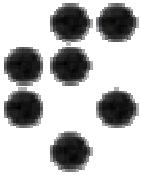


Jozef Stefan Institute

cosylab
CONTROL SYSTEM LABORATORY

Program Generators and Control System Software Development

Klemen Žagar (klemen.zagar@cosylab.com)
Anže Vodovnik (anze.vodovnik@cosylab.com)

Jozef Stefan Institute, Slovenia (http://kgb.ijs.si)
in cooperation with
Cosylab Ltd., Slovenia (http://www.cosylab.com)

Overview

- Why program generators?
- Describing ***what*** a control system is
- Describing ***how*** to build software for the control system
- Demos

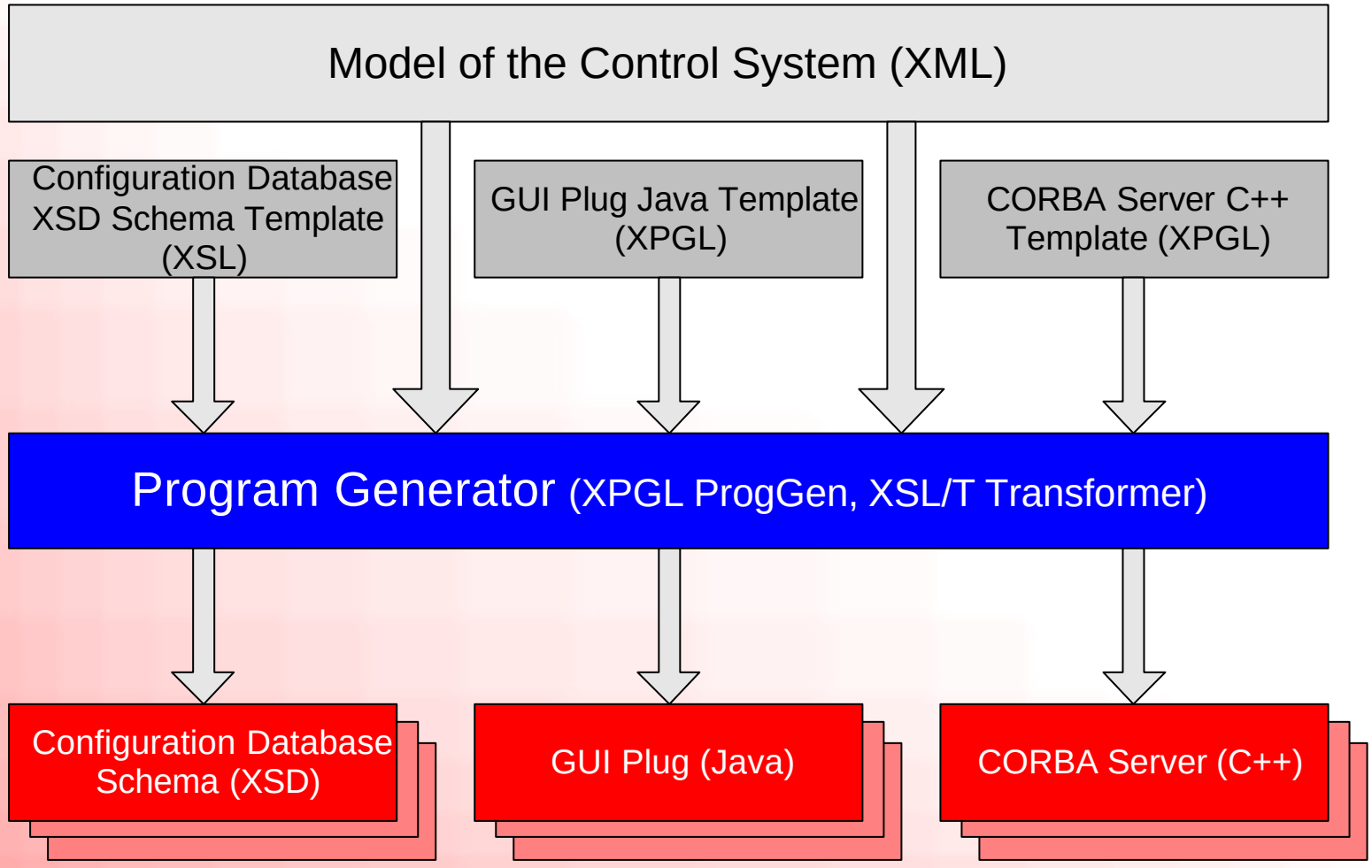
What is a program generator?

