Supporting Separation of Roles in the SmartMDSD-Toolchain: ***Three*** Examples of Integrated DSLs

Christian Schlegel
Alex Lotz
Matthias Lutz
Dennis Stampfer

Computer Science Department
University of Applied Sciences Ulm, Germany

http://www.servicerobotik-ulm.de/

Service Robotics Ulm
autonomous mobile service robots

Hochschule Ulm
University of Applied Sciences

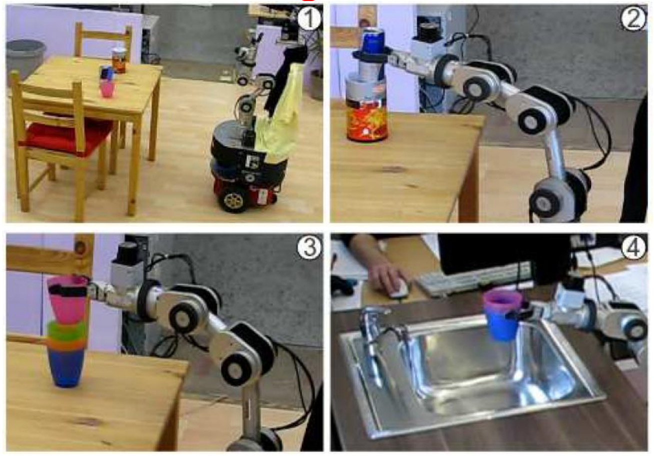
Variability Management in the Lifecycle

Design-Time Software Variability and Run-Time Reconfiguration

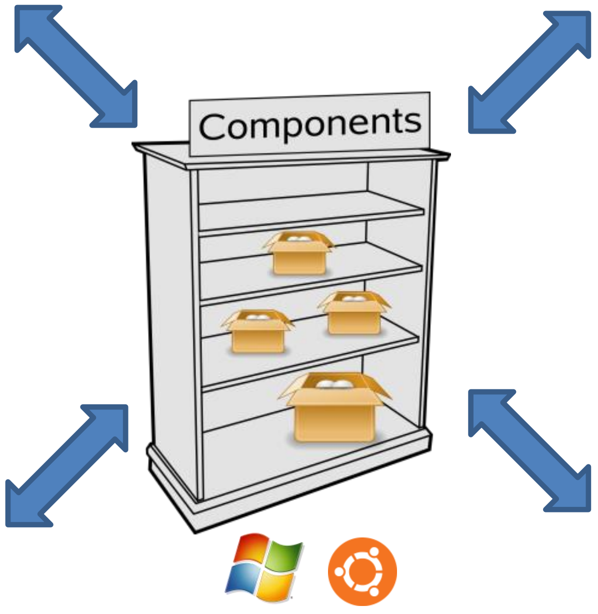
Butler Scenario

- Coffee Delivery
- Clean-up table
- Object Recognition
- States of objects

Runtime Reconfiguration

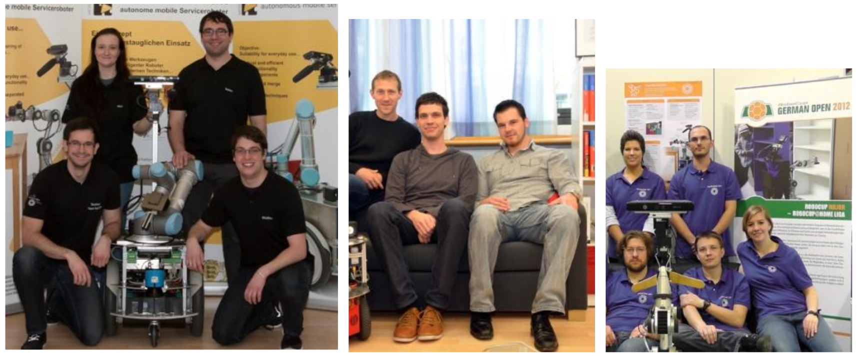


- Which coffee machine? Which velocity?
- Stacking cups and waste separation
- Active information-driven object recognition
- full or empty? Ready or problem?



Intralogistics Scenario

RoboCup@Home Student Team



Variability Management in the Lifecycle

Design-Time Software Variability and Run-Time Reconfiguration

SmartMDSD
(service oriented component model)
- Meta-Model
- Toolchain

SmartSoft
(implementation)
- CORBA / SmartSoft
- ACE / SmartSoft
- Linux, Windows, etc.

