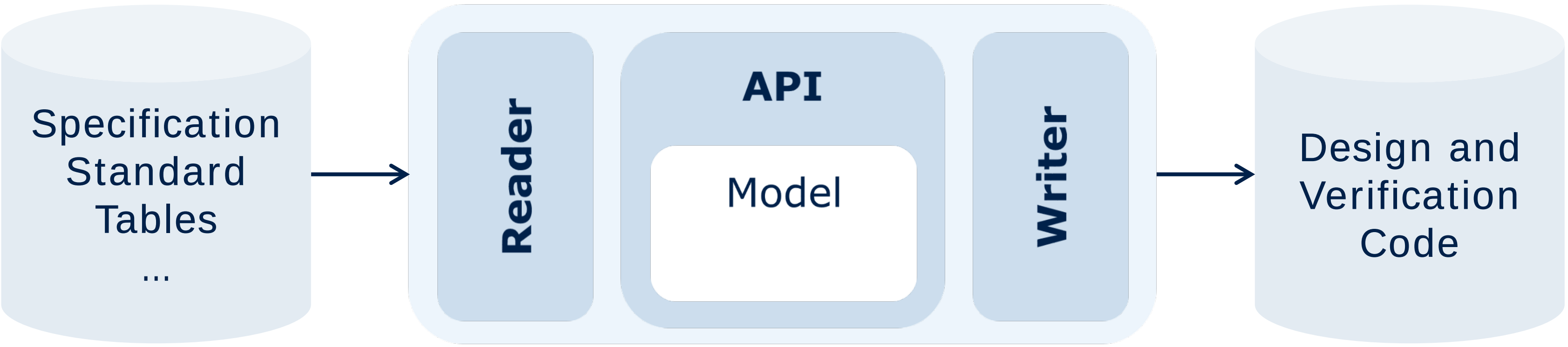
Tasks of a reader:

- 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

```
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)

```
library IEEE;
use IEEE.std_logic_1164.all;

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

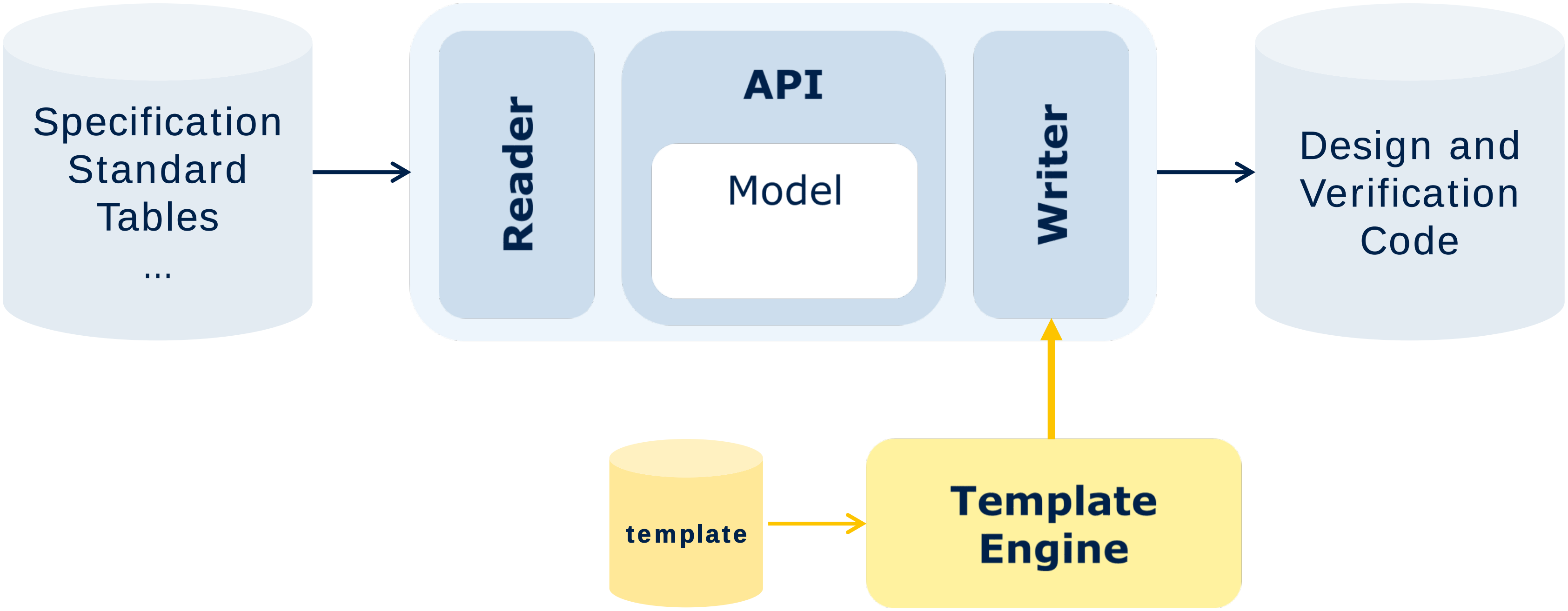
target code

substitution

render directive

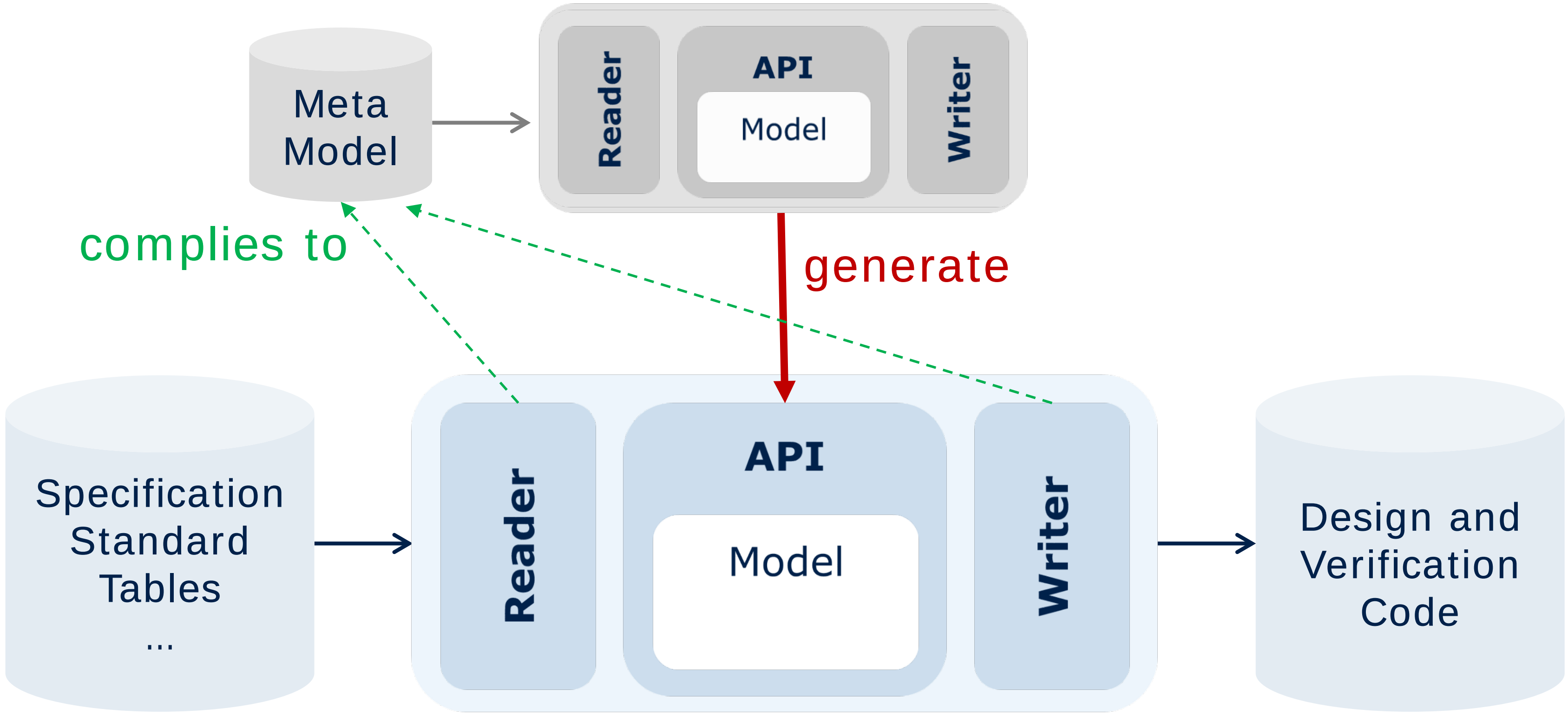
```
library IEEE;
use IEEE.std_logic_1164.all;

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;
```

Model-View Separation
Template Engines



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

Model-View Separation
A Core Model of a Metamodel

What is a simple Metamodel composed of

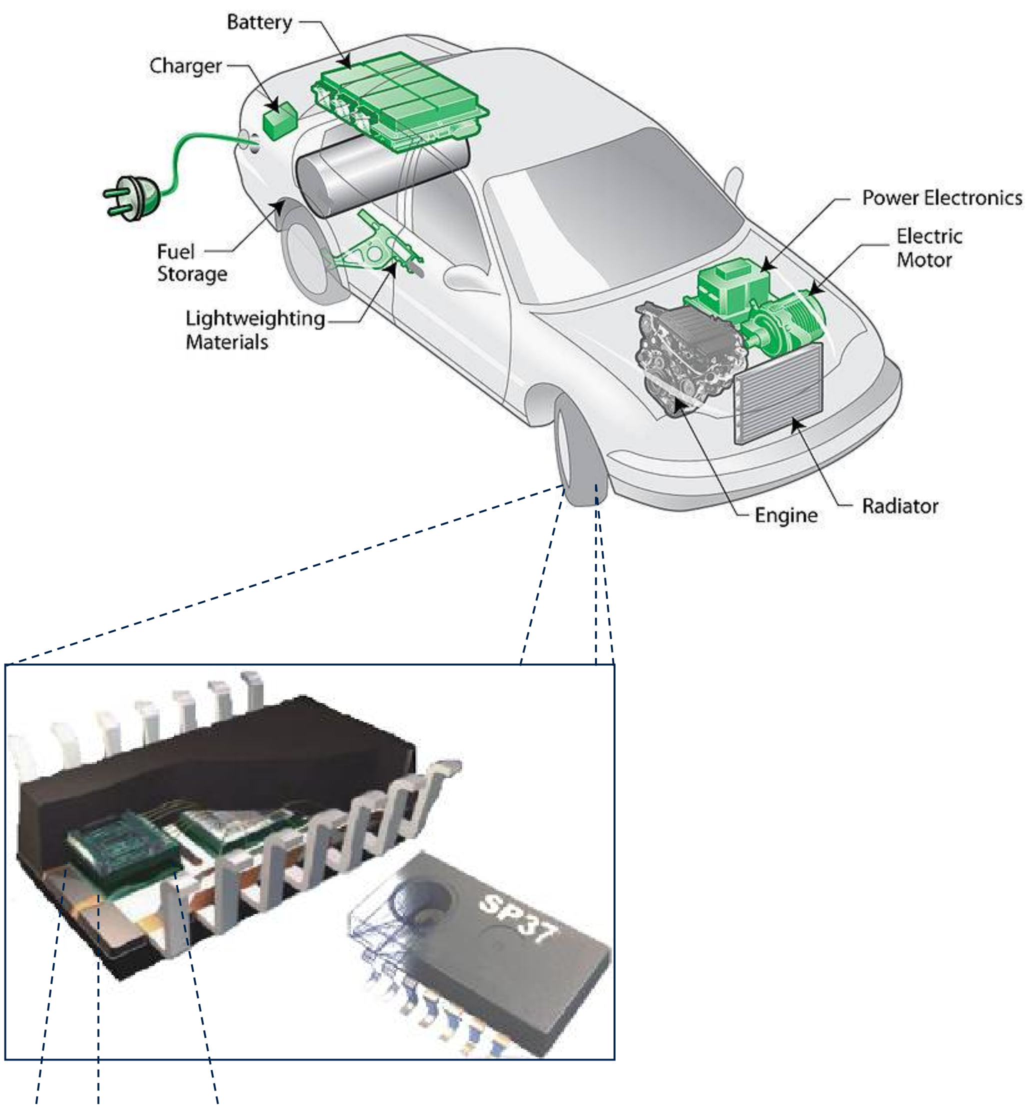
- Composite Data
  - Typed or Un-Typed Attributes
  - Typed or Un-Typed Children
  - Typed or Un-Typed Links
- Optional multiplicity or other constraints

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)

Metamodeling Technology: <u>Modeling</u> Is About Structuring and Formalizing Things