- Generates source code
  - C++, Java, XML, XML schema (XSD, ...), HTML
- Inputs:
  - Templates with placeholders
  - Values of placeholders
- Outputs
  - Useful code
- Comparison with Wizards:
  - Wizards offer interactive assistance to the developer
  - Not suitable for massive use
  - Allow abundant code customization

Need for program generators

- When there is a lot of repetitive code
- When there are many artifacts that need to be kept in-sync:
  - Device driver code
  - Networking protocol
  - Configuration database schema
  - GUI
- Many rules to follow during coding
- *Errare humanorum est*

How program generators work?



Describing what to build?

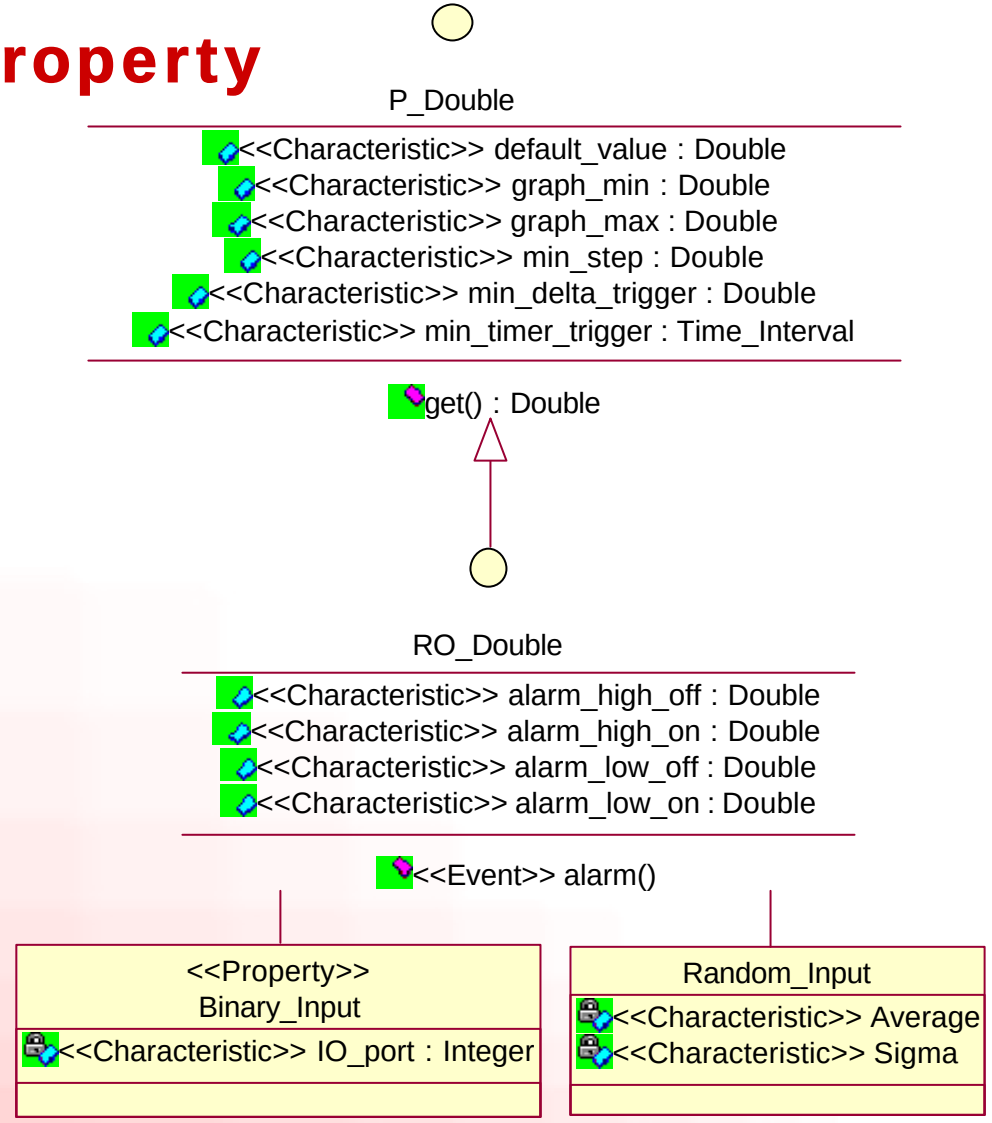
- Control System Modeling Language (CSML) (*work in progress*)
- Use of standard CASE tools
- Notation language (UML diagrams + XML representation)
  - Properties of entities
  - Relationships
  - Documentation
- Defining entities in a control system
  - Devices
  - Properties
  - Operations
  - Events
  - Characteristics

