Introduction on UML for Industrial Systems

Ansgar Radermacher / Asma Smaoui
ansgar.radermacher@cea.fr

Acknowledgments – contains material from my CEA colleagues
Shuai Li, Jérémie Tatibouët, François Terrier, Sébastien Gérard

Code generation from models (Model to Text)

1. **Code generation from models (Model to Text)**
2. Role of static profiles
3. Language xtend
4. Apply to simplified Python/Keras generator
5. Test/debug an Eclipse plugin
6. Revision / questions

Code generation from Models

- Motivation
  - Model = *executable* specification (not only documentation)
  - Provide (limited) early feedback
  - Map model elements to code

- Variants
  - Generate skeleton vs. generate full code
  - Synchronize model from code (roundtrip) vs. Code generation only

Recap: M2M and M2T transformations

- M2M = model to model transformation
  - Transform a source model into a target model

- Examples

  UML into another UML model (refinement)

  UML to an Ecore model

  …

Model to text (M2T) transformation

- Produce textual artefacts from a model, typically
  - Code
  - Formal verification languages
  - Documentation
  - Textual description languages (such as IDL)

- Here: only focus on code generation

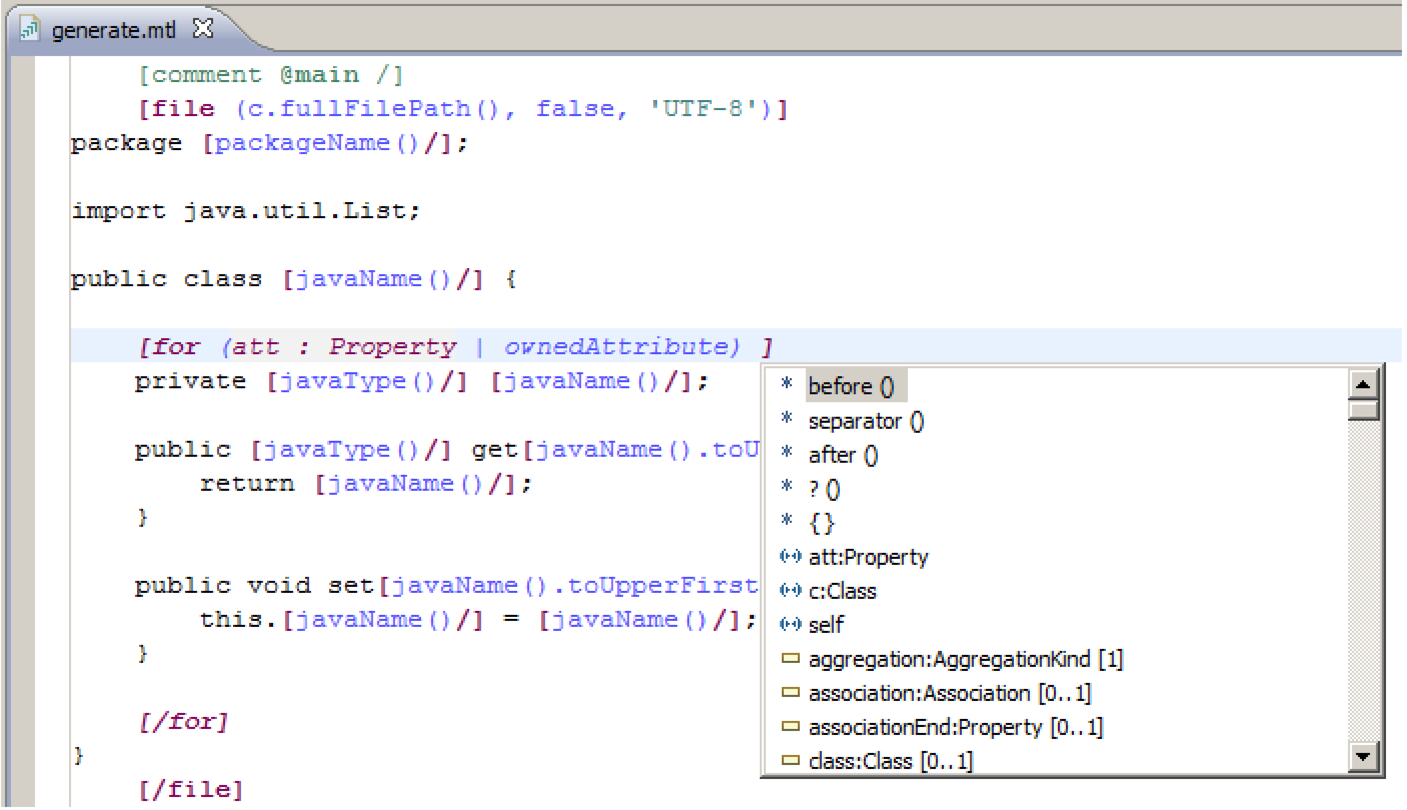
Implementation approaches

- Use QVT or ATL
- Papyrus SW designer
    - Combination of M2M and M2T transformations
      (quite simple code generators, advanced features via M2M)
- OMG M2T (https://www.omg.org/spec/MOFM2T)
- Specific languages – Acceleo and Xtend (more on Xtend later)

Acceleo

- “Generate anything from any EMF model”
- R&D result / support from OBEO

- Pragmatic
- Support for text templates
- Editor with assistance
- Interpreted language


generate.mtl
before ()
separator ()
after ()
? ()
{}
att:Property
c:Class
self
aggregation:AggregationKind [1]
association:Association [0..1]
associationEnd:Property [0..1]
class:Class [0..1]

```
    [comment @main /]
    [file (c.fullFilePath(), false, 'UTF-8')]
package [packageName()/];

import java.util.List;

public class [javaName()/] {

    [for (att : Property | ownedAttribute) ]
    private [javaType()/] [javaName()/];

    public [javaType()/] get[javaName().toU
        return [javaName()/];
    }

    public void set[javaName().toUpperFirst
        this.[javaName()/] = [javaName()/];
    }

    [/for]
}
    [/file]
```

Code generation from models (Model to Text)

1. **Code generation from models (Model to Text)**
2. **Role of static profiles**
3. **Language xtend**
4. **Apply to simplified Python/Keras generator**
5. **Test/debug an Eclipse plugin**
6. **Revision / questions**

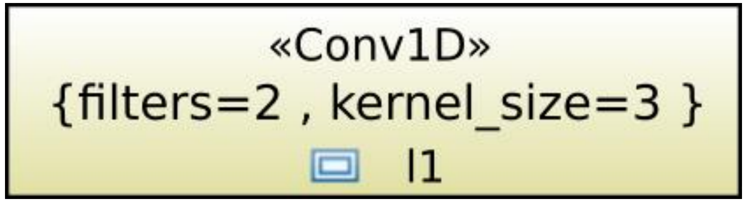
Static profiles - Introduction

- We defined a profile, now need programmatic access to its **attributes**

- “Normal” profile – access attributes via generic methods; provided by common superclass `Element`

  - `getAppliedStereotype(“stereotype-name”) : Stereotype`
  - `getValue(stereotype, “attribute-name”) : Object`

Standard profiles – Attribute access



«Conv1D»
{filters=2 , kernel_size=3 }
l1

```
Stereotype conv1D =
     l1.getAppliedStereotype(”SimpleNN::Conv1D”);
int filters = (int) l1.getValue(conv1D, ”filters”);
```

Quiz

? Disadvantage of generic stereotype access methods?

! Stereotype or attribute name misspelled/renamed
⇨ code still compiles, runtime error

! Generic method returns `Object`
⇨ need cast, no type safety!