car

Type : BMW
Name: i6

wheel

Position: Right
Type; 190

electronics

Version: 1.2.3
Voltage: 24

pressure sensor

Provider: IFX

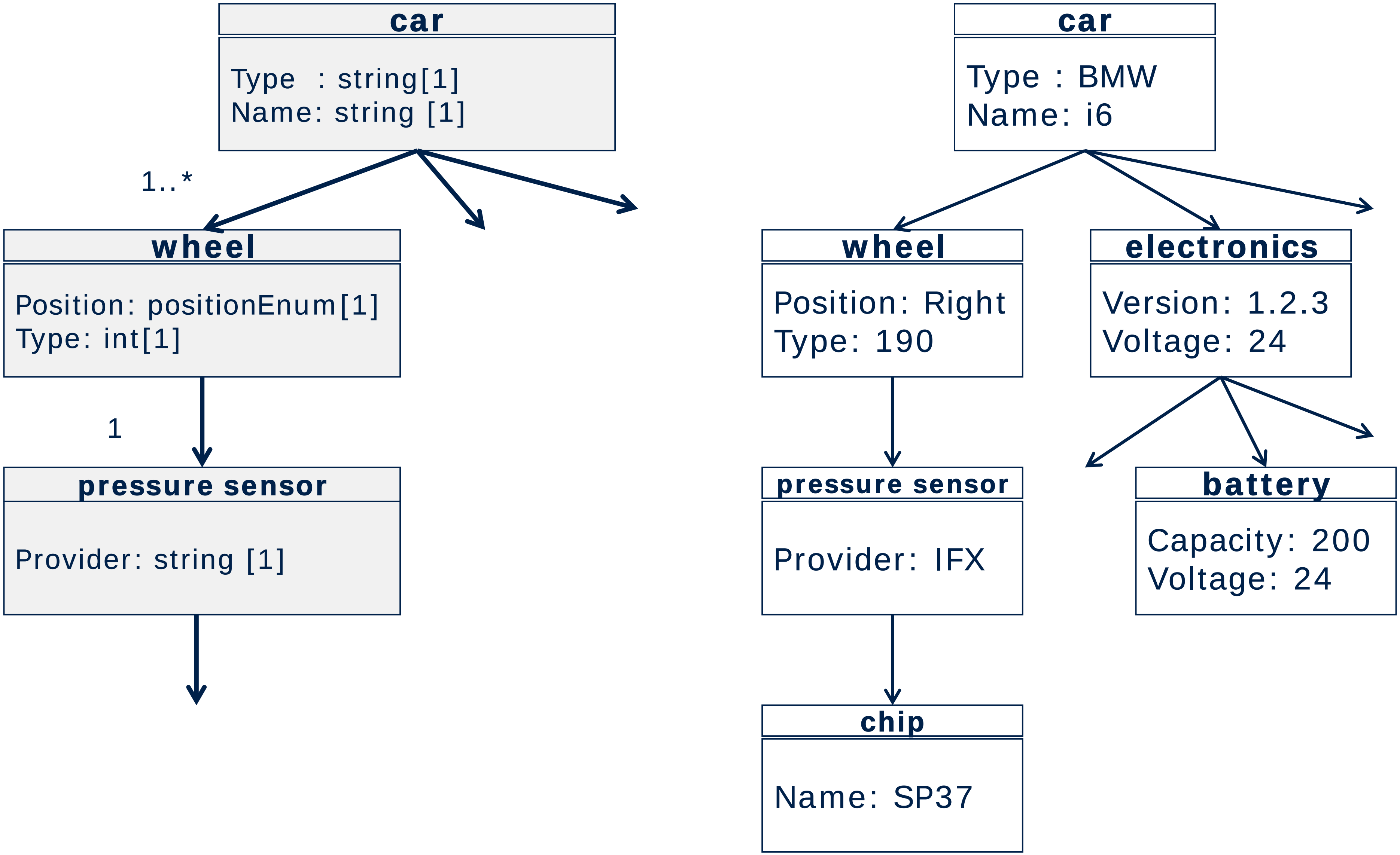
battery

Capacity: 200
Voltage: 24

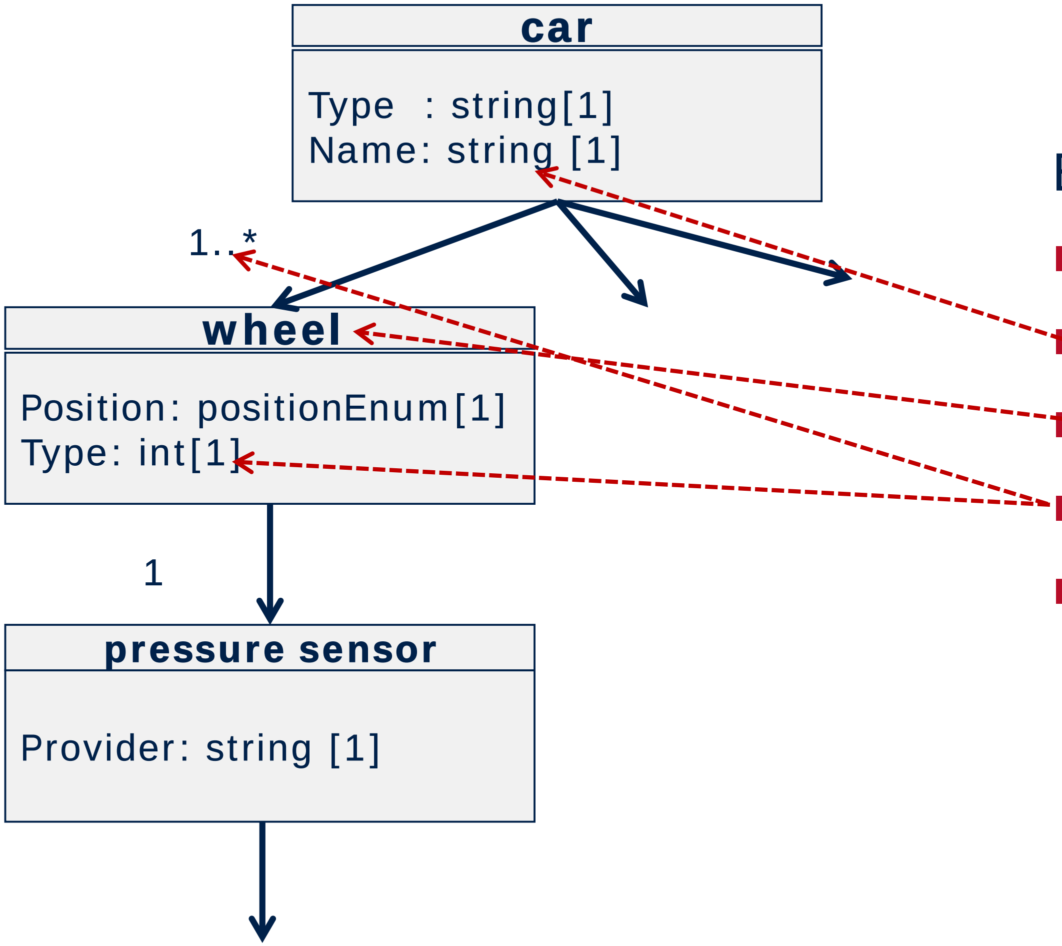
chip

Name: SP37

Metamodeling Technology: Metamodeling Is About Structuring and Formalizing Models



Metamodeling Technology: Metamodeling 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

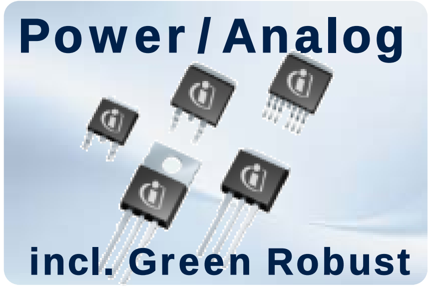
Details in 2nd part of the tutorial

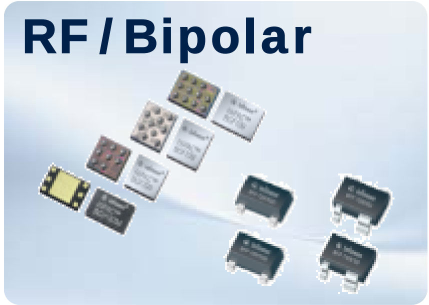
- Graphical formalism (primarily) to describe/model SW Systems
  - Formalisms describe structure, behavior and interaction
  - Examples are class diagrams, object diagrams, state diagrams, activity diagrams
  - UML is based on a superstructure (MOF, EMOF) that defines the formalism
  - OCL (object constraint language) is used to defined further constraints
- Stereotypes as SysML and 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 <u>flexibility in Metamodeling</u>

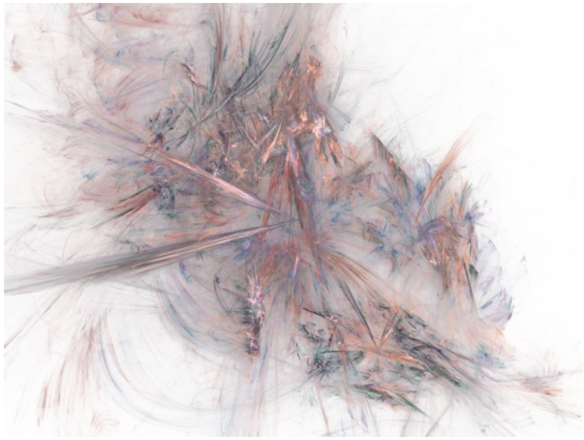
- Extendibility
- Constraints
- Interaction
- Composition

Metamodeling Technique
Extendibility and Constraints

Examples

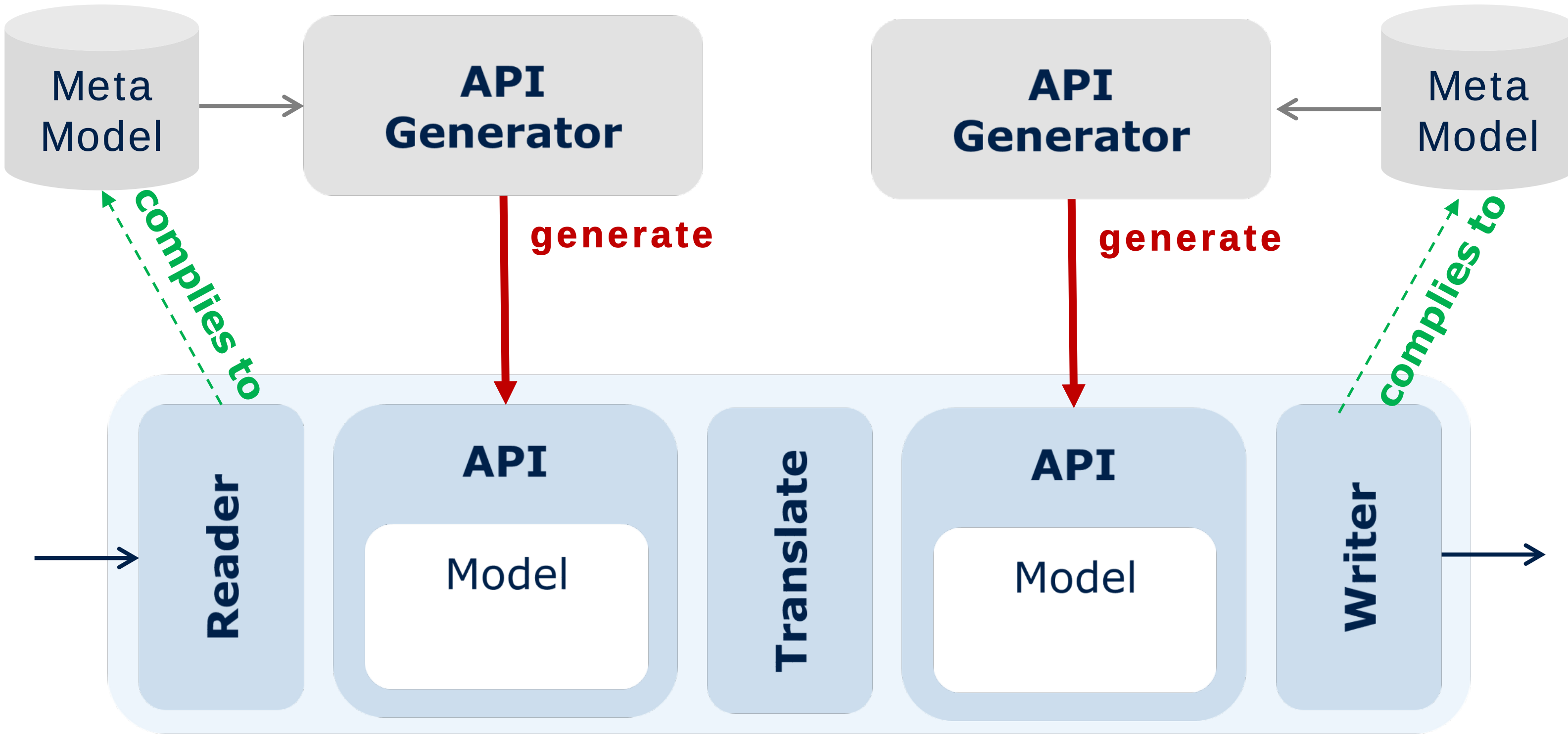
- Analog types and their properties
- Register protection mechanisms
- Clocked State Diagrams

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

Metamodeling 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

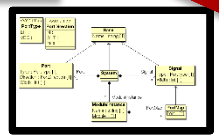
Layers in Structuring Data

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

Defines Structure

Generate Metamodel Infrastructure and Metamodels

Metamodel

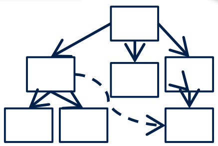


Defines Structure of Model

Defines Structure

Generate Model Infrastructure Models

Model

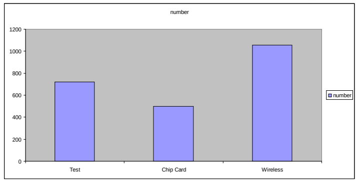


Defines content of view language independently

Defines Content

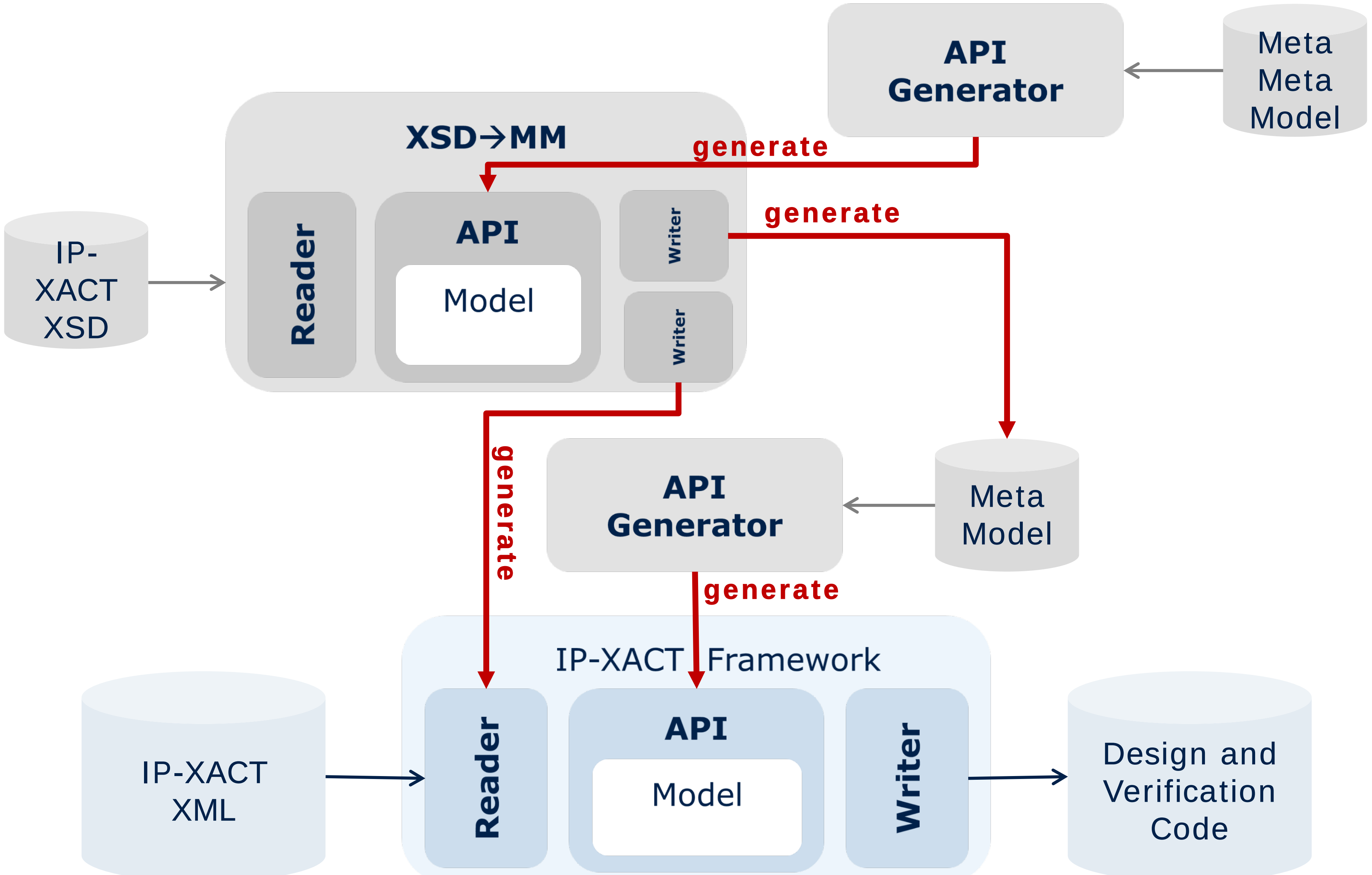
Generate View(s)

View



Implementation of content

Meta-Metamodel: Is About Structuring Metamodels, i.e. Metamodel of Metamodel



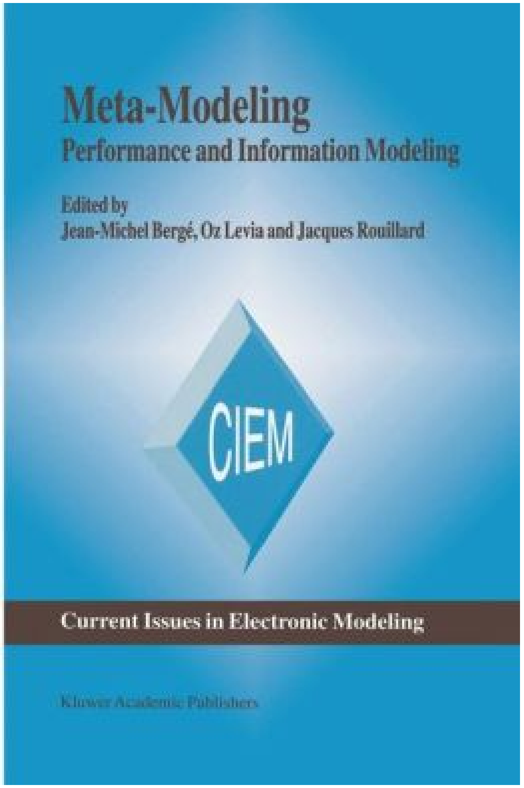
Meta-Metamodel: Is About Structuring Metamodels, i.e. Metamodel of Metamodel

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! Metamodeling 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)*
- *Meta-Modeling: Performance and Information Modeling* Current Issues in Electronic Modeling (6), Springer, ISBN 9780792396871
- Meta-Modeling: Current Issues in Electronic Modeling (6), ISBN 9780792396871