A Property

An observable in a control system
- Temperature
- Electric current

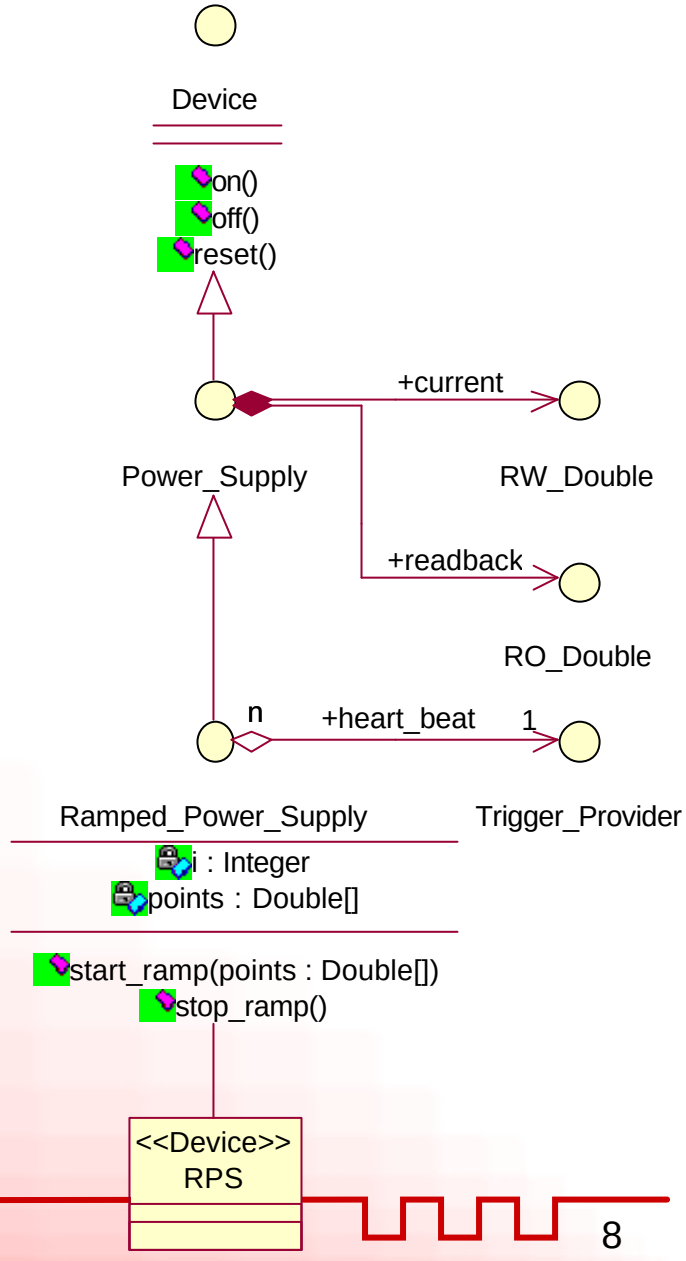
Data and behavior:
- Characteristics
- Events
- Operations