Static profiles - Introduction

- We defined a profile, now need programmatic access to its **attributes**
- “Normal” profile – access attributes via generic methods
  - umlElement.getStereotype(“stereotype-name”)
  - umlElement.getValue(stereotype, “attribute-name”)
- Stereotype instance = Instance of a Java class?
- Ecore provides possibility to generate Java code from a (meta-) model

- Steps : UML => ecore model => code

Generated code …

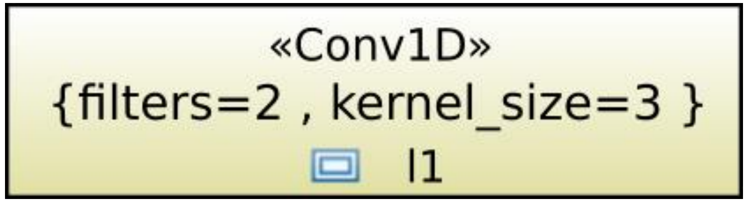
Generated interface reflects inheritance hierarchy

```
public interface Conv1D extends Layer {
   /**
    * Returns the value of the '<em><b>Filters</b></em>' attribute.
    * …
    * @generated
    */
   int getFilters();

   /**
    * Sets value of '{@link …SimpleNN.Conv1D#getFilters <em>Filters</em>}' attribute.
    * …
    * @generated
    */
   void setFilters(int value);
}
```

Getter/setter for attributes

Static profiles – Attribute access



«Conv1D»
{filters=2 , kernel_size=3 }
l1

```
Stereotype conv1D =
     l1.getAppliedStereotype(”SimpleNN::Conv1D”);
int filters = (int) l1.getValue(conv1D, ”filters”);
```

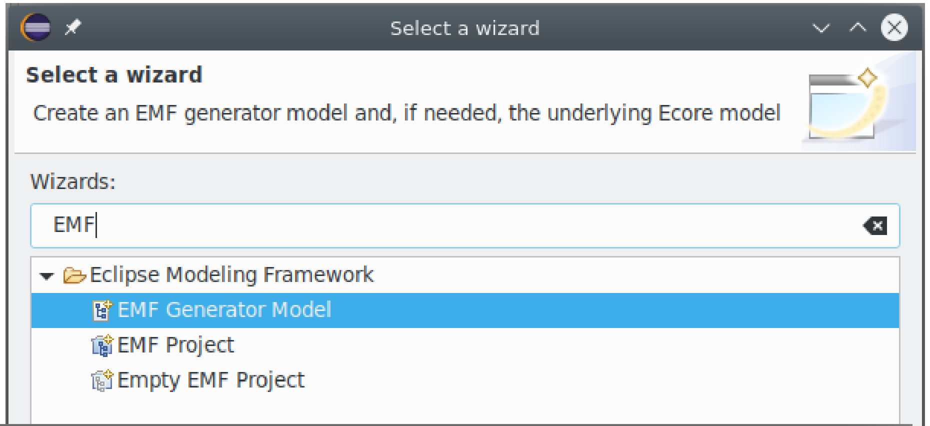
⇩

```
Conv1D conv1D =
     UMLUtil.getStereotypeApplication(l1, Conv1D.class);
int filters = conv1D.getFilters();
```

Type safe – return specific Conv1D interface

Exercise – generate a static profile

- Download plugin `org.eclipse.papyrus.simplenn.profile` from git
- Import the plugin into Eclipse workspace
- In sub-folder `profile`
  - Create new EMF Generator model
  - Use name SimpleNN.profile.genmodel
  - Import option: UML model
  - Load model SimpleNN.profile.uml
  - Precondition install UML2 extender SDK

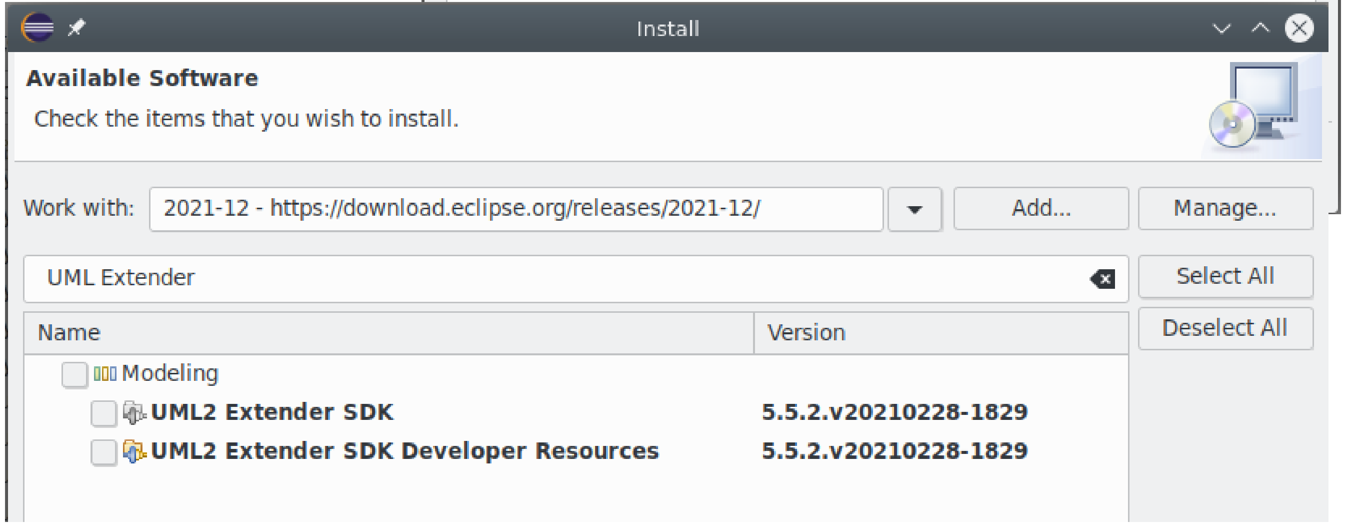


Select a wizard

Create an EMF generator model and, if needed, the underlying Ecore model

Wizards: EMF

- Eclipse Modeling Framework
  - EMF Generator Model
  - EMF Project
  - Empty EMF Project



Available Software

Check the items that you wish to install.

Work with: 2021-12 - https://download.eclipse.org/releases/2021-12/

UML Extender

Name	Version
□ Modeling	
□ UML2 Extender SDK	5.5.2.v20210228-1829
□ UML2 Extender SDK Developer Resources	5.5.2.v20210228-1829

Code generation from models (Model to Text)

1. Code generation from models (Model to Text)
2. Role of static profiles
3. **Language xtend**
4. Apply to simplified Python/Keras generator
5. Test/debug an Eclipse plugin
6. Revision / questions

Xtend language

- Java dialect, compiles into (readable) Java 8 source code.
- Use existing Java libraries seamlessly.
- Macros, lambdas, operator overloading and many more modern language features.
- Type inference – no need to type in most case
- Text Templates (that´s why we´re looking at it)

Type inference & syntactic sugar

- Replace
  `Type a = ...` with
  `var a = ...` or `val a =`

- The type with we infered from the expression on the right.
- Access “get” method as if an attribute – useful for generated MM methods
- Static extension, replace
  `UtilClass.fctA(a)` with
  `a.fctA`
- Clever indentation

Xtend – text templates

Shorthand for method getName()

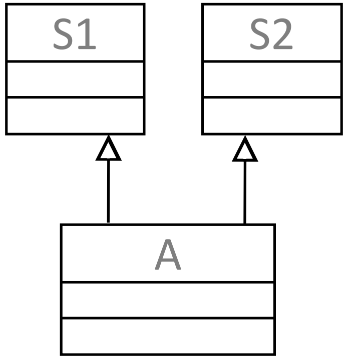
Shorthand for method getSuperClasses()