It's All About Structuring
Summary and Outline

Metamodeling and Code generation is

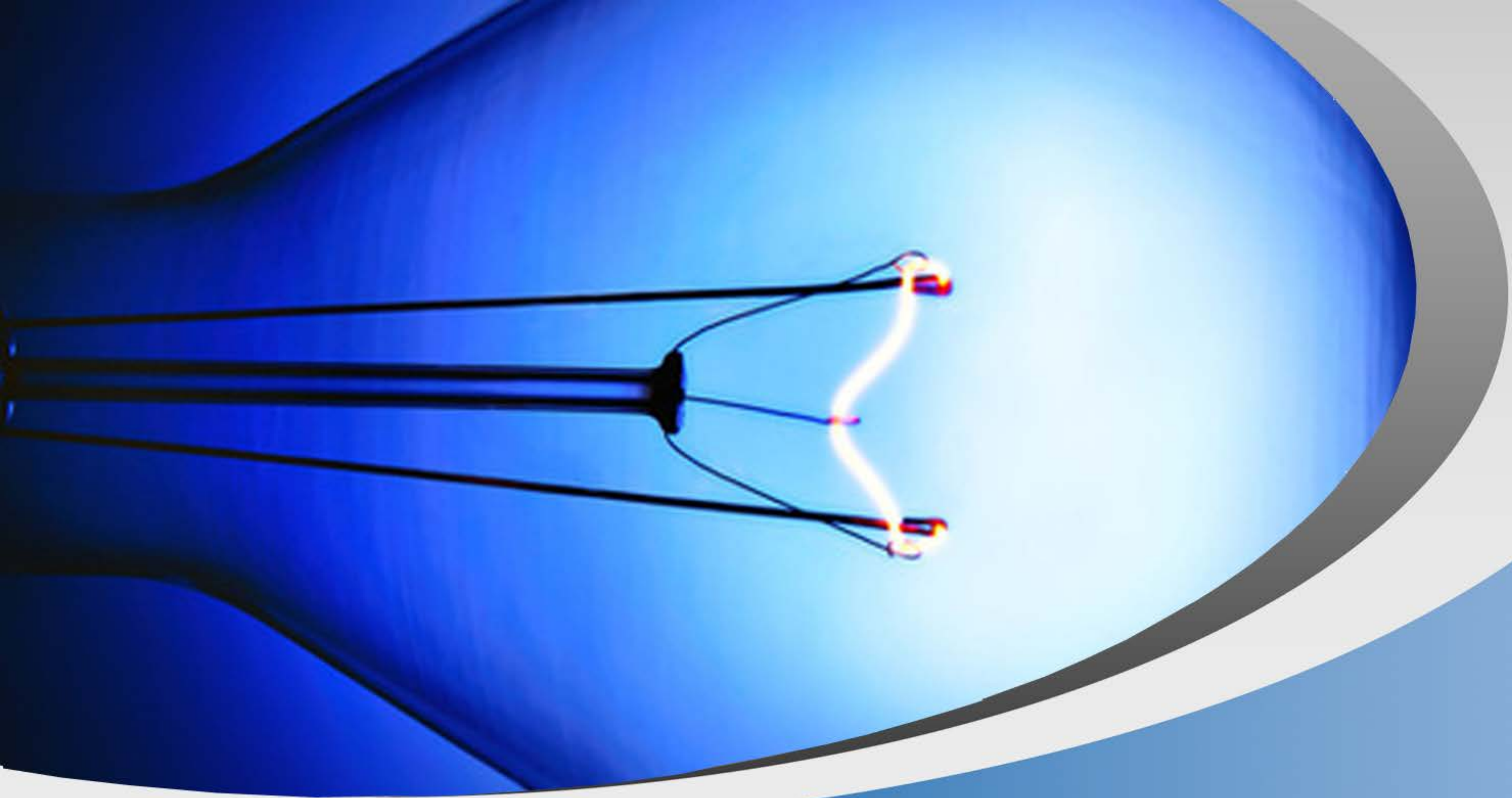
- an industry proven technology to efficiently build domain/problem specific tools following a specific structure

Modeling in the context of Metamodeling is about

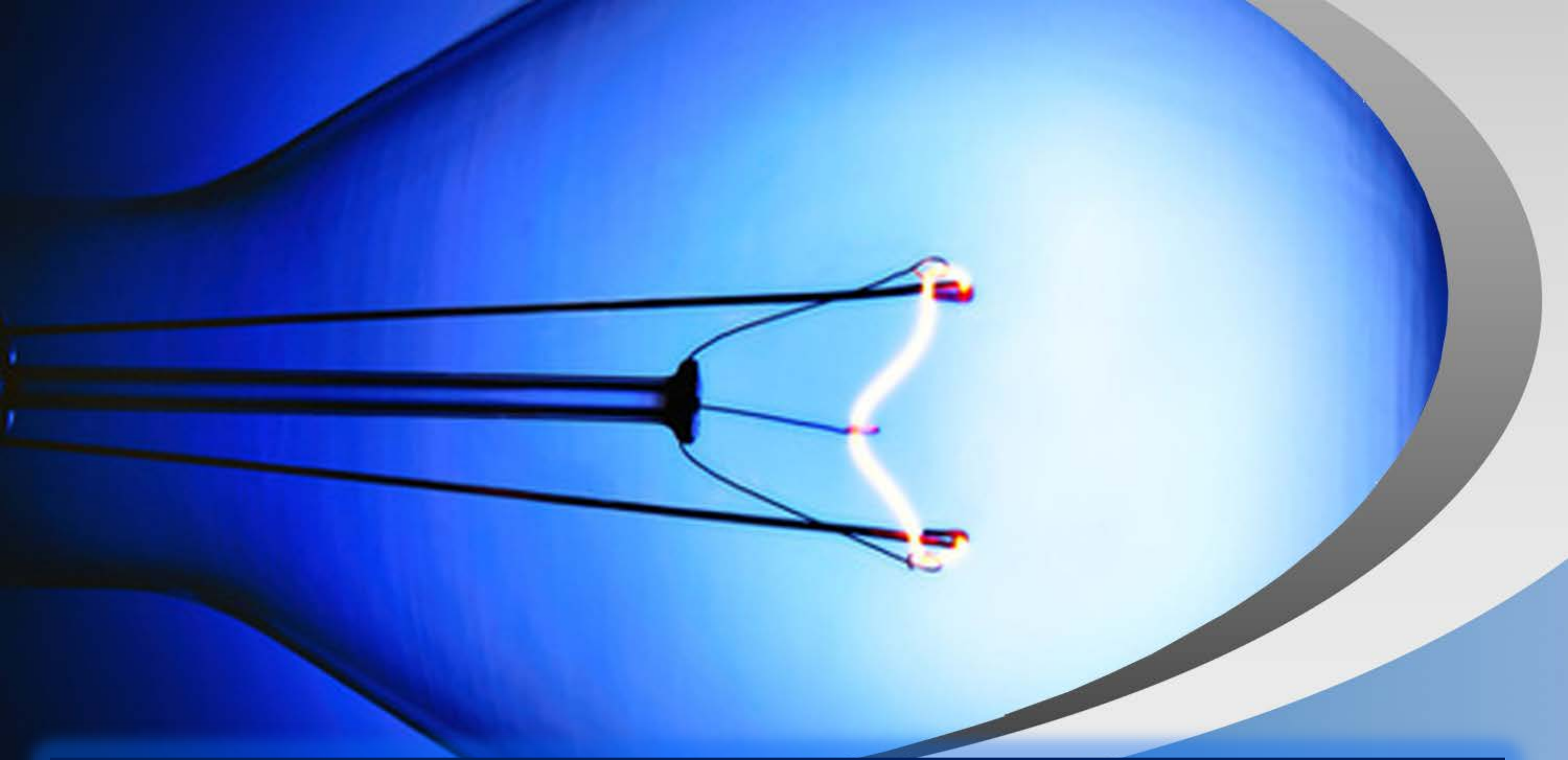
- structuring things in a design context

Metamodeling is about

- Structuring Models



Thank you!



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

Well known Metamodels in EDA and Design: UML/SysML

Wolfgang Ecker, Infineon; Rainer Findenig, Intel

SYSTEMS INITIATIVE

Unified Modeling Language

The Unified Modeling Language (UML) is a general-purpose modeling language in the field of software engineering, which is designed to provide a standard way to visualize the design of a system.

en.wikipedia.org

Unified Modeling Language

Software Centric

The Unified Modeling Language (UML) is a general-purpose modeling language in the field of software engineering, which is designed to provide a standard way to visualize the design of a system.

en.wikipedia.org

Unified Modeling Language

The Unified Modeling Language (UML) is a general-purpose modeling language in the field of software engineering, which is designed to provide a standard way to visualize the design of a system.

Standardized by the Object Management Group

en.wikipedia.org

Unified Modeling Language

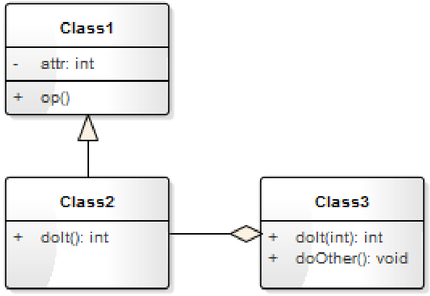
The Unified Modeling Language (UML) is a general-purpose modeling language in the field of software engineering, which is designed to provide a standard way to visualize the design of a system.

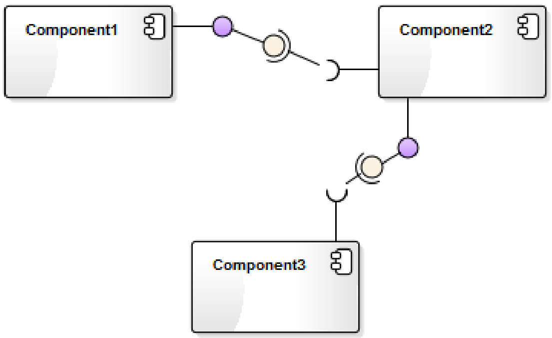
Graphical Language

en.wikipedia.org

Unified Modeling Language

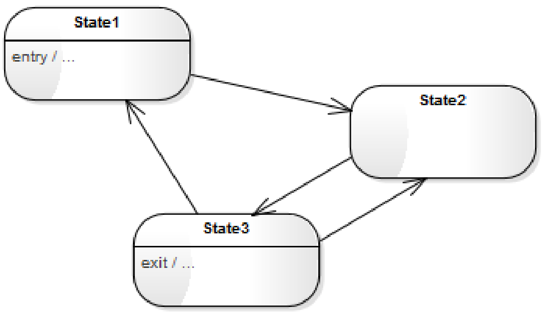
- Structural modeling:
  - Class diagram
  - Component diagram
  - Deployment diagram
  - ...

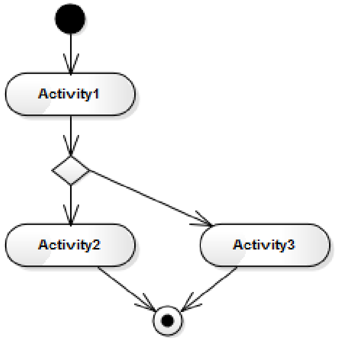




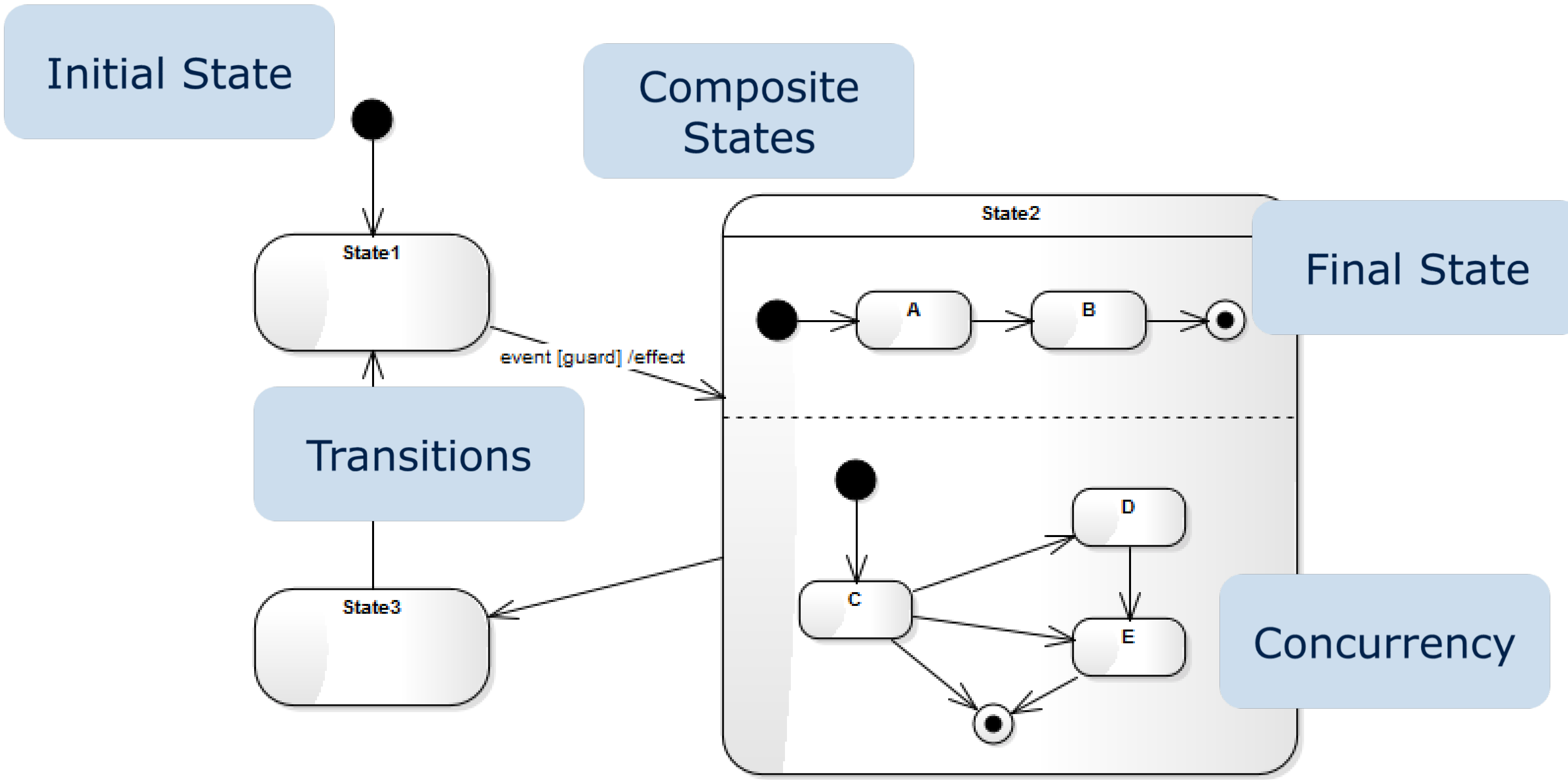
Unified Modeling Language

- Behavioral modeling:
  - Activity diagram
  - Sequence diagram
  - State diagram
  - ...



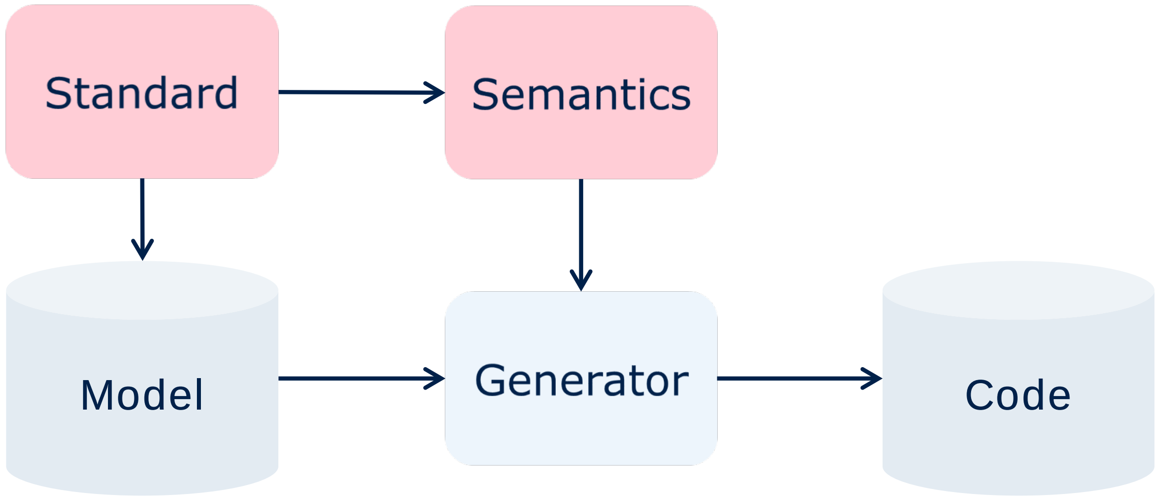


An Example: UML State Diagrams



UML?