Interfaces vs. classes

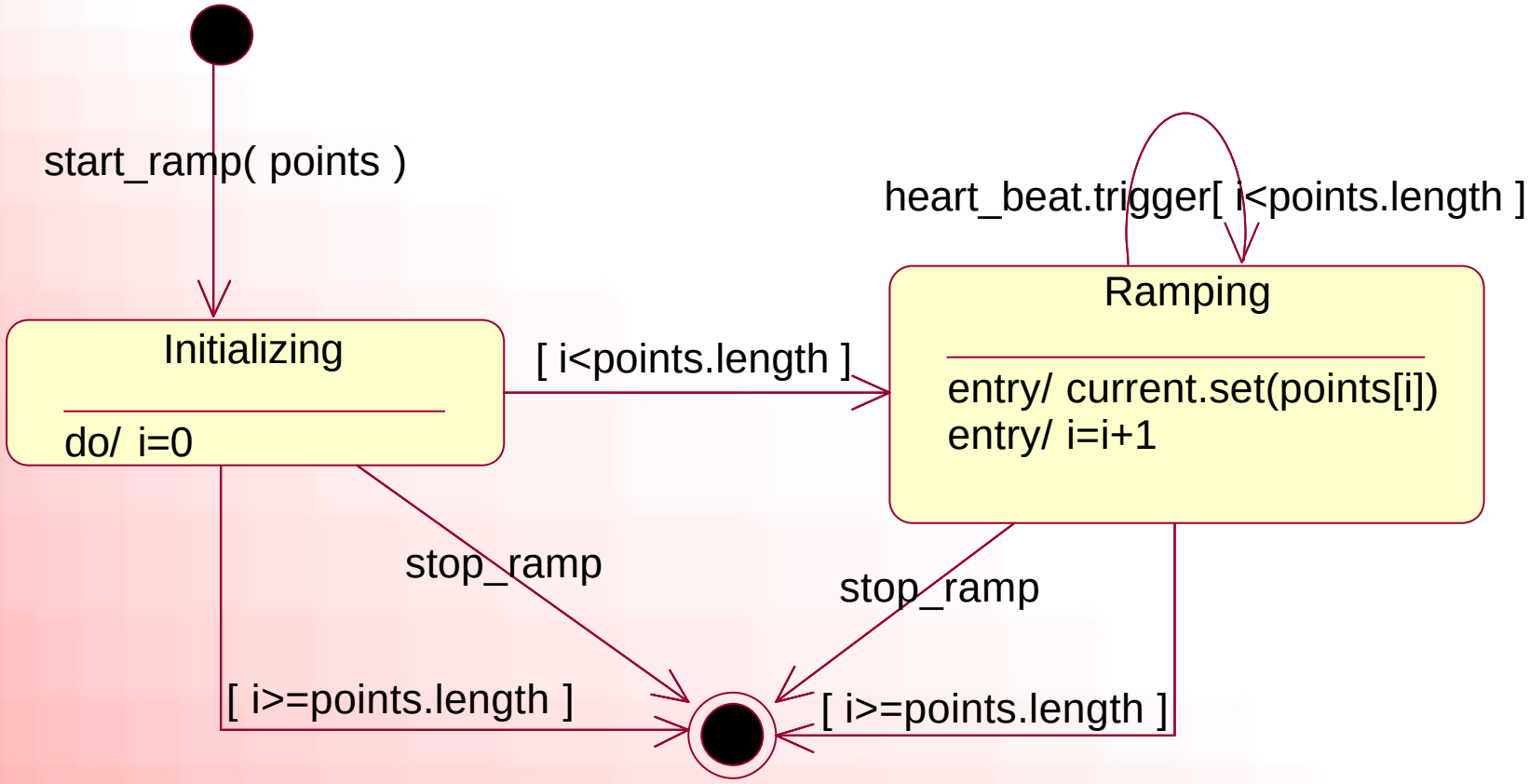


A Device

- Models a physical device
  - Power supplies
  - Antennas
  - Motors
  - Vacuum gauge
- May contain other devices
- May contain properties



Behavior: A State Machine



XML

- Extensible Markup Language
  - World Wide Web Consortium Standard
- XML representation of CSML model:
  - Physical representation of the model
  - CASE-tool neutral
  - Easy to manipulate using widely-available XML tools (parsers)
- Extensible Style-sheet Language Transformations
  - Standard way to transform XML into another XML or text
  - XSL/T files are hard to maintain

How to build?

- Software architects:
  - Decide on technology
  - Prescribe design patterns
  - Define classes and their relationships
- Programmers:
  - Translate the description of the system into source code using architect's instructions
  - In cases of high simplicity and repetition, programmers just apply templates

Extensible Program Generator Language (XPGL)

- Developed at IJS and Cosylab
  - Not a standard (yet ✍)
  - Open for influence
- Reuses open standards wherever possible
  - XSL/T expressions and tag names, XPath, ...
  - Generates code from XML documents (DOM)

What is XPGL?

- XPGL is not an instance of XML
  - In XPGL, white spaces are important
  - Unnecessary verbosity of XML
  - Custom parser and transformation tool was needed (*ProgGen*)
  - Only one configurable escape character
- XPGL simplifies frequently used constructs of XSL
- XPGL allows for several named output streams
- XPGL pays a lot of attention to spaces, so that the output can be neatly indented
- XPGL has provisions for preserving the output manually inserted by the programmer.

Example: our target

```
class SomeClass {
   private int myVariable;
   public int getMyVariable() {
      return myVariable;
   }
   public void setMyVariable(int value) {
      myVariable = value;
   }
};
```

Example: describing a class (XML)

```
<?xml version="1.0"?>
<class name=”some_class”>
  <field type=”int” name=”my_variable”/>
</class>
```

Example: the template (XPGL)

```
<?xpgl version="1.0"?>
class <”xpgl:naming('UU', /class/@name)”> {
  <for-each "/class/field">
    private <”@type”> <”xpgl:naming('LU', @name)”>;
    public <”@type”> get<”xpgl:naming('UU', @name)”>() {
      return <”xpgl:naming('LU', @name)”>;
    }
    public void set<”xpgl:naming('UU', @name)”>(<”@type”> value)  {
      <”xpgl:naming('LU', @name)”> = value;
    }
  </for-each>
};
```

Demos

1. Java accessor and mutator

2. Generating HTML

3. Complex example:
   - A server (“Power Supply”)
   - A GUI client
   - C#

Conclusion

- The entire system is defined in one place
  - E.g., the CSML model or XML
  - Includes documentation
- Other artifacts are kept in-sync with it
  - All other artifacts are generated
  - Generators are adjustable
- Coding effort greatly reduced
- Less error prone

Any questions?