- Example
- '''class «c.name» extends

FOR loop

```
«FOR superclass : c. superclasses SEPARATOR “, “»
        «superclass.name»
«ENDFOR»''‘
```

Separator character

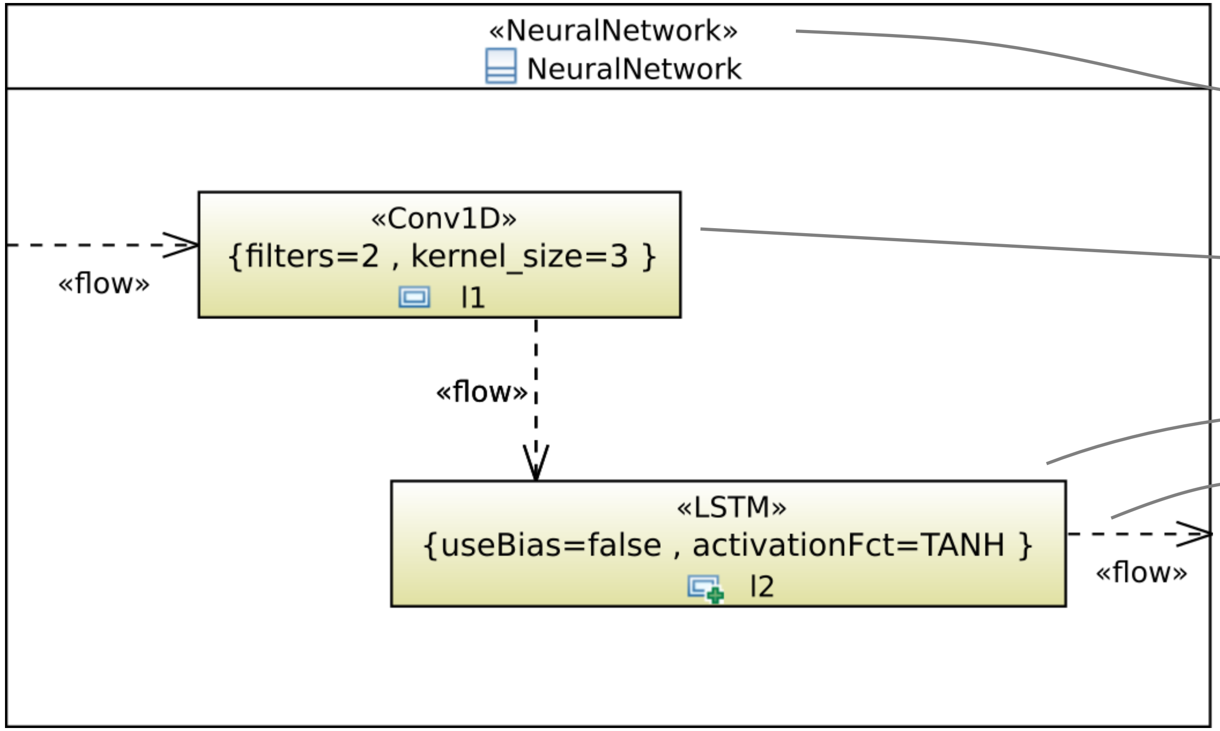


class A extends
S1, S2

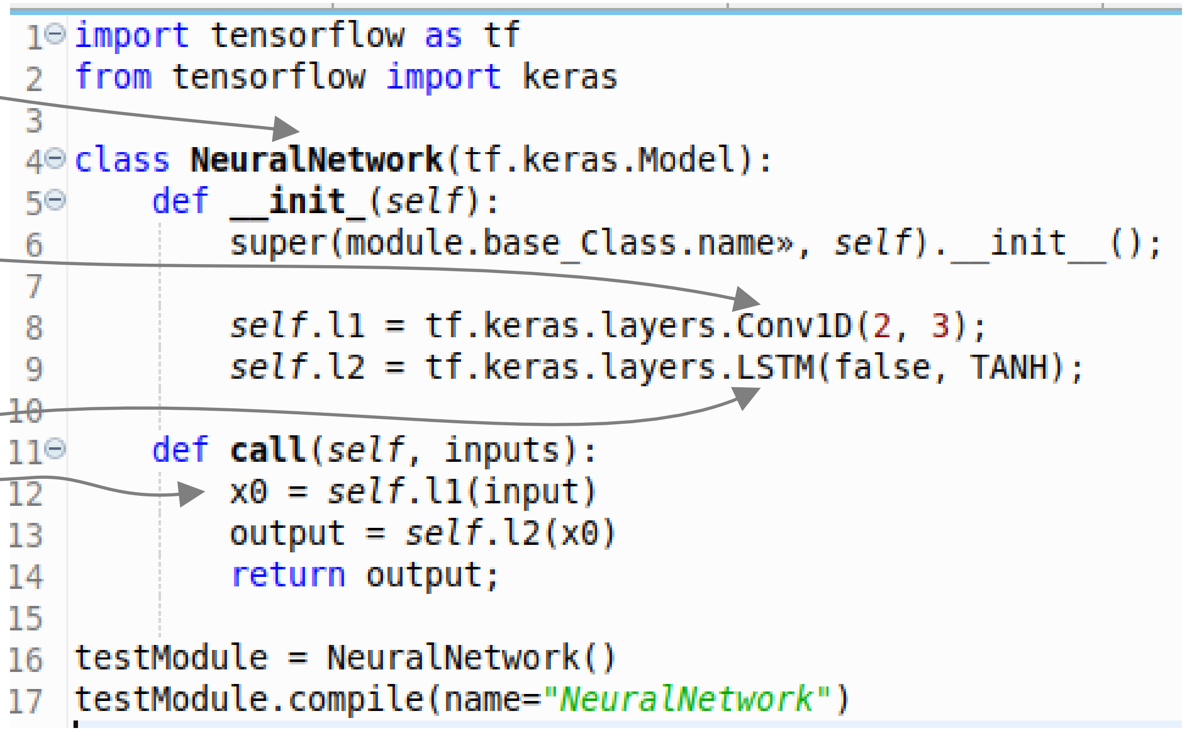
Code generation from models (Model to Text)

1. Code generation from models (Model to Text)
2. Role of static profiles
3. Language xtend
4. **Apply to simplified Python/Keras generator**
5. Test/debug an Eclipse plugin
6. Revision / questions
7. Homework

Application for Neural Network


«NeuralNetwork»
NeuralNetwork
«Conv1D»
{filters=2 , kernel_size=3 }
l1
«flow»
«flow»
«LSTM»
{useBias=false , activationFct=TANH }
l2
«flow»

input model (from cours2)
Stereotyped class with parts
composite structure diagram



```
import tensorflow as tf
from tensorflow import keras

class NeuralNetwork(tf.keras.Model):
    def __init_(self):
        super(module.base_Class.name», self).__init__();

        self.l1 = tf.keras.layers.Conv1D(2, 3);
        self.l2 = tf.keras.layers.LSTM(false, TANH);

    def call(self, inputs):
        x0 = self.l1(input)
        output = self.l2(x0)
        return output;

testModule = NeuralNetwork()
testModule.compile(name="NeuralNetwork")
```

wanted output
Python/Keras code

Code generator, part 1 (xtend)

```
/**
 * Generate a model with different elements necessary for the compilation
 */
static def genModule(NeuralNetwork module) '''
   import tensorflow as tf
   from tensorflow import keras

   class «module.base_Class.name»(tf.keras.Model):
      def __init_(self):
         super(module.base_Class.name», self).__init__();

         «FOR attribute : module.base_Class.attributes»
               «val layer = UMLUtil.getStereotypeApplication(attribute, Layer)»
               self.«attribute.name» = «layer.genLayer»
         «ENDFOR»
```