- Graphical Language
  - Easy to read
  - Easy to write?
- Semantics
  - Not formally defined; software oriented
  - Given to your model as part of the code generation
    - Tool support is critical!



UML: The Spirit of Metamodeling

M3

- UML Infrastructure

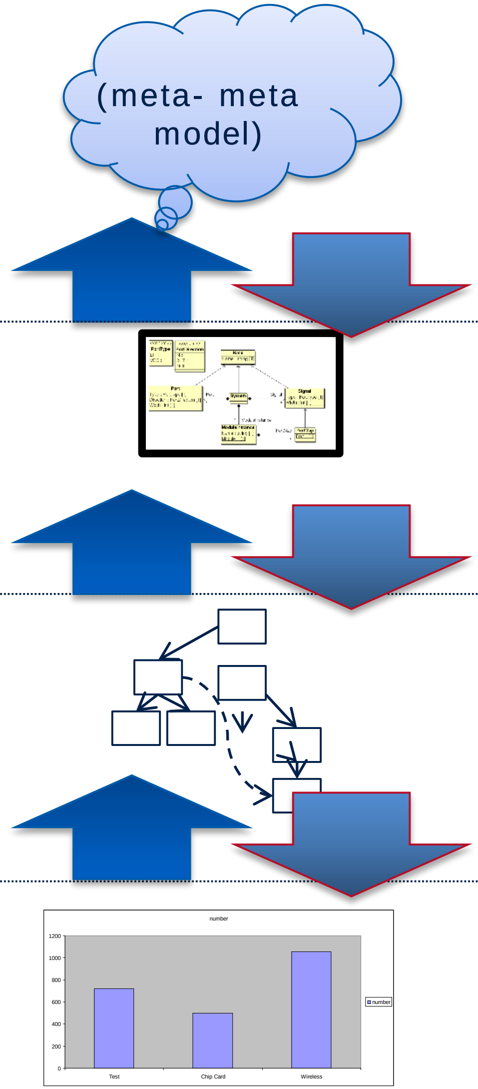
M2

- UML Superstructure

M1

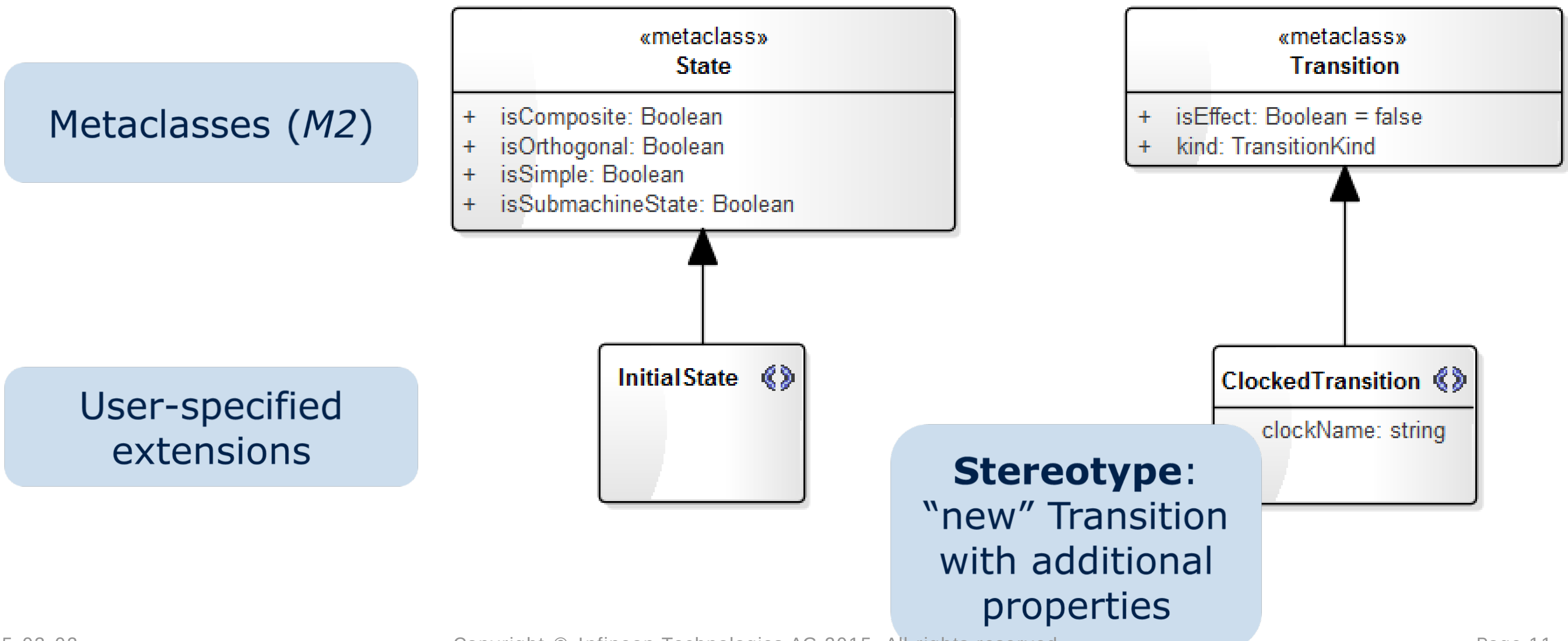
- User Diagrams

M0



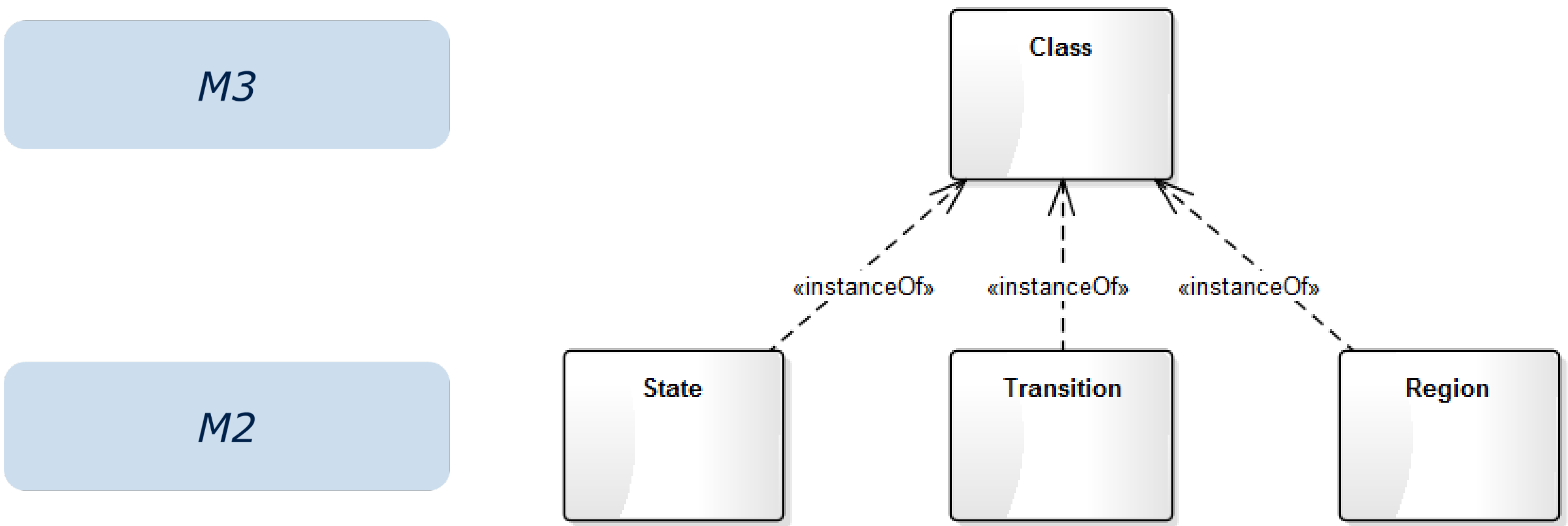
Extending UML: Profiles

- Extension mechanism for customizing UML
- Light-weight, easy
- Strictly additive, no fundamental changes



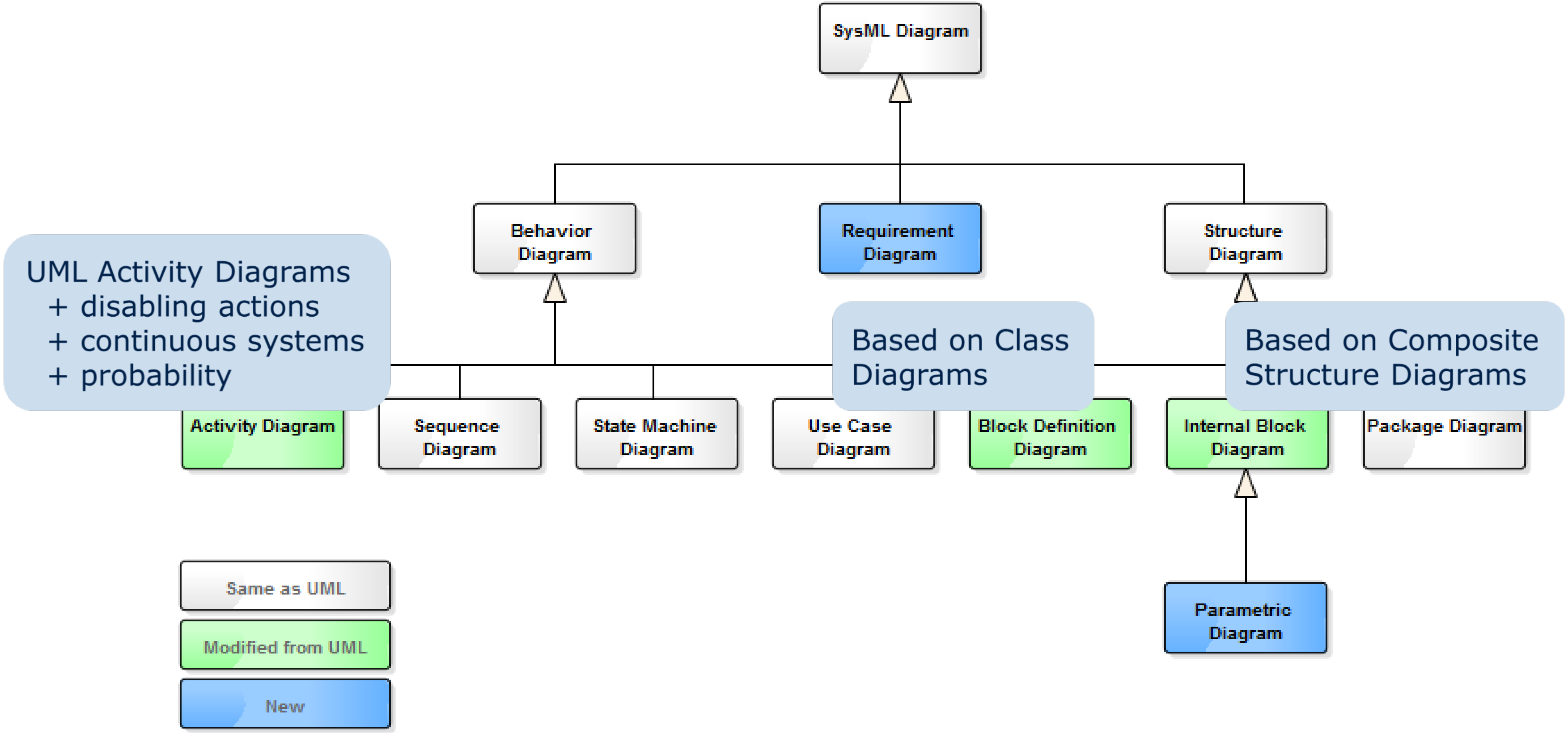
Extending UML: MOF – Meta-Object Facility

- UML itself is defined in the MOF
- Allows defining completely new Metamodels

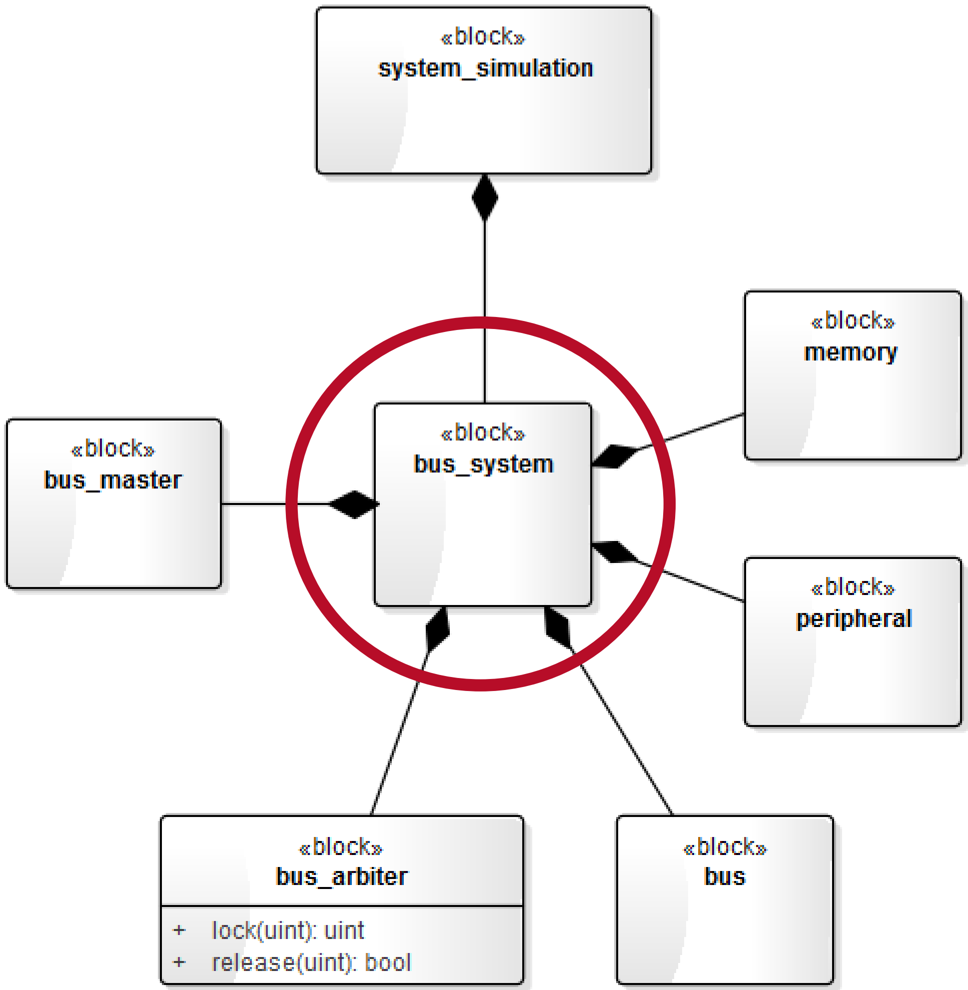


SysML

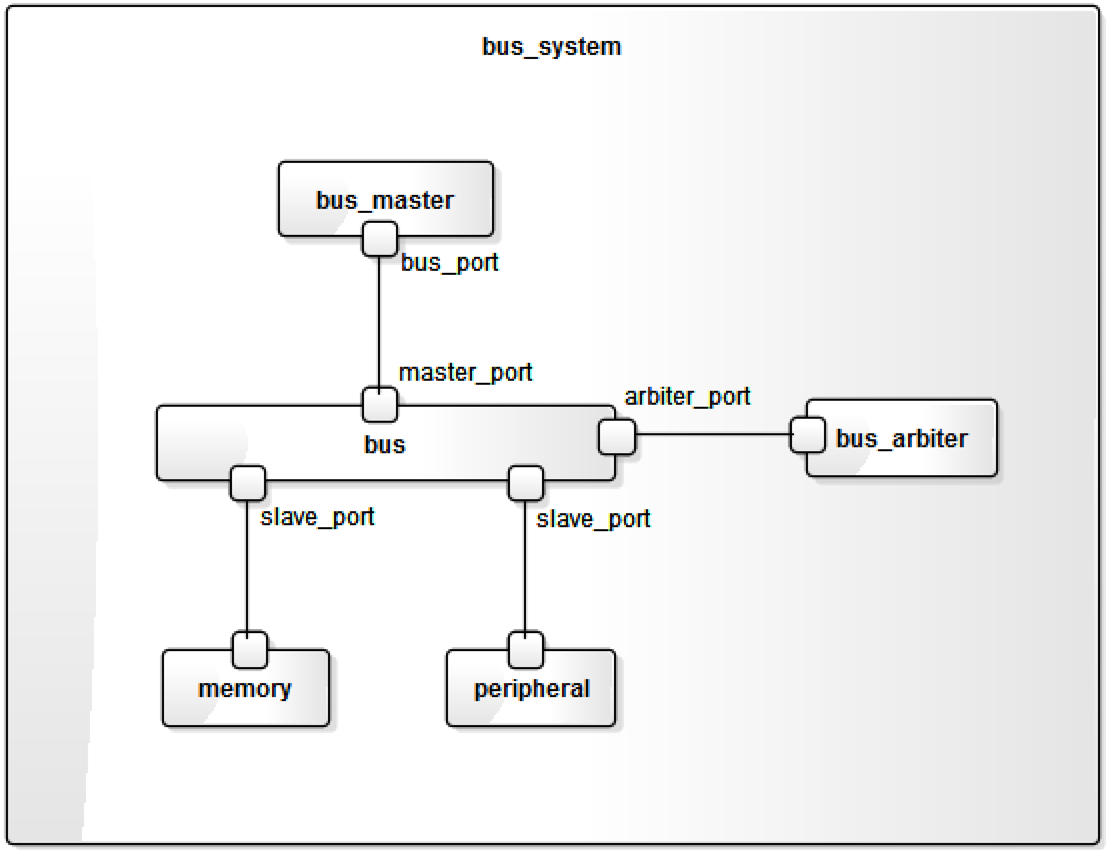
- Extended subset of UML
- Defined using profiles