SmartTCL
(Task Coordination Language)

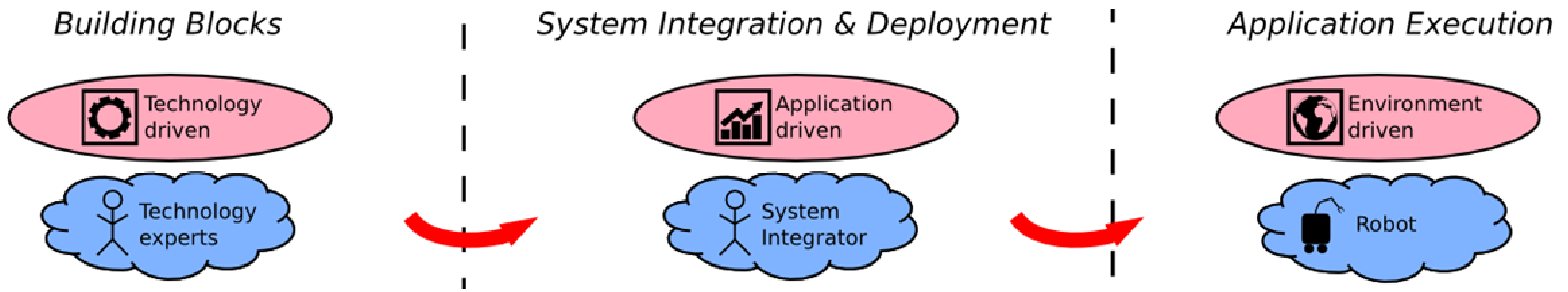
VML
(Variability Modeling Language)

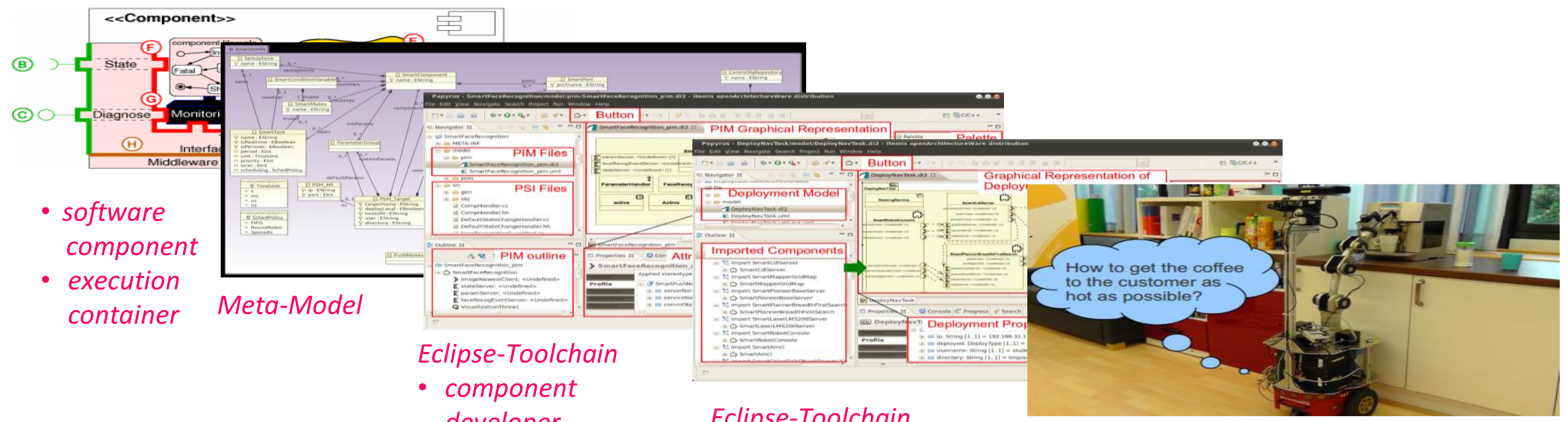
Domain Specific Languages

DSL *Communication Object*

DSL *Parameter*

DSL *Documentation of Components*





- software component
- execution container

Meta-Model

Eclipse-Toolchain
- component developer

Eclipse-Toolchain
- system integrator
- deployment

Real robot in real world

DSLs: Lessons Learned

Guide DSLs by a domain and its requirements: *achieve Separation of Roles and support Separation of Roles*