Text template

Access to (meta-) model

Control loop (over attributes)

Code generator, part 2 (xtend)

```
…
   def call(self, inputs):
      «var i = 0»
      «val flowList = module.base_Class.getFlows()»
      «FOR layer : flowList»
            «val previous = (i == 0) ? "input" : '''x«i-1»'''»
            «val current = (i == flowList.size-1) ? "output" : '''x«i++»'''»
            «current» = self.«layer.name»(«previous»)
      «ENDFOR»
      return output;

testModule = NeuralNetwork()
testModule.compile(name="NeuralNetwork")
```

Auxiliary function

input, intermediate, output

Code generator, part 3 (xtend)

```
…
// Generate each layer according to the model
static def genLayer(Layer layer) {
   if (layer instanceof LSTM) {
      genLstm(layer as LSTM)
   } else if (layer instanceof Conv1D) {
      genConvolutionLayer(layer as Conv1D)
   }
}
```

Use static profile

```
// Generate convolution layer according to the model
static def genConvolutionLayer(Conv1D conv1d) '''
   tf.keras.layers.Conv1D(«conv1d.filters», «conv1d.kernel_size»);
'''
```

Access attributes of static profile

Exercise

- Copy plugin `org.eclipse.papyrus.simplenn.gen.keras` into your workspace (and import it) – it´s on the git (tutos folder)
- Some information about Eclipse plugins (a bit off-topic)
  - OSGI MANIFEST.MF – plugin properties, dependencies
  - plugin.xml - Extension points & properties (for menus, etc.)

Acceleo vs. Xtend

- Text templates are relatively similar in both languages

- Advantages Acceleo
    - Dedicated to M2T, output file creation support
- Advantages Xtend
    - Compiled into Java => 1. Fast
      2. Use any Java function w/o specific declarations,
      3. Easy to debug with standard tooling
    - Standard language for code generators for Papyrus

What is round-trip engineering?

> **“The ability to automatically maintain the consistency of multiple, changing software artifacts, in software development environments/tools, is commonly referred to as round-trip engineering”**

- Related to traditional two software engineering disciplines:
  - Forward engineering: creating software from specifications
  - Reverse engineering: creating specifications from existing software
- Round-trip engineering adds synchronization of existing artifacts that evolved concurrently by incrementally updating each artifact to propagate changes made to the other artifact
- Round-trip generalizes both forward and reverse engineering

[Sendall04] S. Sendall and J. Küster. Taming model round-trip engineering. In *Proceedings of Workshop on Best Practices for Model-Driven Software Development*, Vancouver, Canada, 2004.

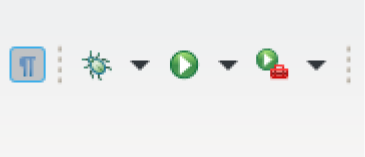
Code generation from models (Model to Text)

1. **Code generation from models (Model to Text)**
2. **Role of static profiles**
3. **Language xtend**
4. **Apply to simplified Python/Keras generator**
5. **Test/debug an Eclipse plugin**
6. **Revision / questions**

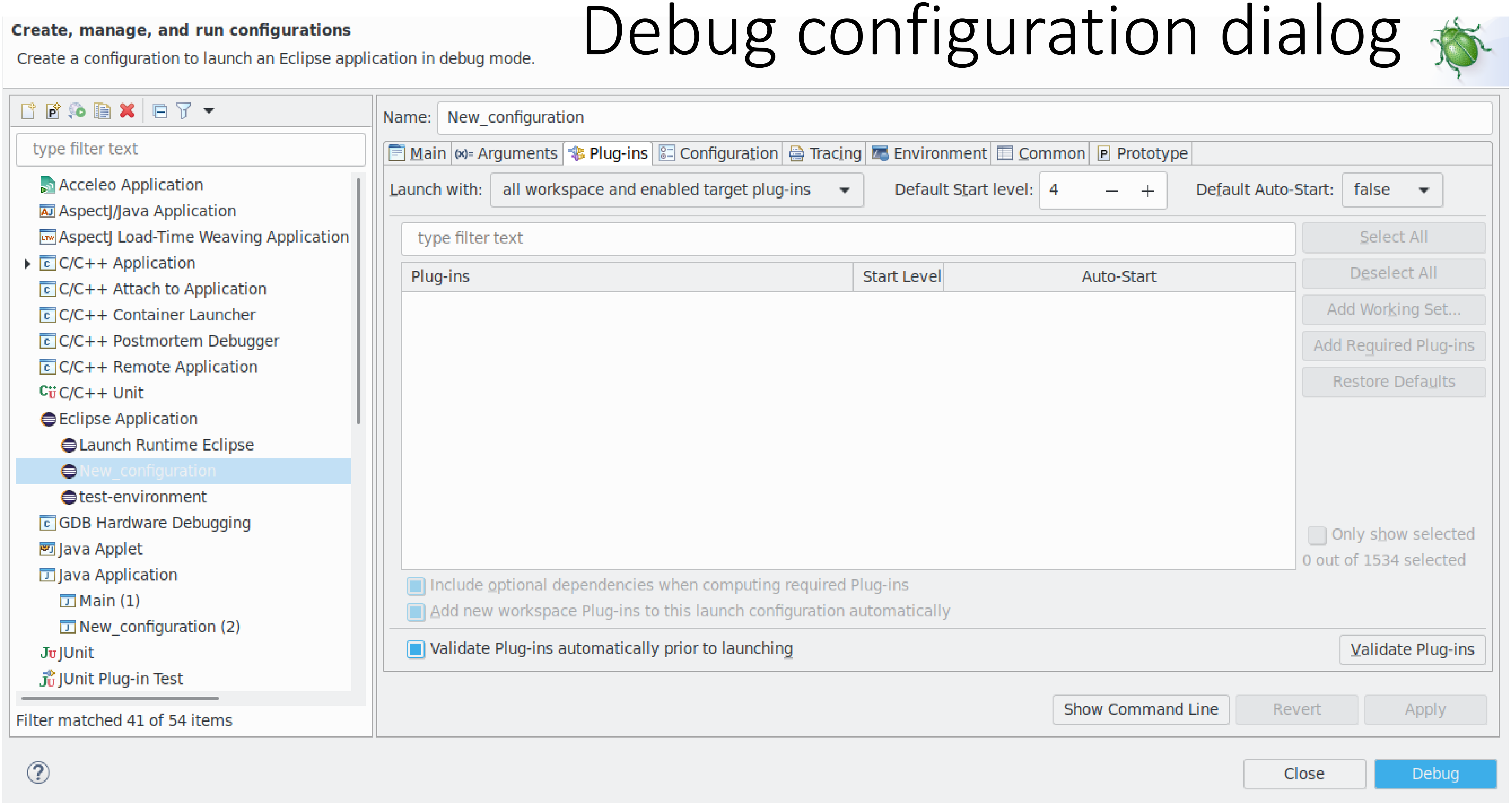
Debugging an Eclipse (workspace) plugin

- Eclipse only takes installed plugins into account
- But a ***new/2nd Eclipse*** instance will contain your workspace plugins
- Behaves like a normal Eclipse, runs in its own workspace

- Create a new Eclipse instance and run it
- Open the Debug Configurations dialog:
  - Menu **Run -> Debug configurations**
  - Via **Tool bar**



Debug configuration dialog



Create, manage, and run configurations

Create a configuration to launch an Eclipse application in debug mode.