SysML: Block Definition Diagram

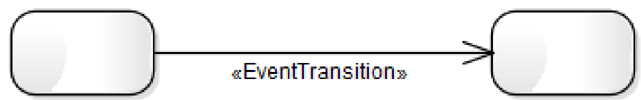


SysML: Internal Block Diagram

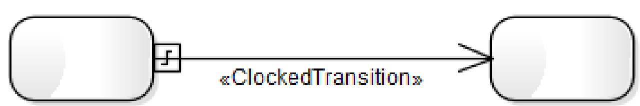


UML State Diagrams for Different Abstraction Levels

- UML Profile
  - Event-driven transitions:
    - Derived from time, transactions, or other internal/external events

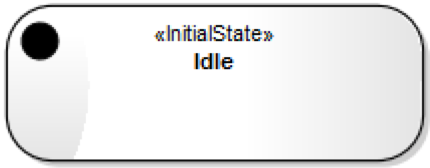


  - Clock-driven transitions:
    - Derived from an internal clock
    - Can use guards for specifying timeouts

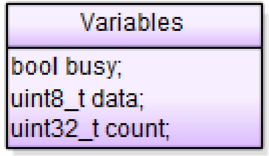


UML State Diagrams for Different Abstraction Levels

- UML Profile
  - Initial states to conform with hardware reset semantics

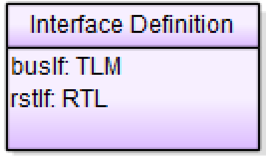


  - Global and local variables

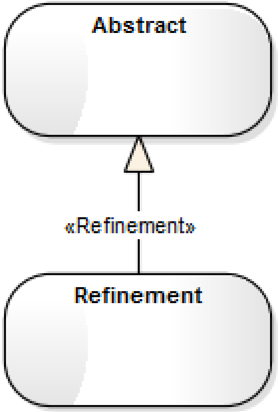


UML State Diagrams for Different Abstraction Levels

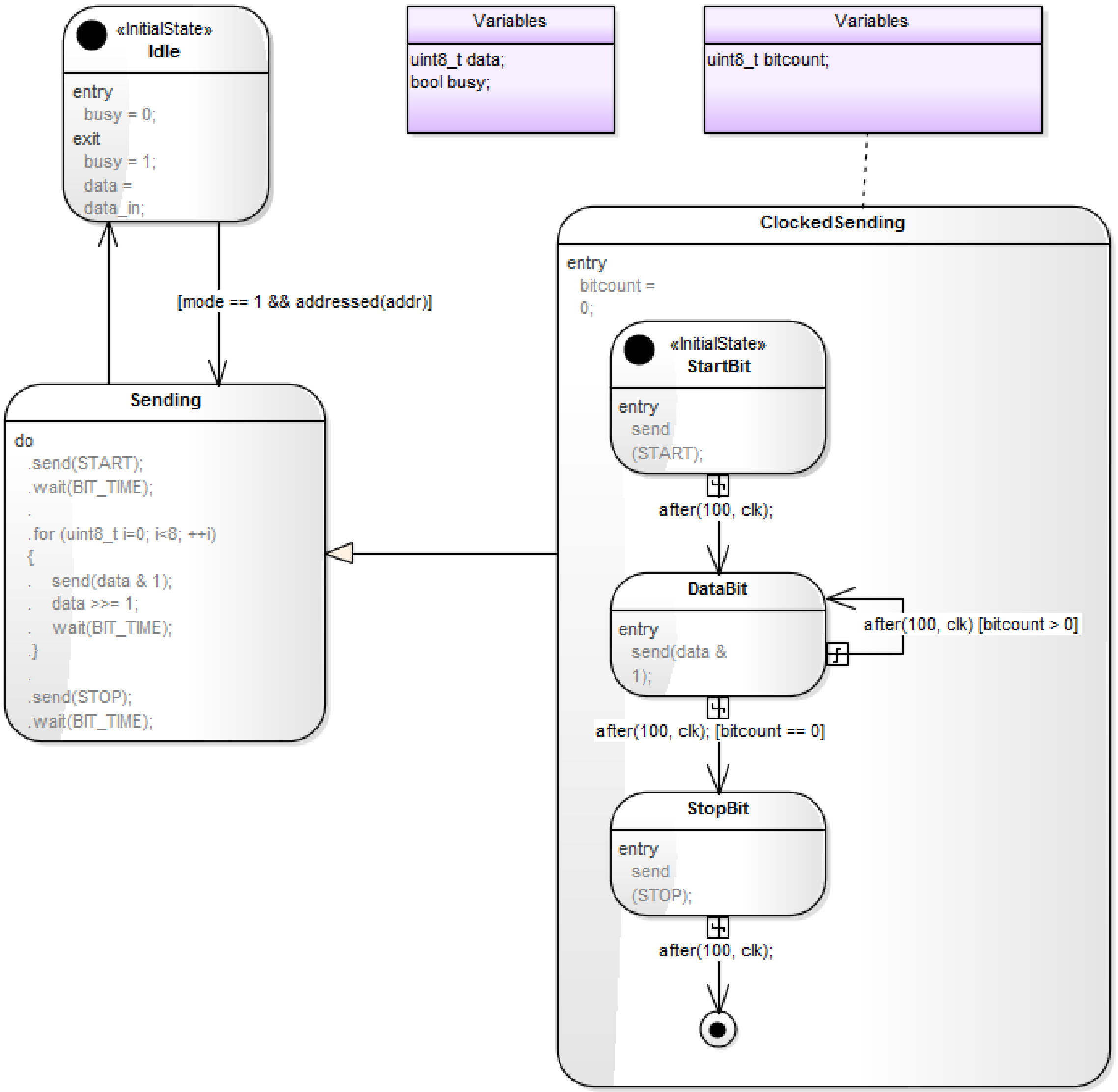
- UML Profile
  - Link to external interface definition
    - Including selection of desired abstraction level

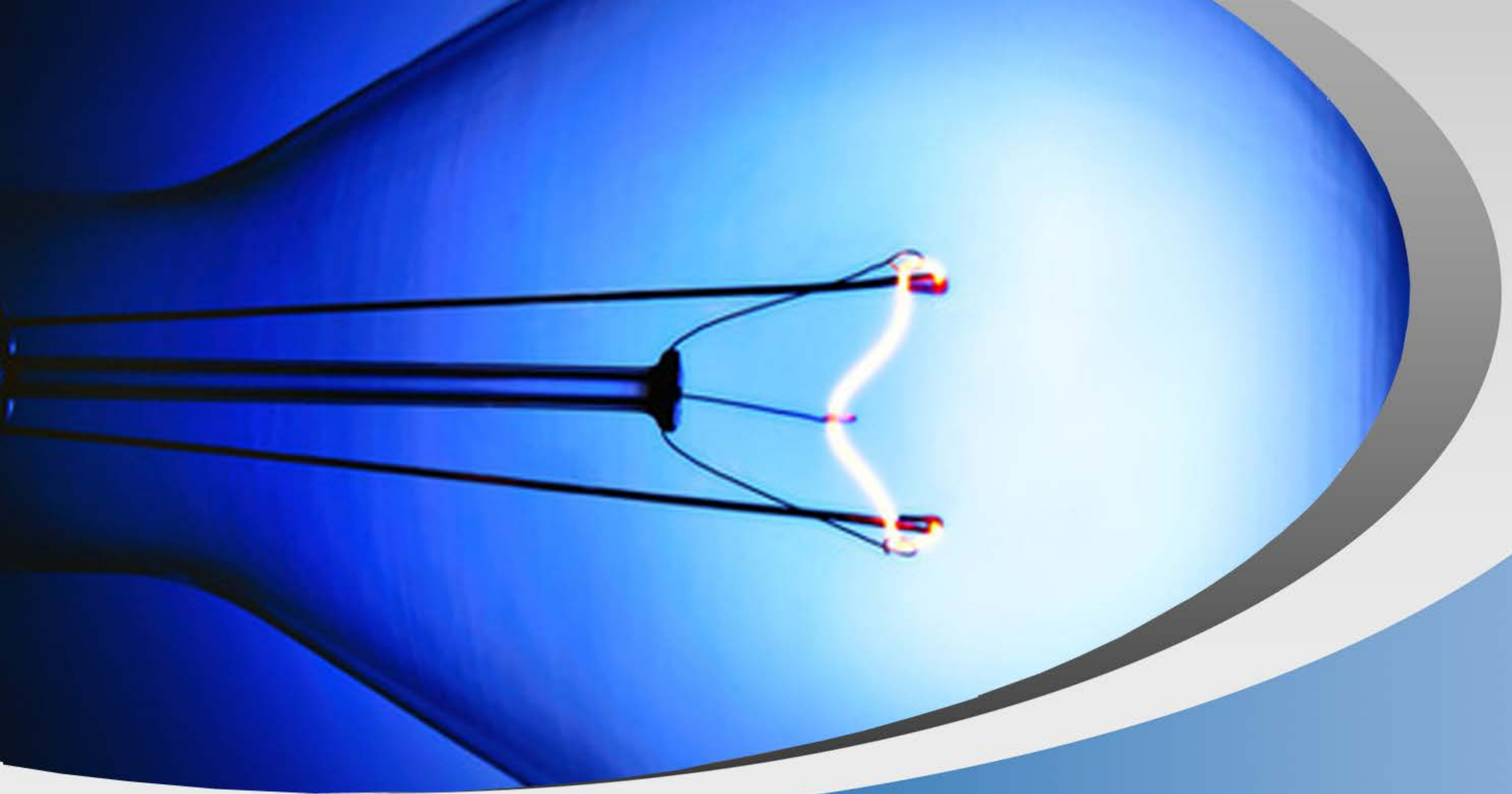


  - Refinement between states and transitions

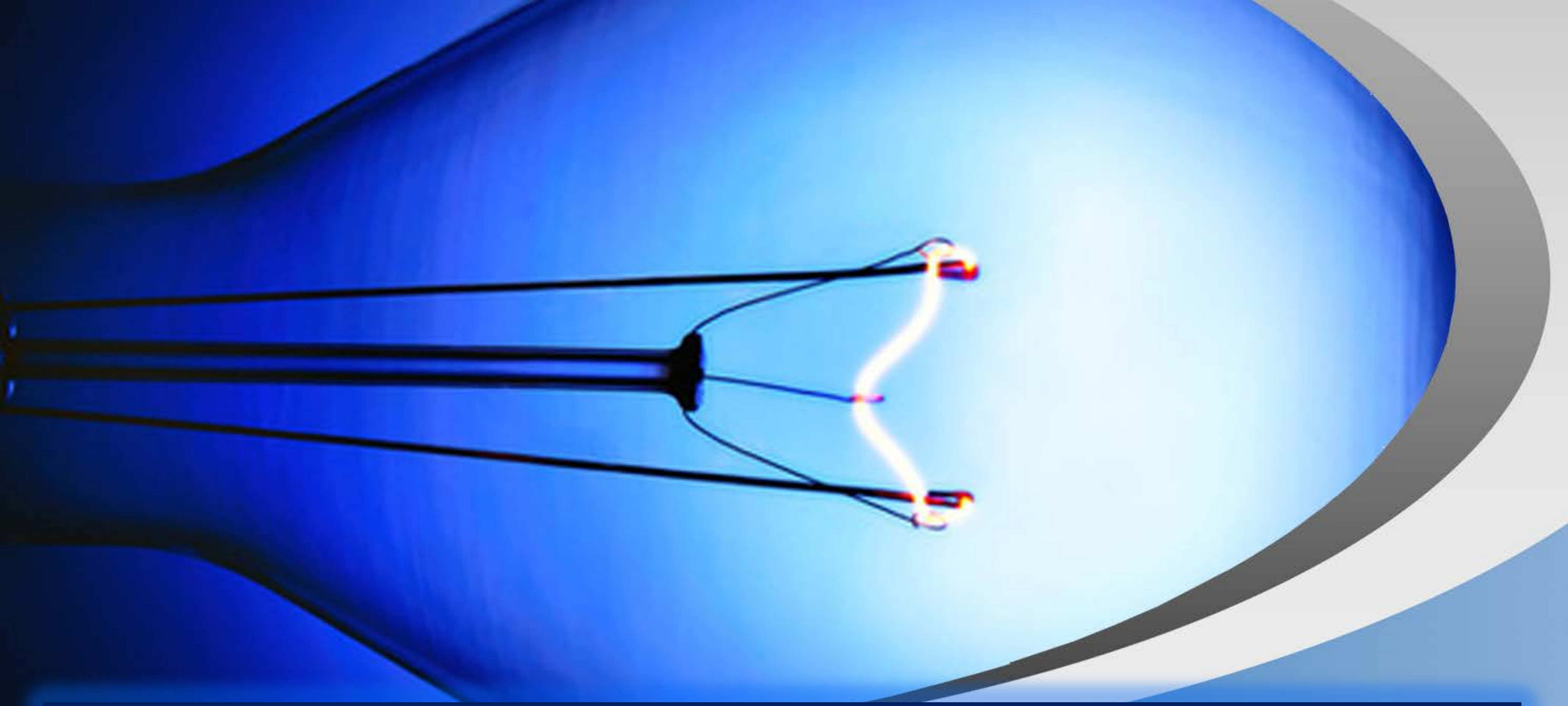


Example: SIF


«InitialState»
Idle
entry
busy = 0;
exit
busy = 1;
data =
data_in;
Variables
uint8_t data;
bool busy;
Variables
uint8_t bitcount;
[mode == 1 && addressed(addr)]
Sending
do
.send(START);
.wait(BIT_TIME);
.for (uint8_t i=0; i<8; ++i)
{
send(data & 1);
data >>= 1;
wait(BIT_TIME);
}
.send(STOP);
.wait(BIT_TIME);
ClockedSending
entry
bitcount =
0;
«InitialState»
StartBit
entry
send
(START);
after(100, clk);
DataBit
entry
send(data &
1);
after(100, clk) [bitcount > 0]
after(100, clk); [bitcount == 0]
StopBit
entry
send
(STOP);
after(100, clk);



Thank you!



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

Well known Metamodels in EDA and Design: IP-XACT

Wolfgang Mueller, Heinz Nixdorf Institute; Daniel Müller-Gritschneder, Technical University of Munich

accellera
SYSTEMS INITIATIVE

IP-XACT Overview

IP-XACT IEEE 1685

Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows

- design-language neutral design exchange format
- Electronic System Level IP components (ESL netlists + Code Generation)
  - IP component attributes: interfaces, signals, parameters, memory maps, registers, ...
  - IP component processing information: generators and file sets
    for assembly, simulation, synthesis, test insertion,

Related Spirit/Accellera standard:
SystemRDL (Register Description Language) for HW/SW interface components, 2009

2003	2009	2009	2014
SPIRIT Consortium → IP-XACT V1.0	Accelllera → IP-XACT V1.4	IEEE 1685-2009	IEEE 1685-2014

IP-XACT Overview

IP-XACT IEEE 1685

Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows

- de
- El
  - s, registers, ...
  - est insertion,

Related Sp

SystemRD ts, 2009

2003

14

IP-XACT complient data are structured by an

XML file format

IP-XACT IEEE 1685-2014 defines the

XML file structure by an

XML schema

What is XML?