- *early agreement* (via modeling) on contracts between building blocks and responsibilities (service definitions)
- *modifications trigger* the need for *agreements*: ensure obligatory workflows (assigned roles / responsibilities / privileges)
- always *up-to-date documentation* (model [including documents] is documentation instead of document driven approach)

Approach: *MDSD* (model-driven software development) supported by *integrated DSLs* (domain specific languages)

Lessons Learned:

- *be user-focused*: simplicity, compactness, specific for a particular user need / user role
  - better have separated and specific DSLs instead of trying to merge everything into a single DSL
  - graphical modeling versus textual DSL: offer whatever is most appropriate for a role and task
- *support different views*:
  - assign user-role specific privileges (see example 2 / DSL “parameter”)
- *needs to be integrated into workflow and tools*:
  - do not come up with just another isolated DSL
  - DSL must fit seamlessly into an overall workflow (e.g. easy and seamless access to textual modeling from within graphical models => do not require manually opening a separated text document)
  - seamless access from different DSLs to information shared between models: no matter whether it is from within graphical or textual models

Selected examples in this talk (fully integrated within the SmartMDSD toolchain):

- DSL 1: immediate use of entities and delayed transformation (part by part as needed) into platform implementation
- DSL 2: textual model accessible from graphical model (stepwise refinement, different views, tool integration)
- DSL 3: graphical models are used by textual models (stepwise annotations)

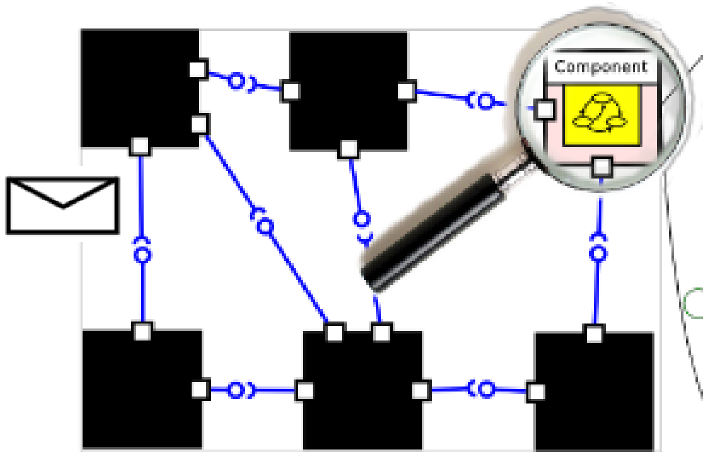
Example 1: DSL „Communication Object“

immediate use of entities and delayed transformation (part by part as needed) into platform implementation

entities are modeled textually (Xtext integrated DSL) and are imported into and referenced from the graphical model (UML)

Purpose:
- achieve composability of services in order to support reuse of software components

Requirements:
- describe (model) entities (data structures) once and consistently reuse those entities as often as possible
- you must be able to work with these entities although e.g. the target platform and target middleware is not yet decided



```
CommObject CommBasePose {
    covMatrix: Double[9] = 0.0
    updateCount: UInt32
    pose3D: CommObjectRef(CommPose3d)
    timeStamp: CommObjectRef(CommTimeStamp)
}
CommObject CommPose3d {
    position: CommObjectRef(CommPosition3d)
    orientation: CommObjectRef(CommOrientation3d)
}
CommObject CommPosition3d {
    x: Double = 0.0
    y: Double = 0.0
    z: Double = 0.0
}
CommObject CommOrientation3d {
    azimuth: Double = .0
    elevation: Double = .0
    roll: Double = .0
}
```

The following data types are available for attributes of the communication object:

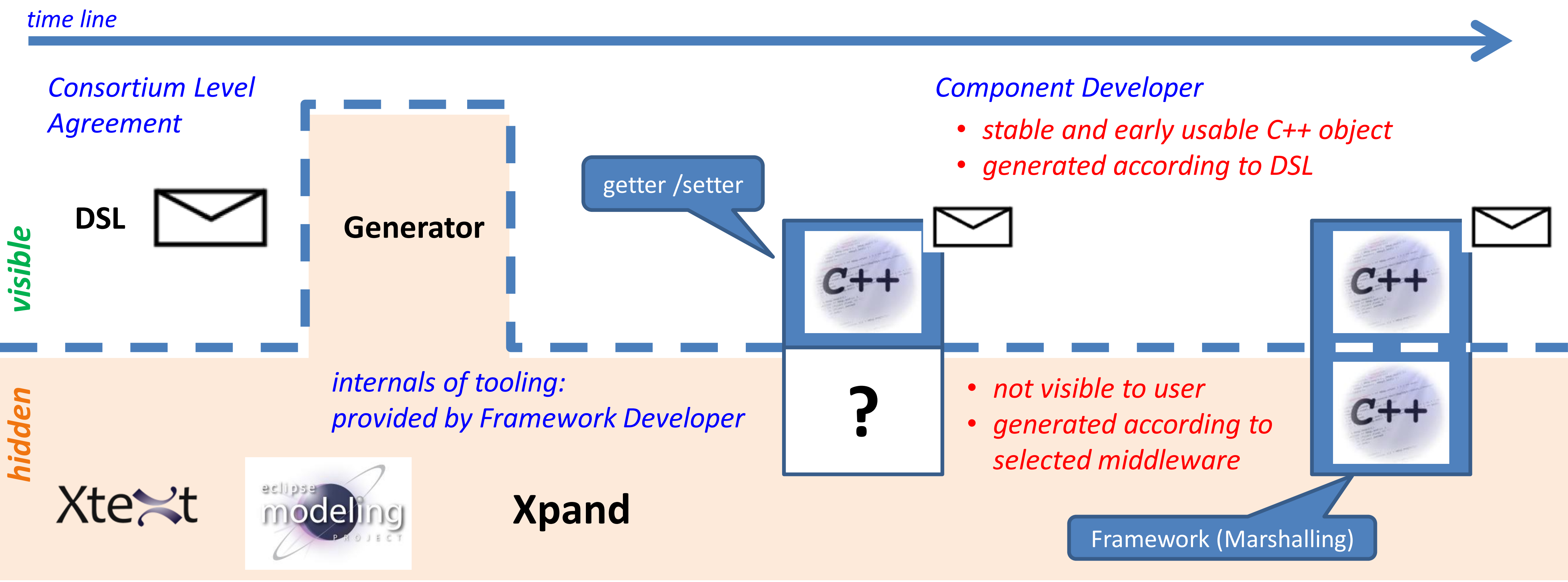
- Boolean
- Double, Float
- Int8, Int16, Int32, Int64
- UInt8, UInt16, UInt32, UInt64
- String
- [N] – Array of any of the previous types. N can be an integer denoting the number of elements or * for a flexible list
- CommObjectRef(NAME) – to indicate a nested communication object
- StructRef(NAME) – to indicate a nested Struct
- EnumRef(NAME) – to indicate an enumeration usage

Example 1: DSL „Communication Object“

immediate use of entities and
delayed transformation (part by part as needed) into platform implementation

Approach: describe “data structures” independently from their implementation, i.e.

- models must be implementable with different kinds of middleware
- the part relevant to a component developer must be transformed early into the used programming language
- late binding of middleware to execution must be possible seamlessly



Example 2: DSL „Parameter“

textual model accessible from graphical model (stepwise refinement, different views, tool integration)

Design-Time
consortium level agreement

stepwise refinement binds more and more variability

DSL

Xtext

```
ParamSet LaserObstacleAvoidParameter {

    Param speed {
        v_x : Double
        v_omega : Double
    }

}
```