type filter text

- Acceleo Application
- AspectJ/Java Application
- AspectJ Load-Time Weaving Application
- C/C++ Application
- C/C++ Attach to Application
- C/C++ Container Launcher
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- C/C++ Unit
- Eclipse Application
  - Launch Runtime Eclipse
  - New_configuration
  - test-environment
- GDB Hardware Debugging
- Java Applet
- Java Application
  - Main (1)
  - New_configuration (2)
- JUnit
- JUnit Plug-in Test

Filter matched 41 of 54 items

Name: New_configuration

Launch with: all workspace and enabled target plug-ins

Default Start level: 4

Default Auto-Start: false

type filter text

Plug-ins	Start Level	Auto-Start

□ Only show selected

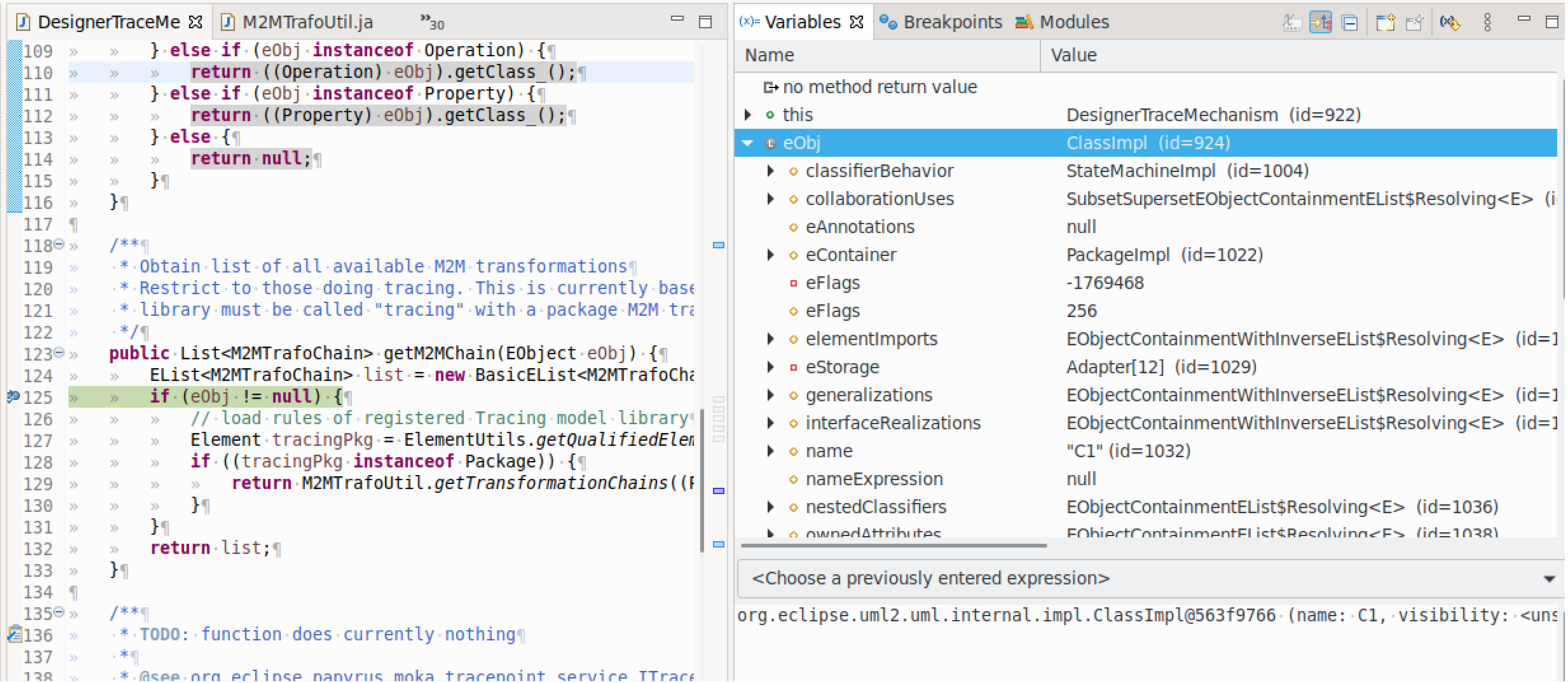
0 out of 1534 selected

☑ Include optional dependencies when computing required Plug-ins

☑ Add new workspace Plug-ins to this launch configuration automatically

☑ Validate Plug-ins automatically prior to launching

Place a breakpoint, inspect variables



```
		} else if (eObj instanceof Operation) {
			return ((Operation) eObj).getClass_();
		} else if (eObj instanceof Property) {
			return ((Property) eObj).getClass_();
		} else {
			return null;
		}
	}

	/**
	 * Obtain list of all available M2M transformations
	 * Restrict to those doing tracing. This is currently base
	 * library must be called "tracing" with a package M2M tra
	 */
	public List<M2MTrafoChain> getM2MChain(EObject eObj) {
		EList<M2MTrafoChain> list = new BasicEList<M2MTrafoCha
		if (eObj != null) {
			// load rules of registered Tracing model library
			Element tracingPkg = ElementUtils.getQualifiedElem
			if ((tracingPkg instanceof Package)) {
				return M2MTrafoUtil.getTransformationChains((F
			}
		}
		return list;
	}

	/**
	 * TODO: function does currently nothing
	 *
	 * @see org.eclipse.papyrus.moka.tracepoint.service.ITrace
```

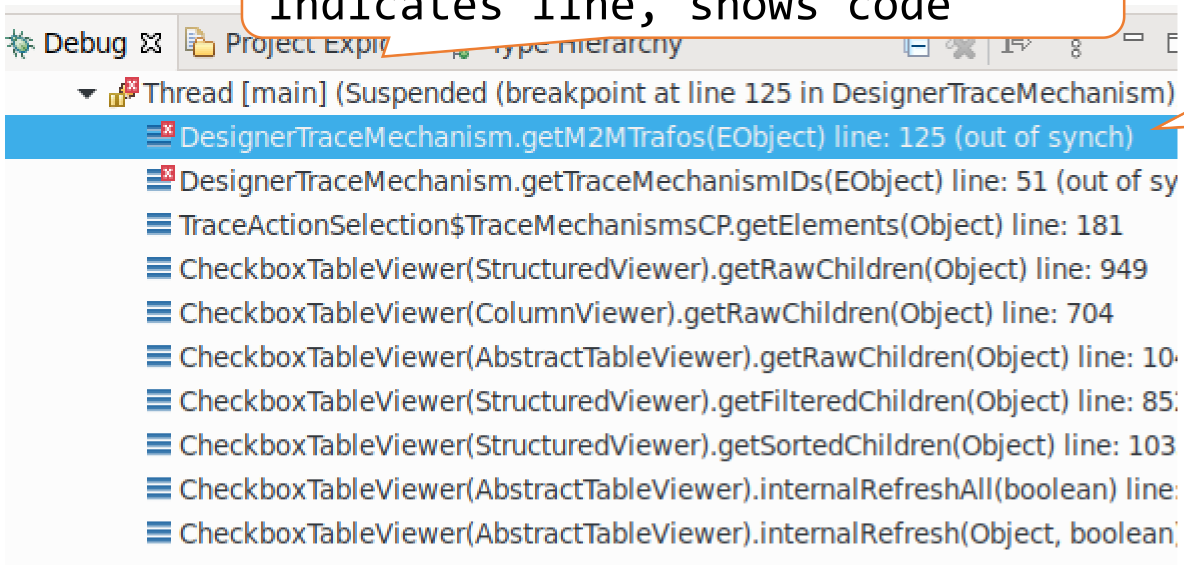
Name	Value
no method return value	
this	DesignerTraceMechanism (id=922)
eObj	ClassImpl (id=924)
classifierBehavior	StateMachineImpl (id=1004)
collaborationUses	SubsetSupersetEObjectContainmentEList$Resolving<E> (i
eAnnotations	null
eContainer	PackageImpl (id=1022)
eFlags	-1769468
eFlags	256
elementImports	EObjectContainmentWithInverseEList$Resolving<E> (id=1
eStorage	Adapter[12] (id=1029)
generalizations	EObjectContainmentWithInverseEList$Resolving<E> (id=1
interfaceRealizations	EObjectContainmentWithInverseEList$Resolving<E> (id=1
name	"C1" (id=1032)
nameExpression	null
nestedClassifiers	EObjectContainmentEList$Resolving<E> (id=1036)
ownedAttributes	EObjectContainmentEList$Resolving<E> (id=1038)