XML (eXtensible Markup Language)

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<program>
  <language> C </language>
   <file> myFirst.c </file>
   <source>
         int main() { cout << "Hi there";}
   </source>
</program>
```

xml declaration

plain text file
(sequence of characters)

tags (start ... end)

data

Filename extension	.xml
Internet media type	application/xml text/xml [1]
Uniform Type Identifier (UTI)	public.xml
UTI conformation	public.text
Developed by	World Wide Web Consortium
Type of format	Markup language
Extended from	SGML
Extended to	Numerous languages, including XHTML · RSS · Atom · KML
Standard	1.0 (Fifth Edition) (November 26, 2008; 6 years ago) 1.1 (Second Edition) (August 16, 2006; 8 years ago)
Open format?	Yes

Source: en.wikipedia.org

XML tags & structure defined by either

- Data Type Definition (DTD)
- XML Schema Definition (XSD)

What is XSD?

XML Schema Definition (XSD)

- defines structure for xml file
- developed by World Wide Web Consort.
- file extension: .xsd
- compares to UML Class Diagrams
- note: xsd is defined in xml format!

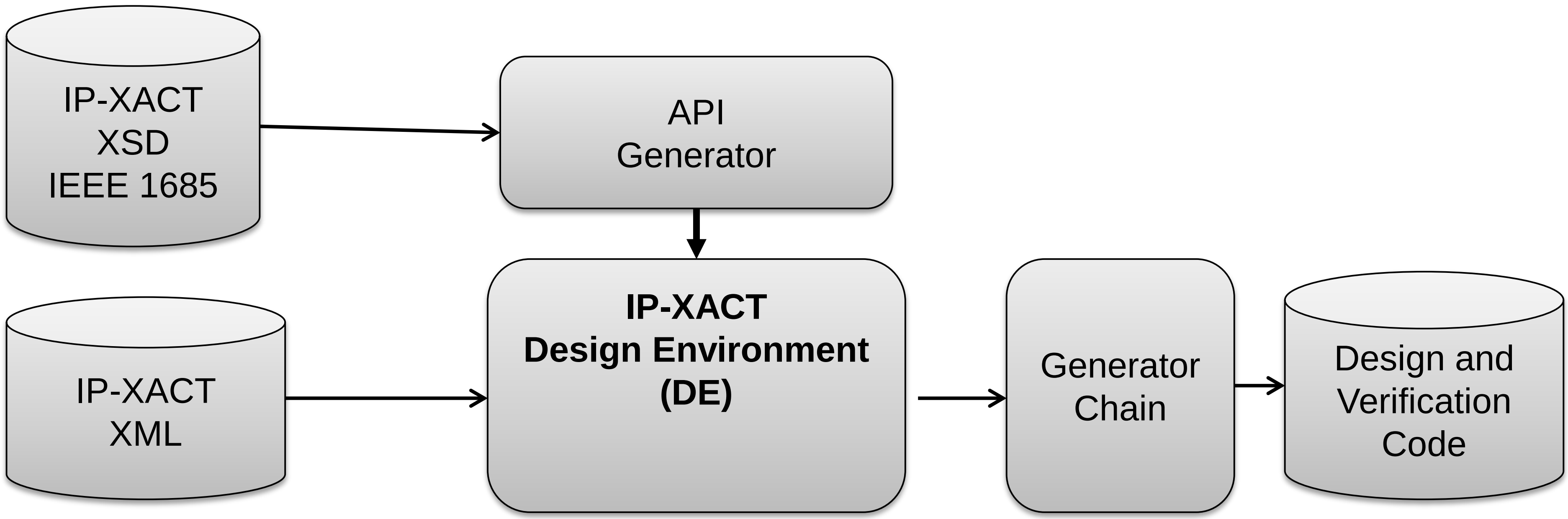
XML