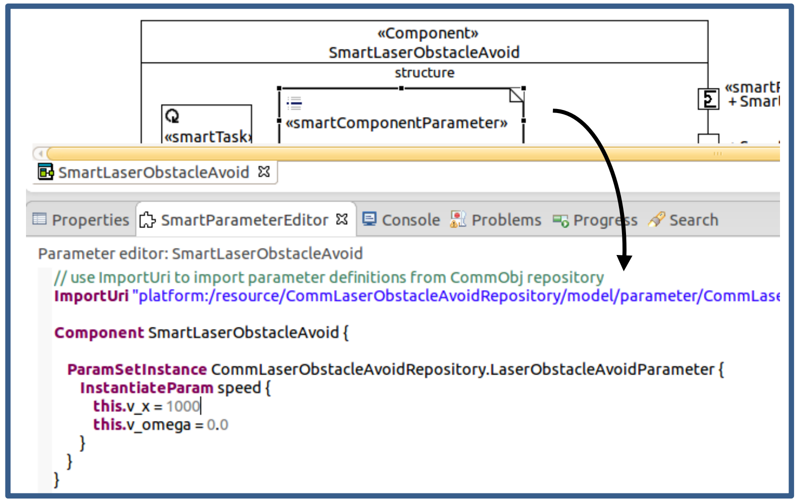
Datatypes:
Boolean, Double, Float, Int8, Int16, Int32, Int64, UInt8, UInt16, UInt32, UInt64, String, Arrays, Enumerations

Design-Time
component developer

DSL

Xtext

<values>: can be referenced and enriched,
<name>, <type>: cannot be modified

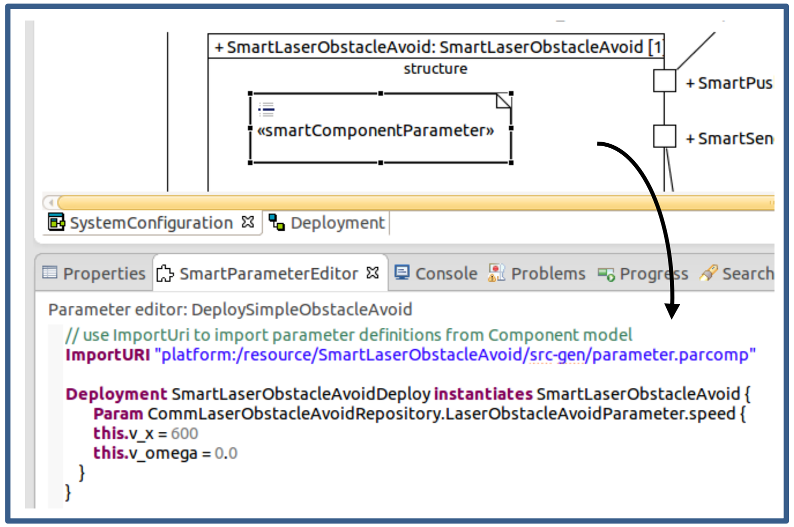


```
Parameter editor: SmartLaserObstacleAvoid
  // use ImportUri to import parameter definitions from CommObj repository
  ImportUri "platform:/resource/CommLaserObstacleAvoidRepository/model/parameter/CommLase

  Component SmartLaserObstacleAvoid {

    ParamSetInstance CommLaserObstacleAvoidRepository.LaserObstacleAvoidParameter {
      InstantiateParam speed {
        this.v_x = 1000
        this.v_omega = 0.0
      }
    }
  }
```

Design-Time
System Integrator

DSL

Xtext



```
Parameter editor: DeploySimpleObstacleAvoid
  // use ImportUri to import parameter definitions from Component model
  ImportURI "platform:/resource/SmartLaserObstacleAvoid/src-gen/parameter.parcomp"

  Deployment SmartLaserObstacleAvoidDeploy instantiates SmartLaserObstacleAvoid {
      Param CommLaserObstacleAvoidRepository.LaserObstacleAvoidParameter.speed {
      this.v_x = 600
      this.v_omega = 0.0
    }
  }
```