<Choose a previously entered expression>

```
org.eclipse.uml2.uml.internal.impl.ClassImpl@563f9766 (name: C1, visibility: <uns
```

Stack trace

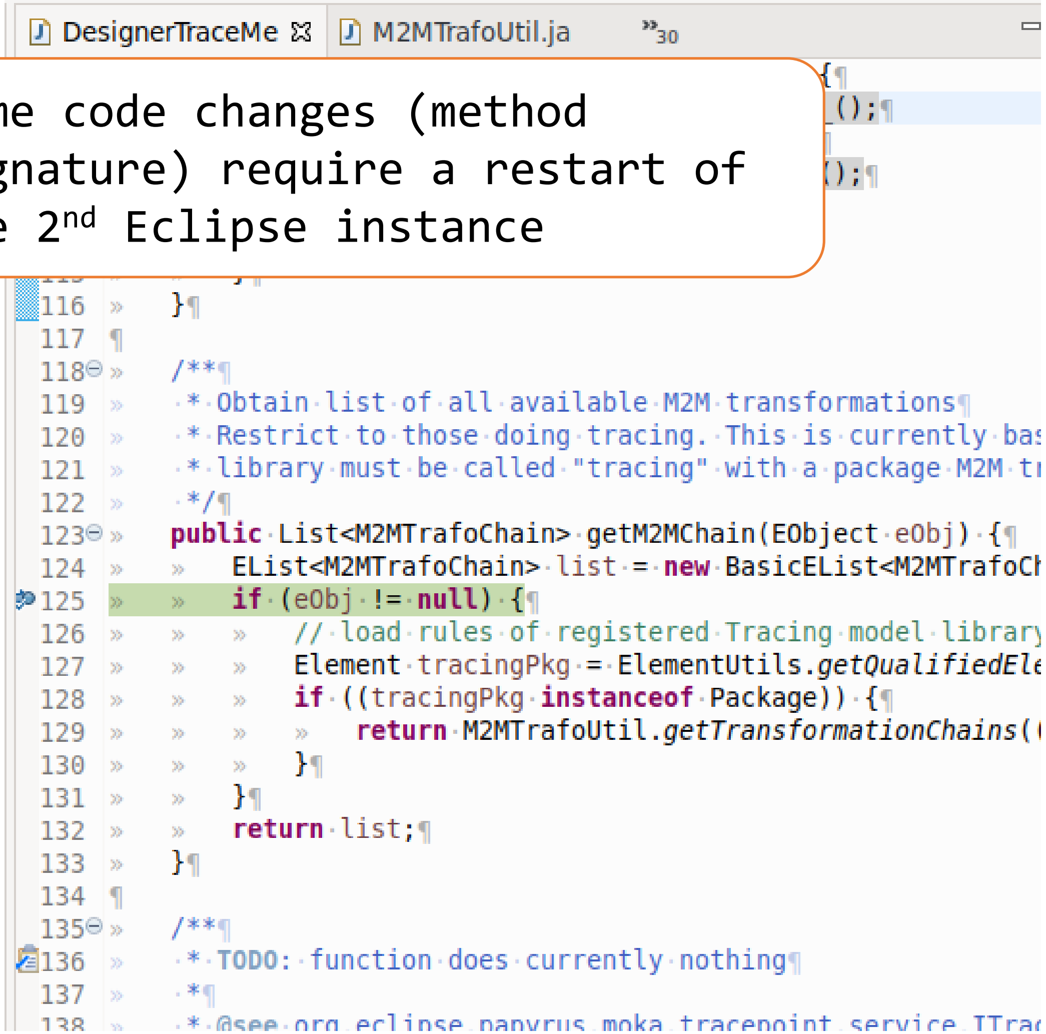
Debugger stopped on breakpoint, indicates line, shows code



```
Thread [main] (Suspended (breakpoint at line 125 in DesignerTraceMechanism)
    DesignerTraceMechanism.getM2MTrafos(EObject) line: 125 (out of synch)
    DesignerTraceMechanism.getTraceMechanismIDs(EObject) line: 51 (out of sy
    TraceActionSelection$TraceMechanismsCP.getElements(Object) line: 181
    CheckboxTableViewer(StructuredViewer).getRawChildren(Object) line: 949
    CheckboxTableViewer(ColumnViewer).getRawChildren(Object) line: 704
    CheckboxTableViewer(AbstractTableViewer).getRawChildren(Object) line: 10
    CheckboxTableViewer(StructuredViewer).getFilteredChildren(Object) line: 85
    CheckboxTableViewer(StructuredViewer).getSortedChildren(Object) line: 103
    CheckboxTableViewer(AbstractTableViewer).internalRefreshAll(boolean) line
    CheckboxTableViewer(AbstractTableViewer).internalRefresh(Object, boolean
```

User can navigate to any element in the hierarchy

Some code changes (method signature) require a restart of the 2nd Eclipse instance



```
116        }
117
118        /**
119         * Obtain list of all available M2M transformations
120         * Restrict to those doing tracing. This is currently bas
121         * library must be called "tracing" with a package M2M tr
122         */
123        public List<M2MTrafoChain> getM2MChain(EObject eObj) {
124            EList<M2MTrafoChain> list = new BasicEList<M2MTrafoCh
125            if (eObj != null) {
126                // load rules of registered Tracing model library
127                Element tracingPkg = ElementUtils.getQualifiedEle
128                if ((tracingPkg instanceof Package)) {
129                    return M2MTrafoUtil.getTransformationChains(
130                }
131            }
132            return list;
133        }
134
135        /**
136         * TODO: function does currently nothing
137         *
138         * @see org.eclipse.papyrus.moka.tracepoint.service.ITrac
```

Code generation from models (Model to Text)

1. **Code generation from models (Model to Text)**
2. **Role of static profiles**
3. **Language xtend**
4. **Apply to simplified Python/Keras generator**
5. **Test/debug an Eclipse plugin**
6. **Revision / questions**

Revision – What we´ve learned ...

- Motivation – why modeling?
- Domain specific vs. general purpose languages
- Textual vs. graphical modeling languages
- Meta-models
- UML diagrams
  - Global functions – Use case diagram
  - Interactions modeling – sequence diagrams
  - Different modes/states – state machine diagram
  - Component-based modeling – composite structure diagram
  - Behavior – Activity diagrams

Revision – We´ve learned ...