```
<?xml version="1.0" encoding="UTF-8"?>
<program>
  <language> C  </language>
   <file> MyFirst.c </file>
   <source>  int main() { cout << “Hi there“;} </source>
</program>
```

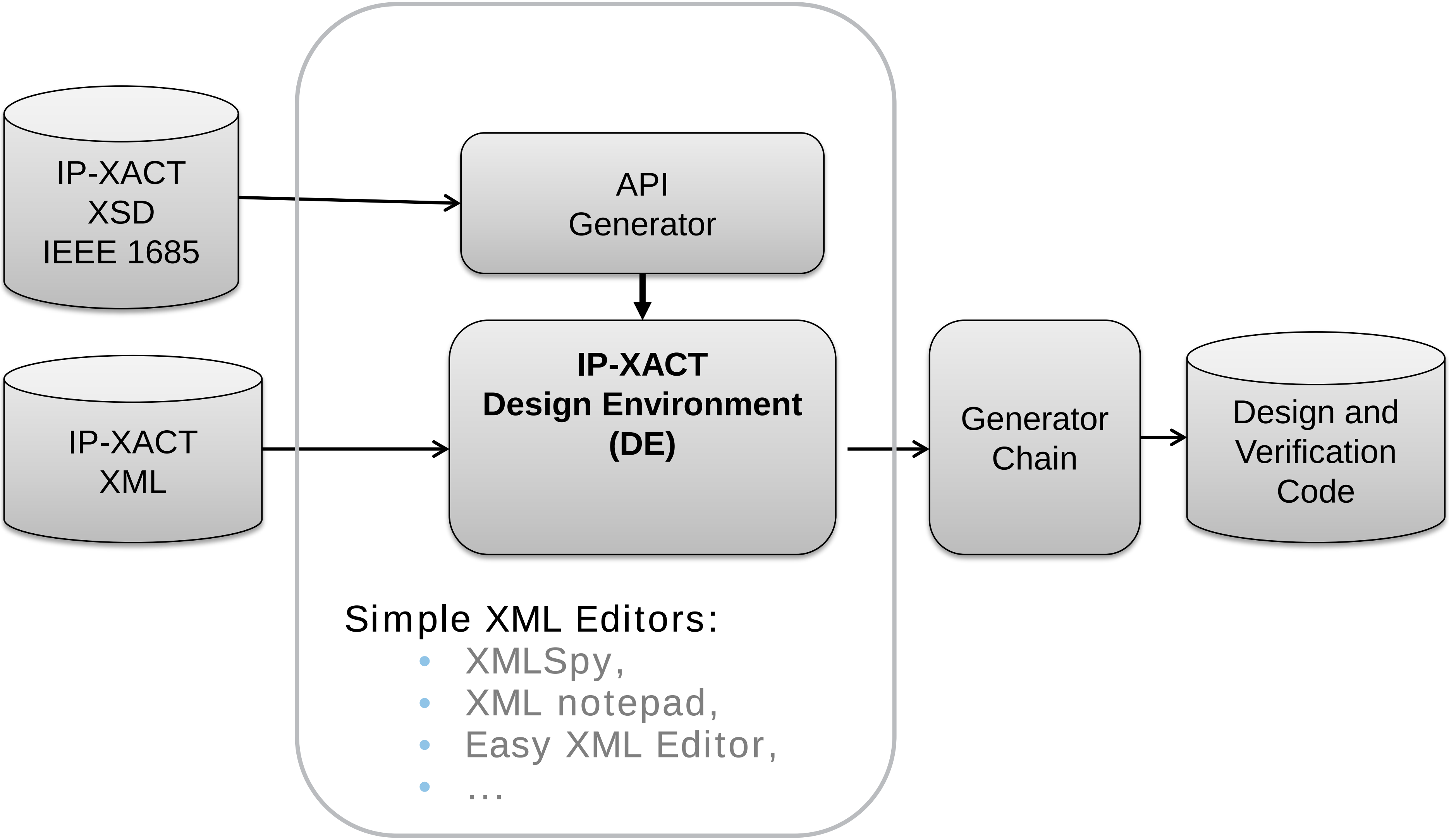
XSD

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name=“program">
  <xs:complexType>
    <xs:sequence>
      <xs:element name=“language" type="xs:string"/>
      <xs:element name=“file" type="xs:string"/>
      <xs:element name=“source" type="xs:string"/>
     </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

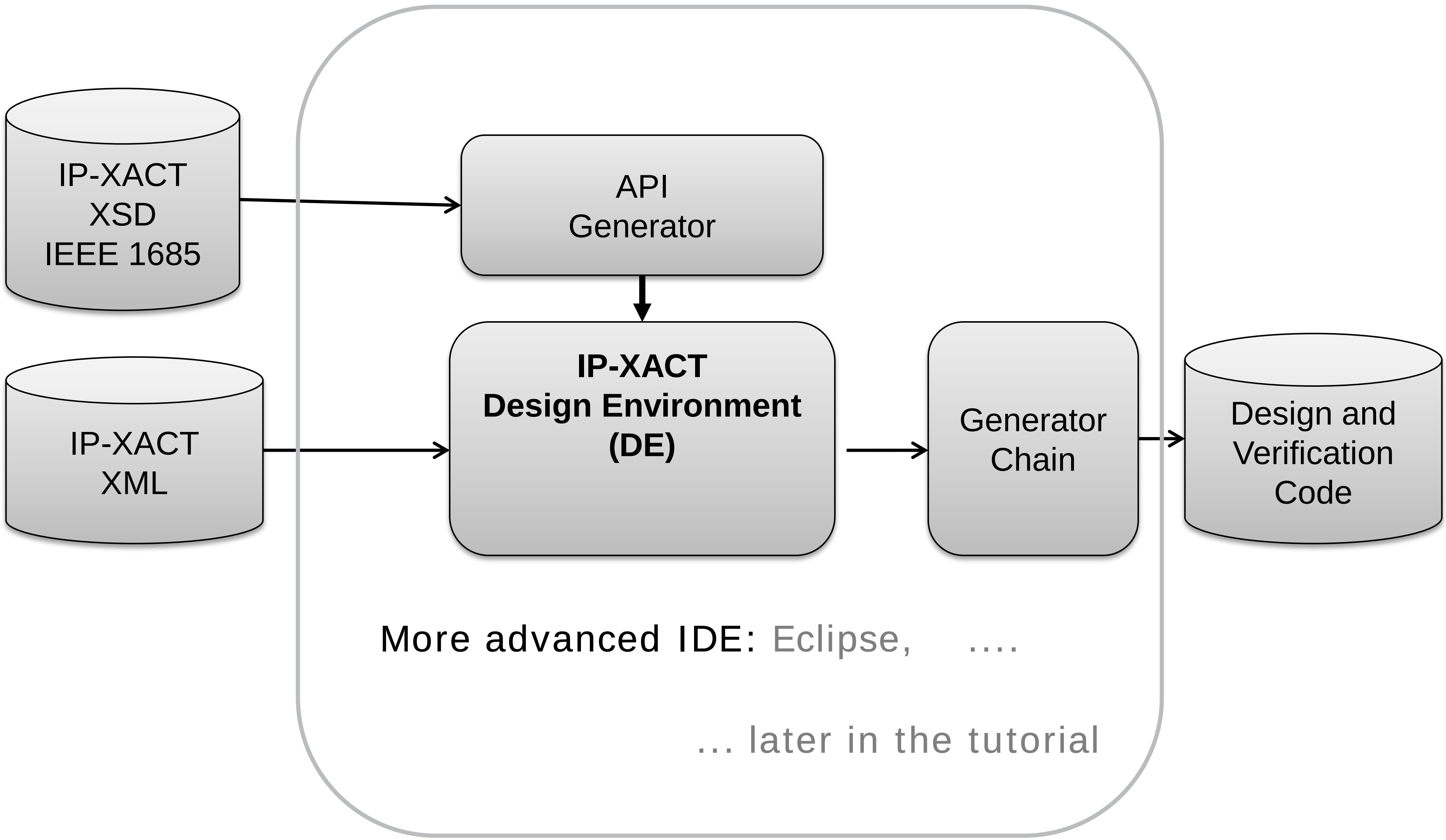
Recall: Metamodel Defines IP-XACT Structure



IP-XACT Design Environment



IP-XACT Design Environment



IP-XACT IEEE 1685-2014

IEEE STANDARDS ASSOCIATION IEEE

IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows

IEEE Computer Society

Sponsored by the
Design Automation Standards Committee

IEEE
3 Park Avenue
New York, NY 10016-5997
USA

IEEE Std 1685™-2014
(Revision of
IEEE Std 1685-2009)

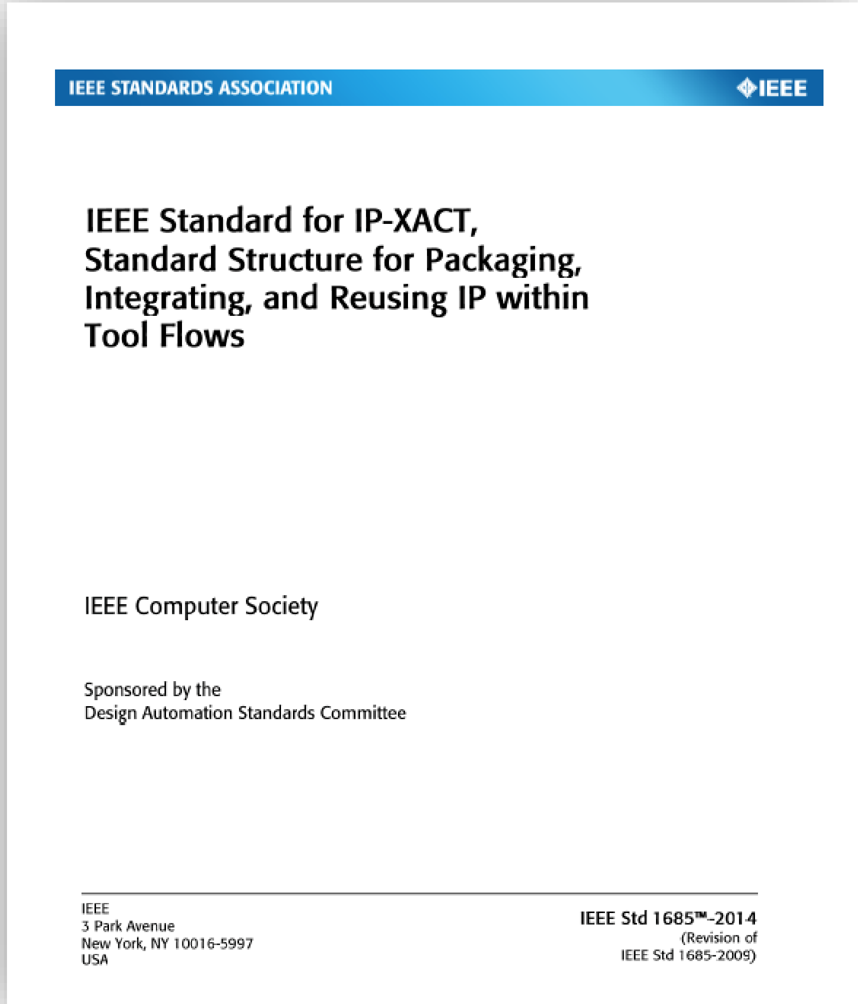
Contents

1. Overview
2. Normative references
3. Definitions, acronyms, abbreviations
4. Interoperability use model
5. Interface definition descriptions
6. Component descriptions
7. Design descriptions
8. Abstractor descriptions
9. Generator chain descriptions
10. Design configuration descriptions
11. Catalog descriptions
12. Addressing
13. Data visibility

Annex A – Annex I:

Bibliography, Semantic consistency rules, Common elements and concepts,Types, SystemVerilog expressions, Tight generator interface, External bus with an internal/digital interface, Bridges & channels, Examples

IP-XACT IEEE 1685-2014


IEEE STANDARDS ASSOCIATION ◆IEEE

IEEE Standard for IP-XACT,
Standard Structure for Packaging,
Integrating, and Reusing IP within
Tool Flows

IEEE Computer Society

Sponsored by the
Design Automation Standards Committee

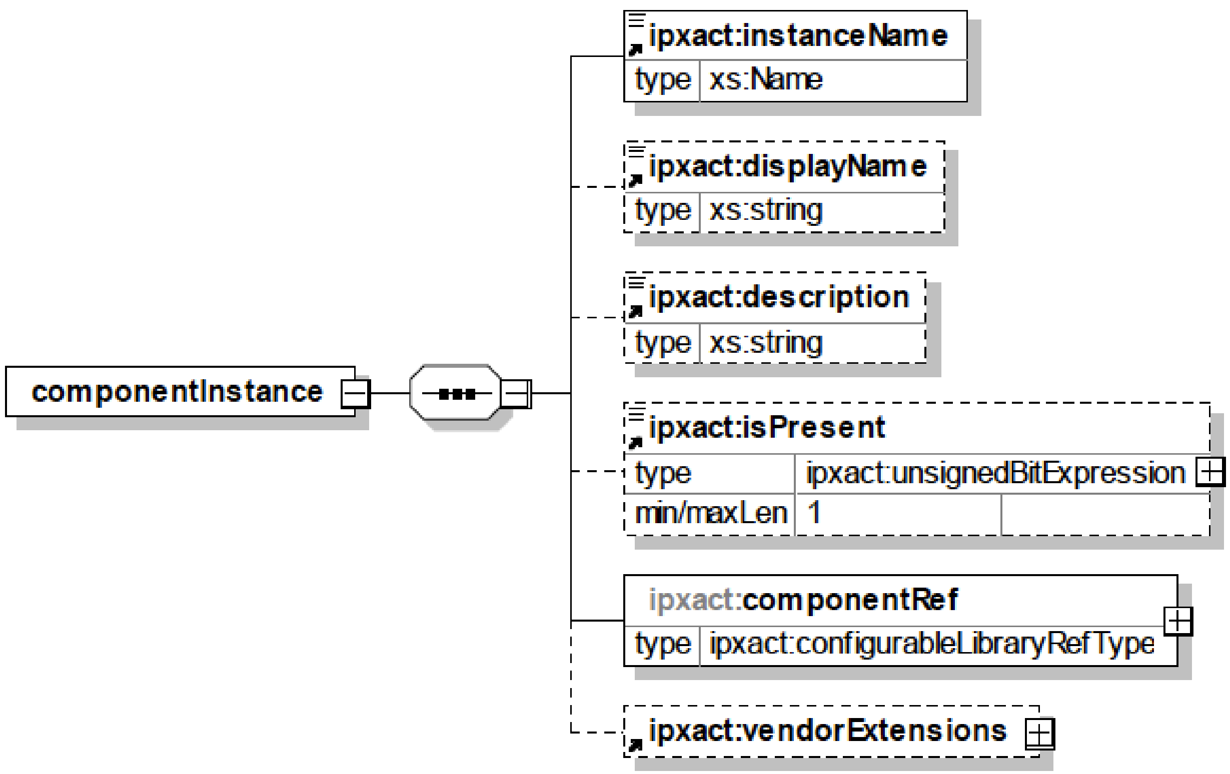
IEEE
3 Park Avenue
New York, NY 10016-5997
USA

IEEE Std 1685™-2014
(Revision of
IEEE Std 1685-2009)

Contents

IP-XACT Example: componentInstance

IEEE 1685 XSD



IEEE 1685 Description

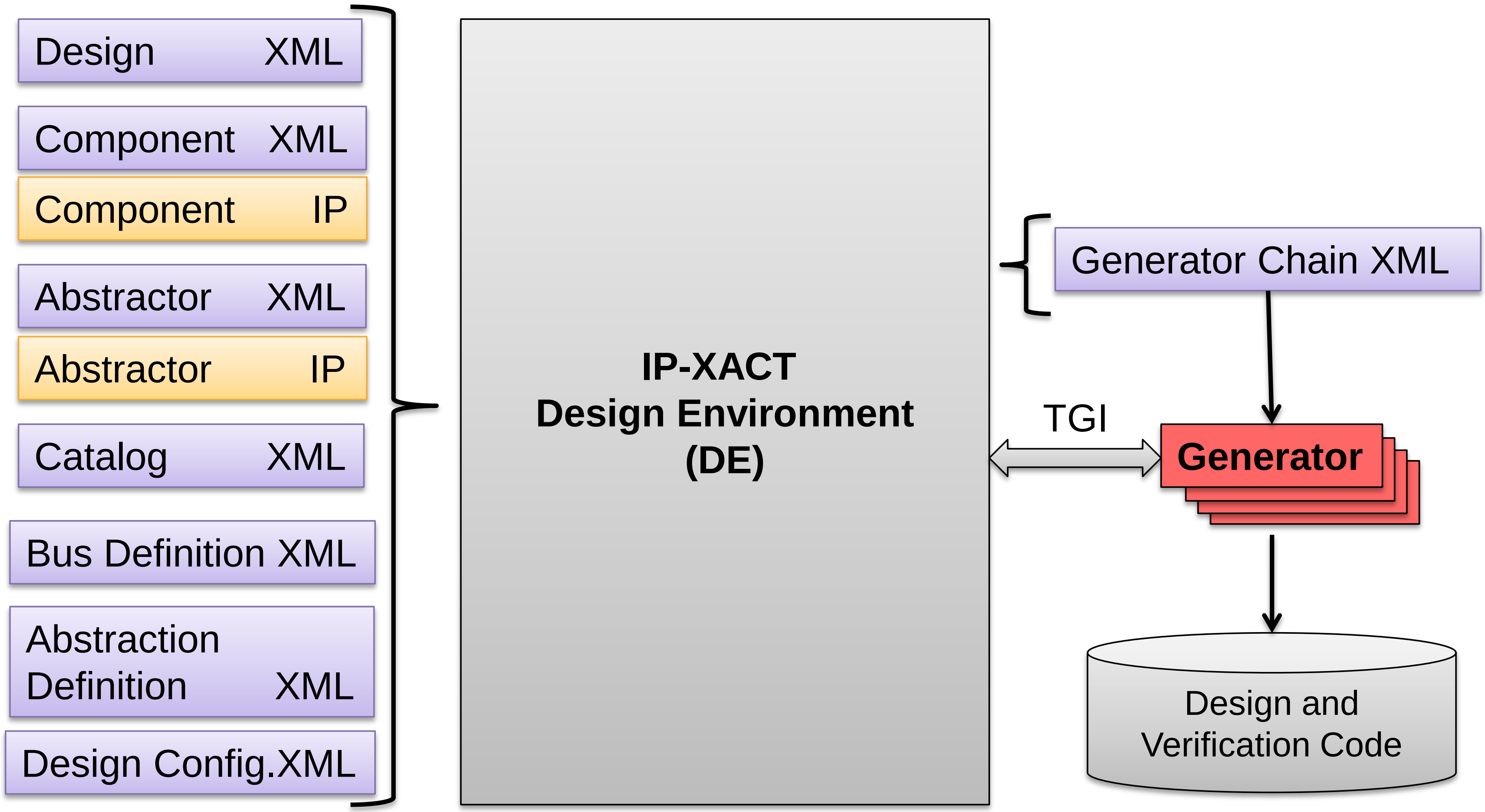
The **componentInstance** element documents the existence of a component in a design. This element contains the following subelements:

a) **instanceName** (mandatory; type: ***Name***) assigns a unique name for this instance of the component in this design. The value of this element shall be unique inside a **design** element.

…

f) **vendorExtensions** (optional) adds any extra vendor-specific data related to the design. See C.24.

IP-XACT Design Environment



IP-XACT Design Environment

- Design XML
- Component XML
- Component IP
- Abstractor XML
- Abstractor IP
- Catalog XML
- Bus Definition XML
- Abstraction Definition XML
- Design Config.XML

Some IP-XACT Objects

Design Configures component instances & interconnections (Netlist)

Component Describes IP's interfaces: Ports, bus interfaces with bus and abstraction type, address spaces, memory maps, registers, parameters, views, file sets, ...
IP stored in physical file as Verilog, VHDL, ...

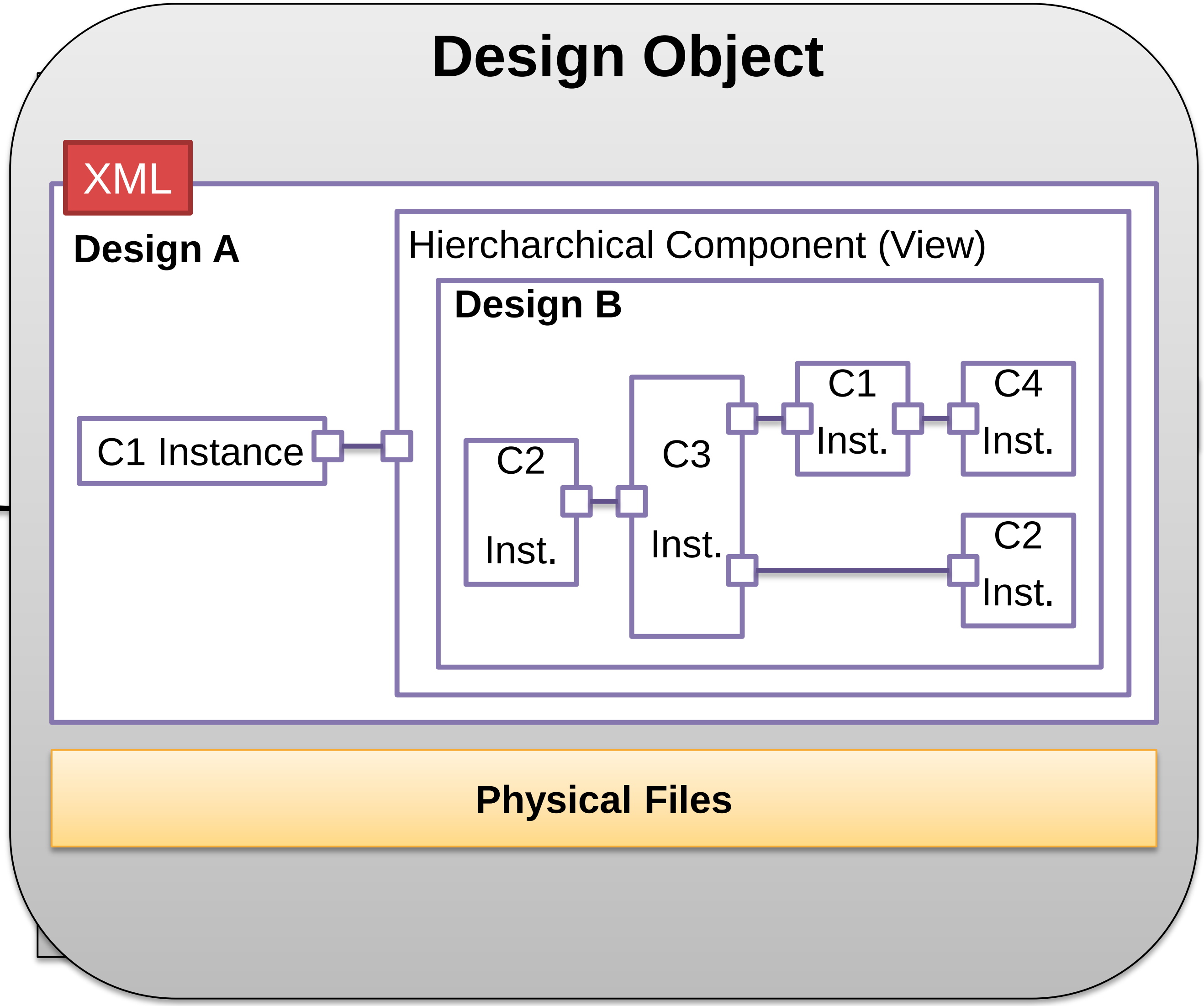
Bus Definition describes bus protocol

AbstractionDefinition describes bus on one abstraction layer e.g. RTL, TLM

References done by unique IP-XACT VLNV identification (Vendor Library Name Version)

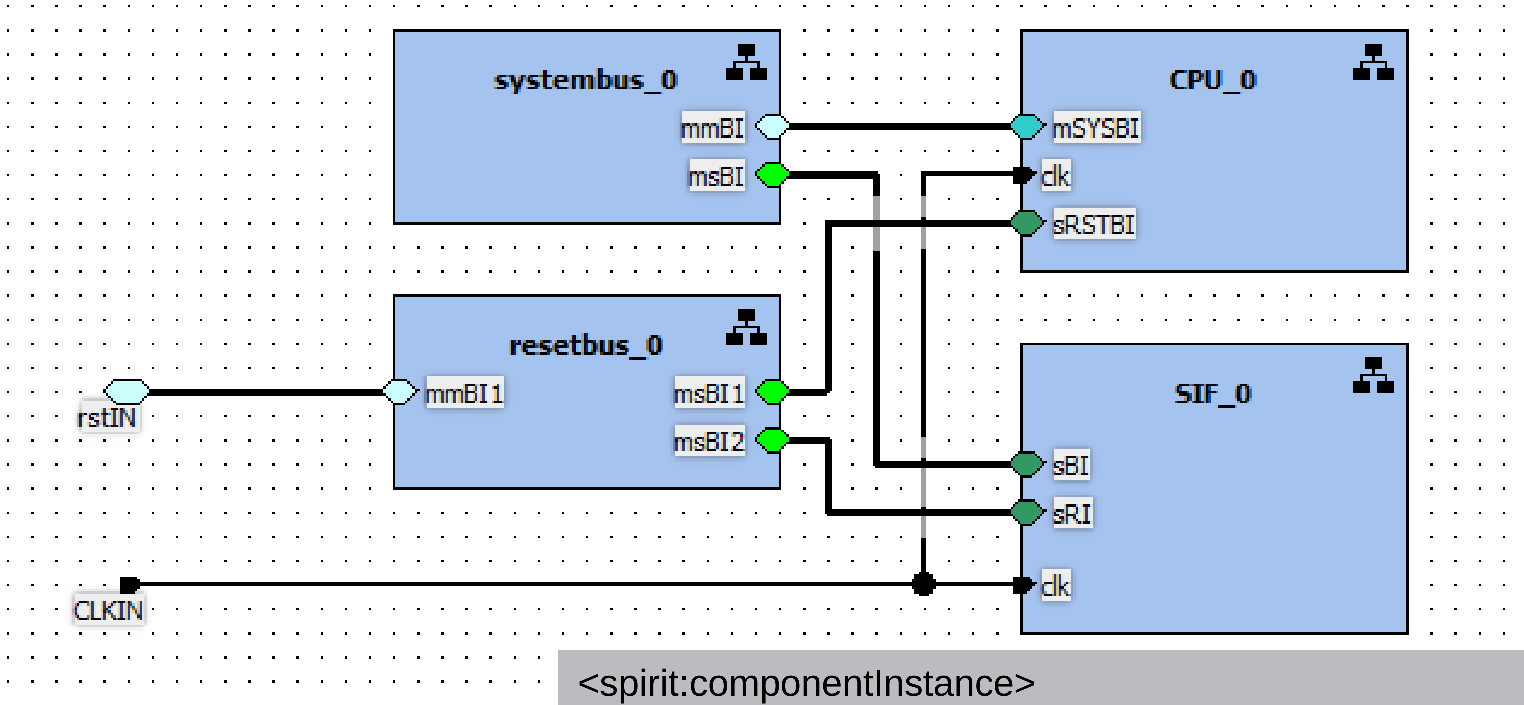
IP-XACT Design Environment

- Design XML
- Component XML
- Component IP
- Abstractor XML
- Abstractor IP
- Catalog XML
- Bus Definition XML
- Abstraction Definition XML
- Design Config.XML



IP-XACT Design Example

- Design with four components: CPU, Serial Interface (SIF), two buses (system, reset)




```
<spirit:componentInstance>
      <spirit:instanceName>SIF_0</spirit:instanceName>
      <spirit:displayName></spirit:displayName>
      <spirit:description></spirit:description>
      <spirit:componentRef spirit:vendor="TUM" spirit:library="components"
      spirit:name="SIF" spirit:version="1.0"/>
      …
```

- Sceenshot taken from Kactus2 tool (open source IP-XACT editor and code generator)

IP-XACT Component Example: SIF

- Version & text encoding
- Component tag w/ schema information
- Unique VLNV Identification
- Bus Interfaces
- Views, e.g, flat, hierarchical
- Extra Ports