- Domain specific (modeling) languages – DSLs / DSMLs
  - Editor generation with xtext
- Domain specific modeling with UML
  - Real-time and embedded systems – MARTE profile
  - Extend and Restrict the language – UML profiles with OCL rules
  - Exploit the language for code generation – static profiles
- Model transformations and code generation
  - OMG QVT and ATL
  - Code generation (running/debugging plugins)
- Practical use of UML in Eclipse/Papyrus, debugging

Papyrus (for Robotics)

Papyrus for Robotics – customization for the robotics domain

- https://eclipse.org/papyrus/components/robotics
- Youtube channel: https://www.youtube.com/c/PapyrusEclipseUML
- RobMoSys aligned – European project https://robmosys.eu/

Papyrus Use Case Story

" A CONSOLIDATED EU DIGITAL INDUSTRIAL PLATFORM FOR ROBOTICS "

[An enabler]

Papyrus4Robotics, a RobMoSys-Compliant Toolset for Composable Models and Software

The Business

RobMoSys envisions an integration approach built on top of current code-centric platforms by means of systematic development and application of modern model-driven methods and tools. This enables system-of-systems type of integration, which occurs at all levels of abstraction and interaction, rather than merely at the code level.

The Challenge

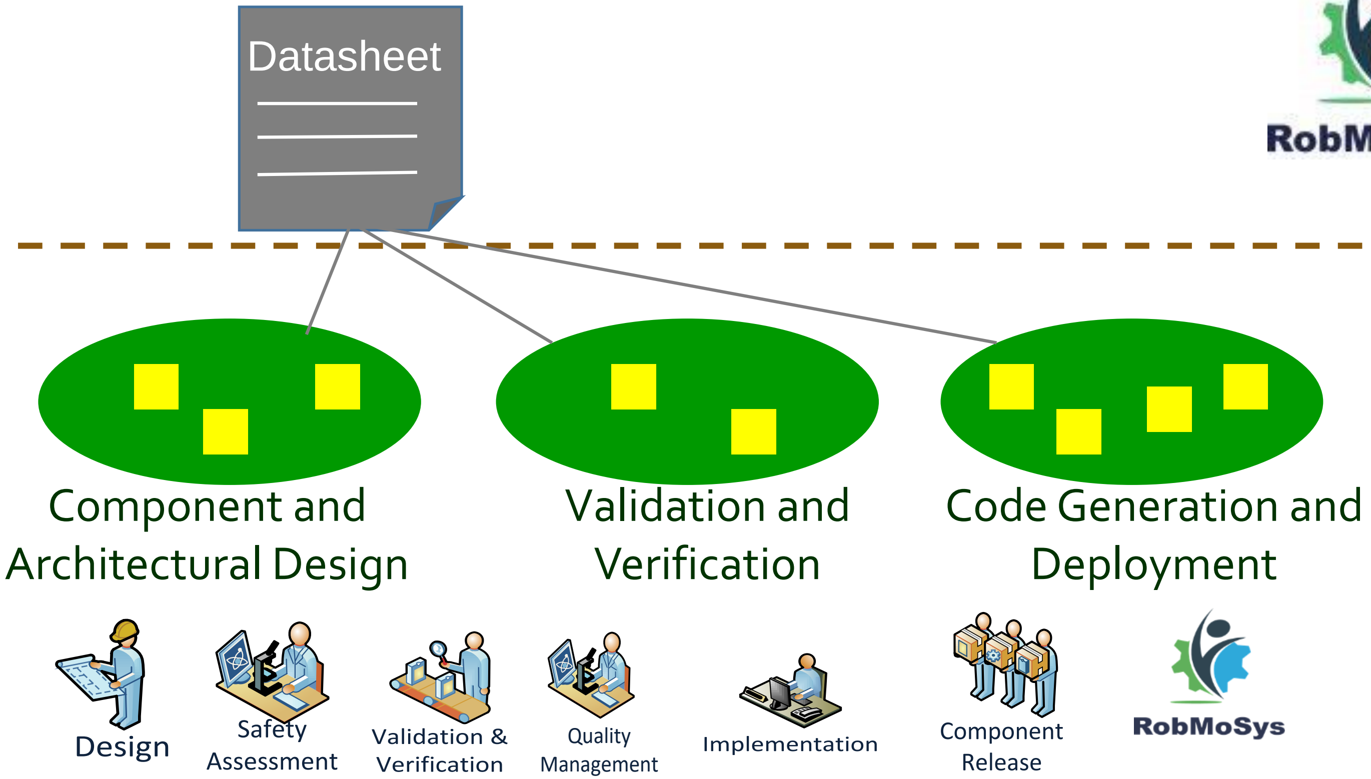
The current approach to developing robotics software is fundamentally code-centric and fragmented into numerous, often incompatible, frameworks. Consequently, software engineering of robotics applications is still in an early "craft" stage when compared to other domains such as automotive or avionics.

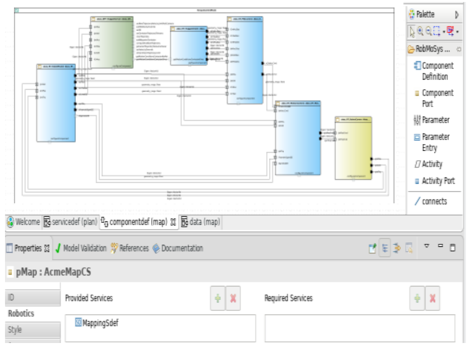
The Solution

RobMoSys provides a consolidated EU Digital Industrial Platform for Robotics, which (a) establishes a common methodology for software development, (b) improves tools and fosters interoperability through model interchange and, (c) facilitates composability. One of the primary assets of the RobMoSys ecosystem is the Papyrus4Robotics toolset. Papyrus4Robotics is an adaptation of the generic Papyrus tool, which provides a model-based graphical development environment specifically designed for RobMoSys.

The Benefits

Papyrus4Robotics provides the following core advantages: (a) facilitates reuse of robotics assets in different contexts (different robots, robot versions, upgrades, etc.) by means of dedicated configuration and instantiation tools; (b) eases the exploitation of safety standards used in certification processes; (c) enables robot designers to integrate modules developed in other environments into their own proprietary environments.


Datasheet
Component and Architectural Design
Validation and Verification
Code Generation and Deployment
Design
Safety Assessment
Validation & Verification
Quality Management
Implementation
Component Release
RobMoSys



pMap : AcmeMapCS

Provided Services: MappingSdef

Required Services:

Papyrus4Robotics

The Robot Operating System (ROS)

- https://index.ros.org/doc/ros2/

- Set of software libraries and tools for building robot applications.
- Wide range (drivers, algorithms, visualization tools …), open source.
- ROS 1 was started in 2007
- ROS 2 – reduced footprint, based on DDS middleware, better real-time support
- microROS for resource-constrained systems

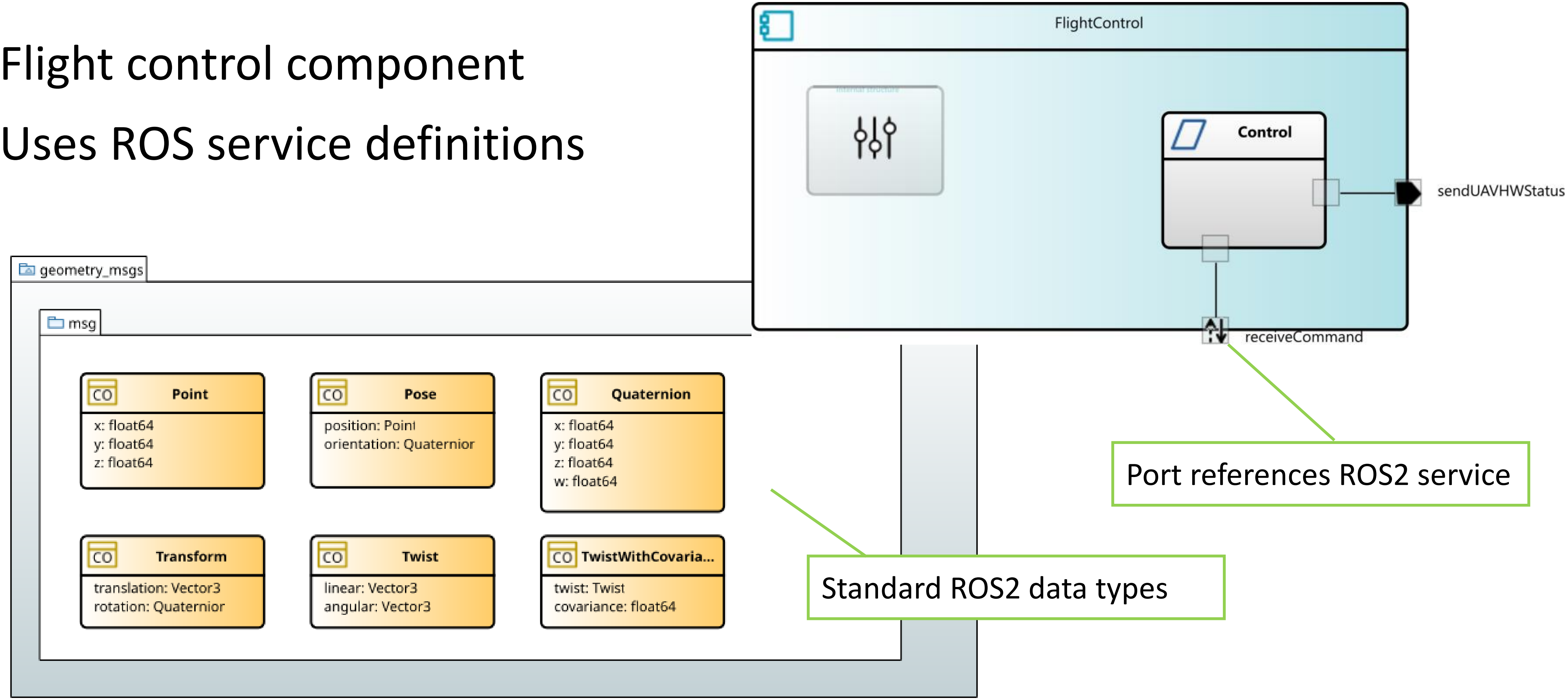
⇨ Growing number of companies migrating to ROS2

⇨ Papyrus for Robotics supports code generation for ROS2

ROS-MDD ROSin

Examples in Papyrus for Robotics

- Flight control component
- Uses ROS service definitions



FlightControl

Control

sendUAVHWStatus

receiveCommand

geometry_msgs

msg

Point
x: float64
y: float64
z: float64

Pose
position: Poin
orientation: Quaternior

Quaternion
x: float64
y: float64
z: float64
w: float64

Transform
translation: Vector3
rotation: Quaternior

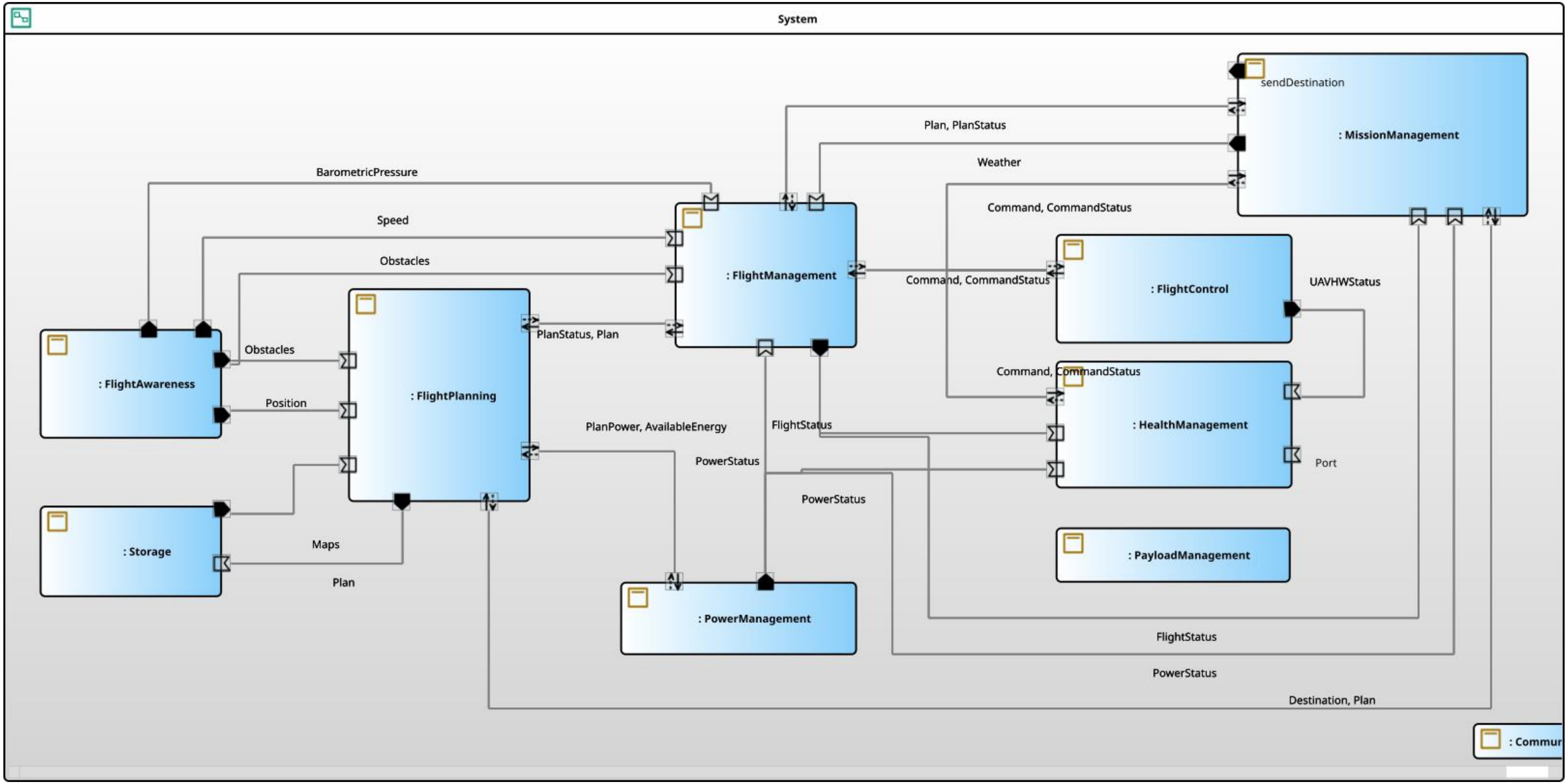
Twist
linear: Vector3
angular: Vector3

TwistWithCovaria...
twist: Twist
covariance: float64

Port references ROS2 service

Standard ROS2 data types

Example System model


System
sendDestination
: MissionManagement
Plan, PlanStatus
Weather
BarometricPressure
Speed
Command, CommandStatus
Obstacles
: FlightManagement
Command, CommandStatus
: FlightControl
UAVHWStatus
PlanStatus, Plan
Obstacles
: FlightAwareness
: FlightPlanning
Command, CommandStatus
Position
PlanPower, AvailableEnergy
FlightStatus
: HealthManagement
PowerStatus
Port
PowerStatus
: Storage
Maps
: PayloadManagement
Plan
: PowerManagement
FlightStatus
PowerStatus
Destination, Plan
: Commur