```
<?xml version="1.0" encoding="UTF-8"?>
<spirit:component xmlns:kactus2="http://funbase.cs.tut.fi/"
xmlns:spirit="http://www.spiritconsortium.org/XMLSchema/SPIRIT/1.5"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.spiritconsortium.org/XMLSchema/SPIRIT/1.5
http://www.spiritconsortium.org/XMLSchema/SPIRIT/1.5/index.xsd">
<spirit:vendor>TUM</spirit:vendor>
<spirit:library>components</spirit:library>
<spirit:name>SIF</spirit:name>
<spirit:version>1.0</spirit:version>
<spirit:busInterfaces>
…
</spirit:busInterfaces>
<spirit:model>
        <spirit:views>
        …
        </spirit:views>
        <ports>
        …
        </ports>
</spirit:model>
</spirit:component>
```

IP-XACT Component Example: SIF - Bus Interfaces

Bus type with reference to VLNV of bus definition

Abstraction type with reference to VLNV of bus definition

Configuration, Slave/master, endianness

```
…
<spirit:busInterfaces>
<spirit:busInterface>
        <spirit:name>sBI</spirit:name>
        <spirit:busType spirit:vendor="TUM" spirit:library="bus" spirit:name="busif"
        spirit:version="1.0"/>
        <spirit:abstractionType spirit:vendor="TUM" spirit:library="bus"
        spirit:name="busif.absDef" spirit:version="1.0"/>
        <spirit:slave/>
        <spirit:connectionRequired>false</spirit:connectionRequired>
        …
        <spirit:endianness>little</spirit:endianness>
</spirit:busInterface>
<spirit:busInterface>
        <spirit:name>sRI</spirit:name>
        <spirit:busType spirit:vendor="TUM" spirit:library="bus" spirit:name="resif"
        spirit:version="1.0"/>
        <spirit:abstractionType spirit:vendor="TUM" spirit:library="bus"
        spirit:name="resif.absDef" spirit:version="1.0"/>
        <spirit:slave/>
        <spirit:connectionRequired>false</spirit:connectionRequired>
        …
        <spirit:endianness>little</spirit:endianness>
</spirit:busInterface>
</spirit:busInterfaces>
…
```

IP-XACT Component Example: SIF - Ports

Port name

Port is a wire
with direction “in“

Definition of wire

```
…
<spirit:ports>
<spirit:port>
        <spirit:name>clk</spirit:name>
        <spirit:wire spirit:allLogicalDirectionsAllowed="false">
                <spirit:direction>in</spirit:direction>
                <spirit:vector>
                <spirit:left>0</spirit:left>
                <spirit:right>0</spirit:right>
                </spirit:vector>
                <spirit:wireTypeDefs>
                <spirit:wireTypeDef>
                <spirit:typeName
                spirit:constrained="false">std_logic</spirit:typeName>
                <spirit:typeDefinition>IEEE.std_logic_1164.all</spirit:typeDefinition>
                …
                </spirit:wireTypeDef>
                </spirit:wireTypeDefs>
        </spirit:wire>
        …
</spirit:port>
</spirit:ports>
…
```

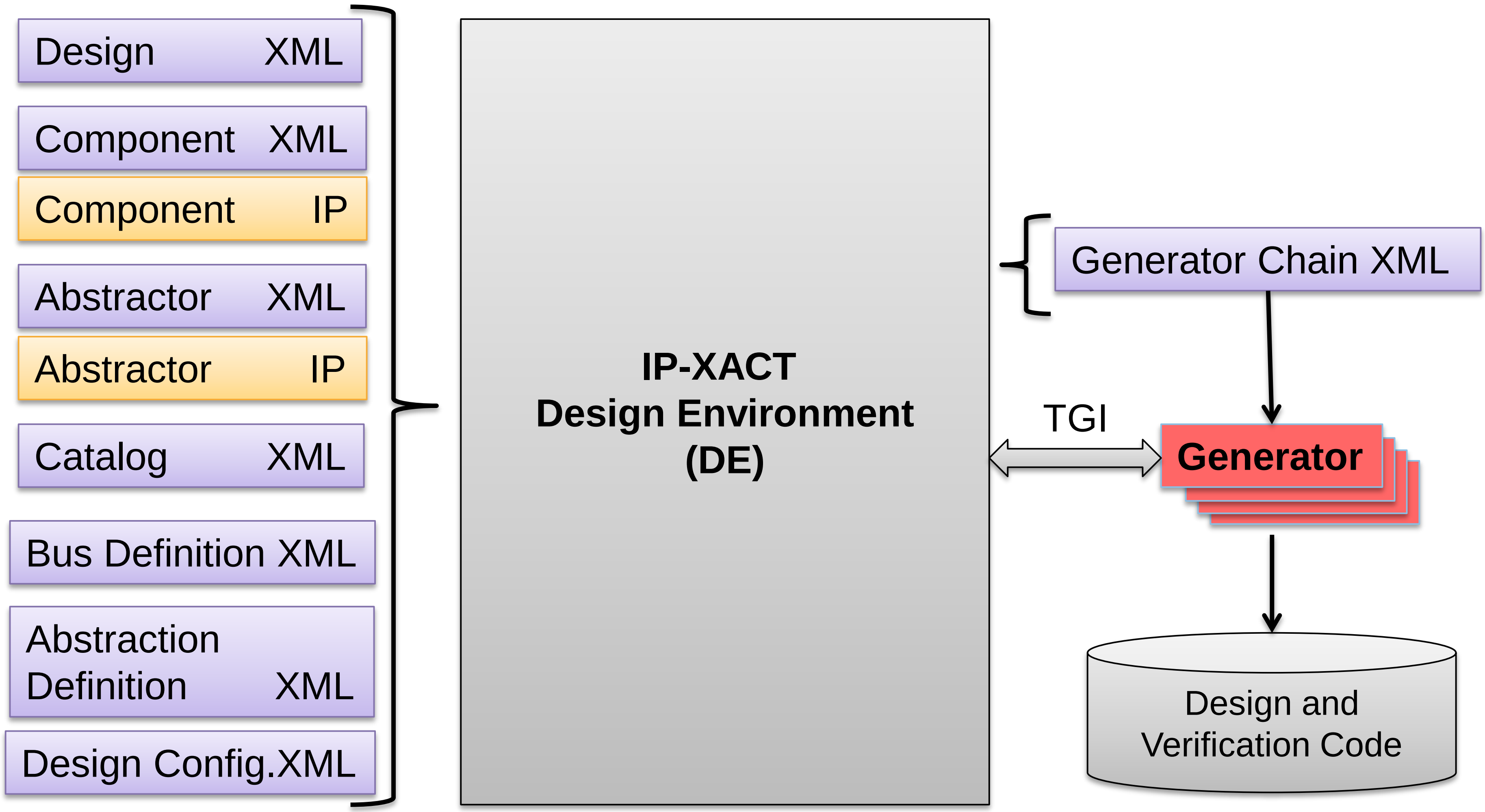
IP-XACT Design Environment

- Design XML
- Component XML
- Component IP
- Abstractor XML
- Abstractor IP
- Catalog XML
- Bus Definition XML
- Abstraction Definition XML
- Design Config.XML

Additional

Abstractor	converter between two bus interfaces of two abstraction types
Catalog	mapping of XP-XACT VLNV (Vendor Library Name Version) to physical file defining the object
Design Configuration	additional information for design / generator

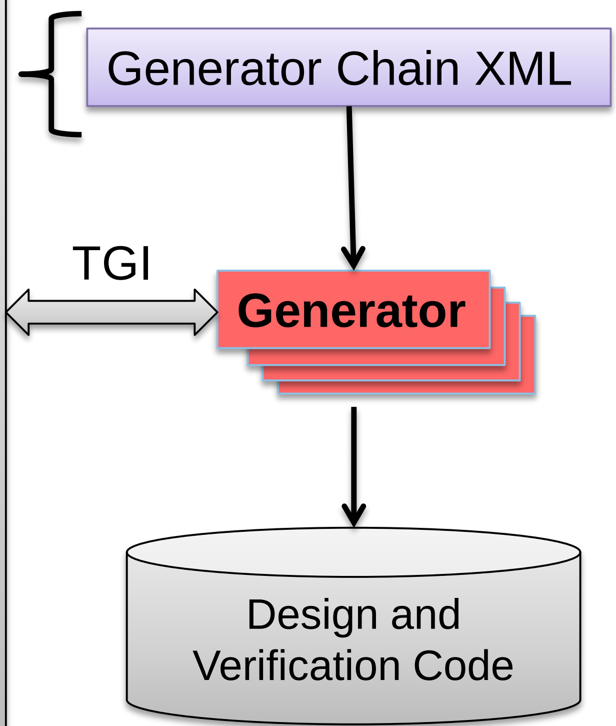
IP-XACT Design Environment



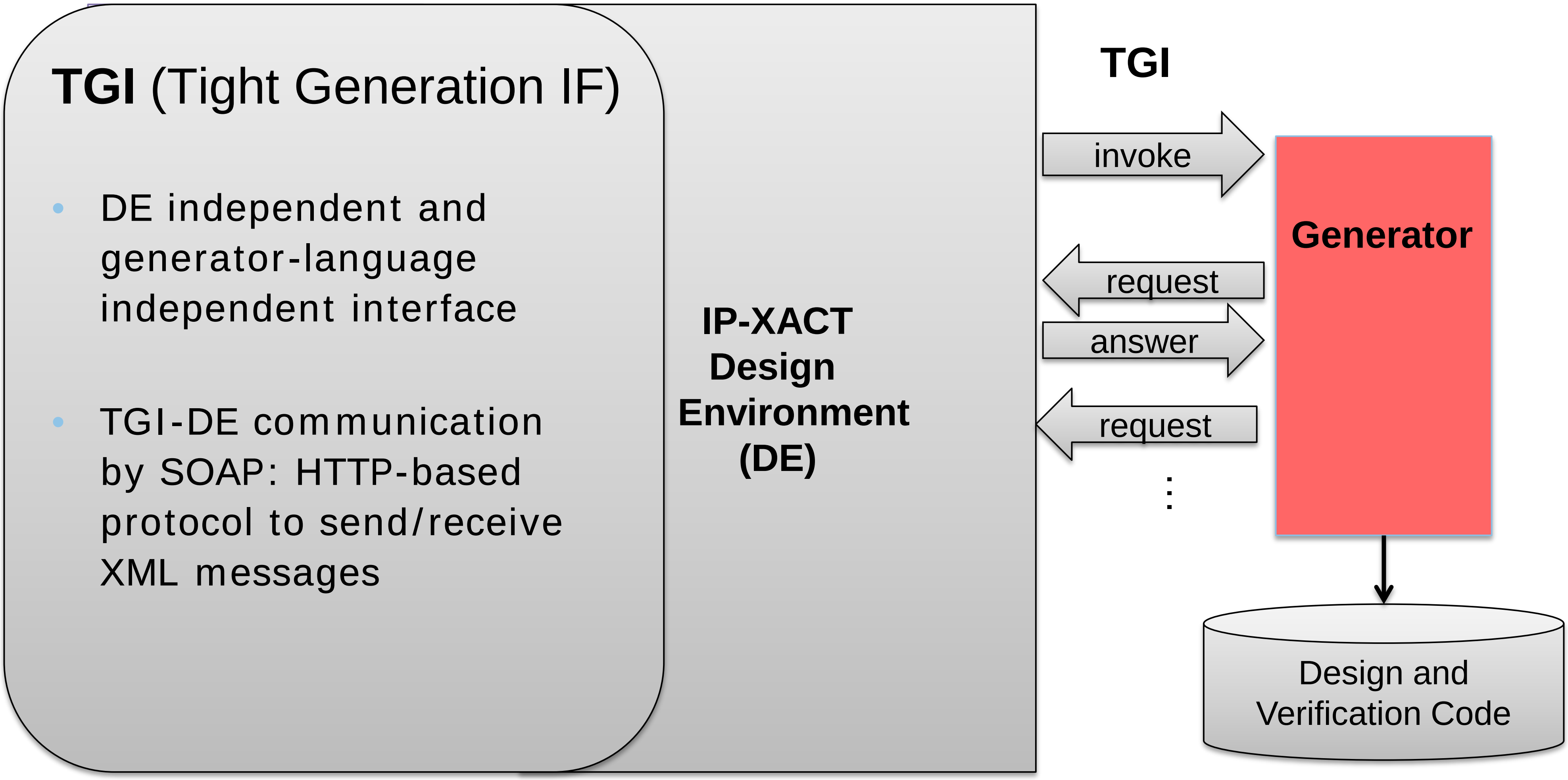
IP-XACT Design Environment

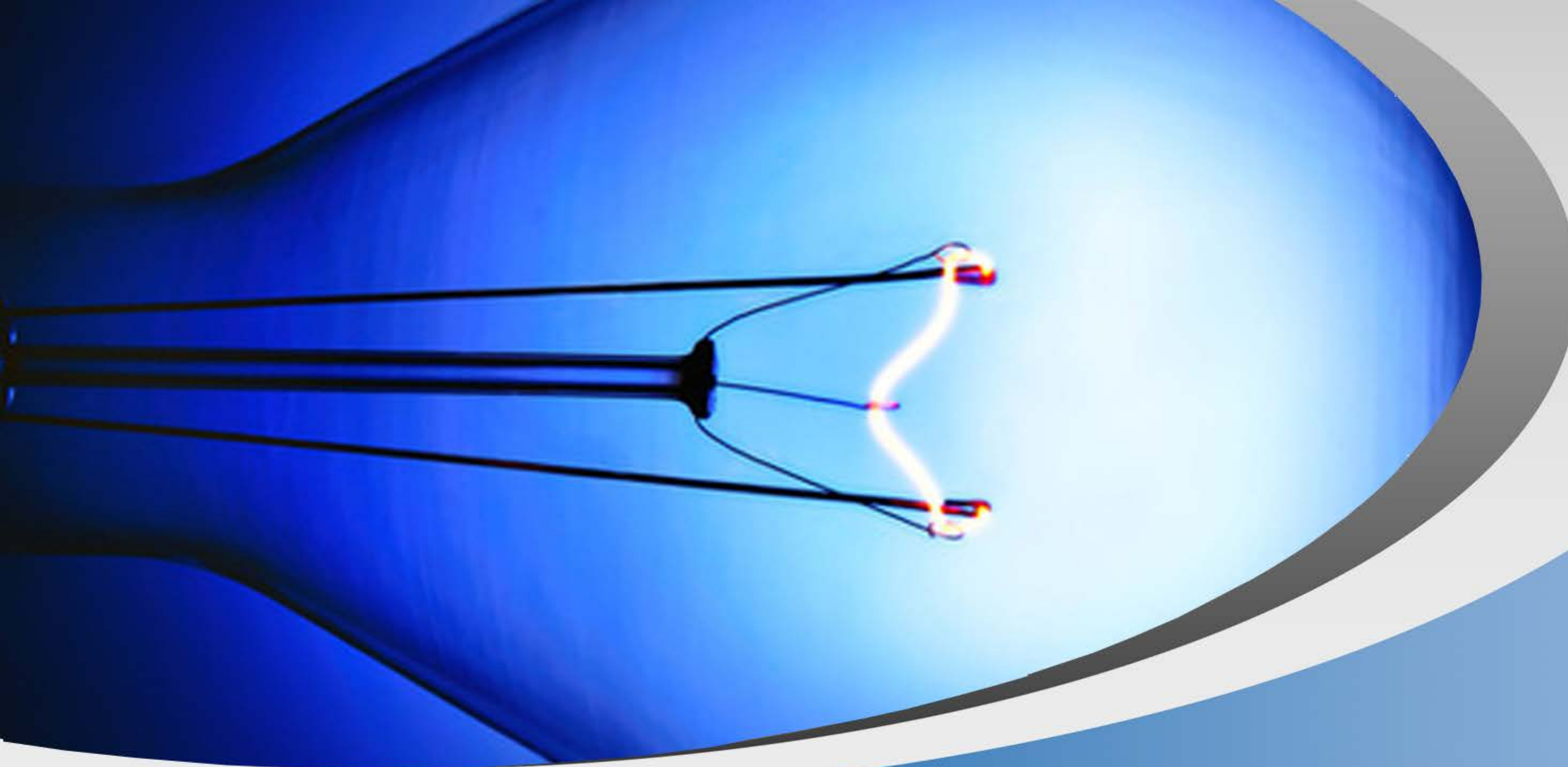
Generator

- program module processes IP-XACT XML and generates code
- Implementation can be in any language
  - XSLT (eXt. Stylesheet Language Transform.) language: XML ➔ other presentations
  - scripting language like Tcl, Python
  - programming language like Java, C++
- uses TGI (Tight Generation Interface) to access IP-XACT models



IP-XACT Design Environment





Thank you!