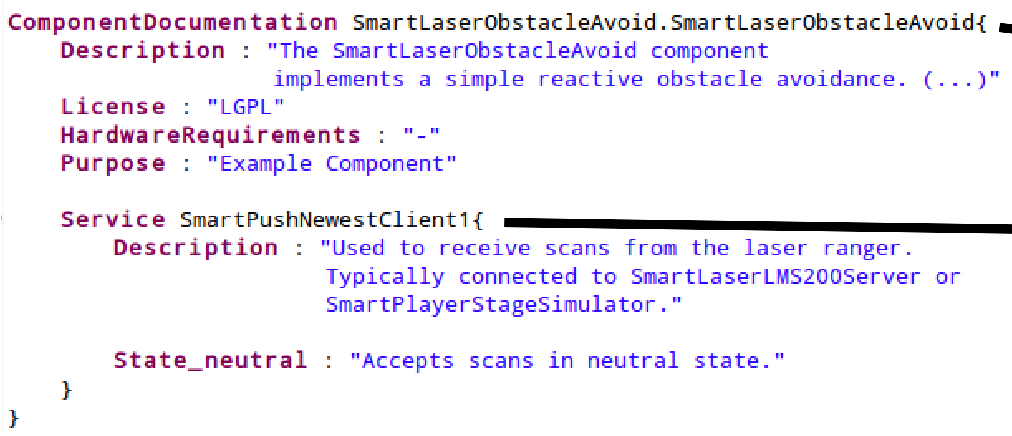
Run-Time
Robot

Run-time access via Parameter Pattern (Comm. Pattern / Component Model)

Example 3: DSL „Documentation“

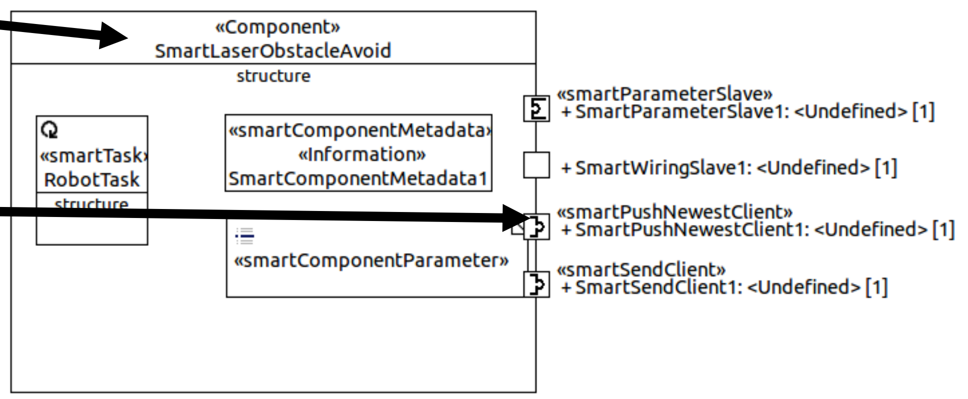
graphical models are used by textual models (stepwise annotations)

Xtext DSL



```
ComponentDocumentation SmartLaserObstacleAvoid.SmartLaserObstacleAvoid{
    Description : "The SmartLaserObstacleAvoid component
                   implements a simple reactive obstacle avoidance. (...)"
    License : "LGPL"
    HardwareRequirements : "-"
    Purpose : "Example Component"

    Service SmartPushNewestClient1{
        Description : "Used to receive scans from the laser ranger.
                       Typically connected to SmartLaserLMS200Server or
                       SmartPlayerStageSimulator."

        State_neutral : "Accepts scans in neutral state."
    }
}
```

Papyrus / UML-Profile



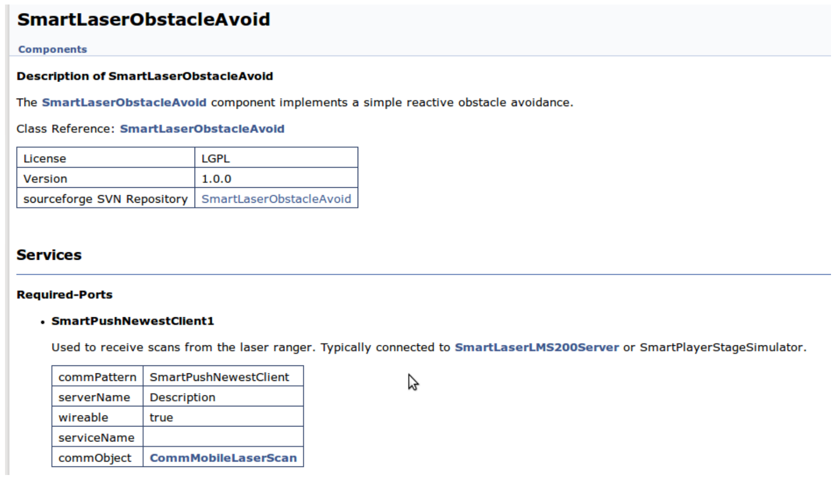
Current State:

- from stepwise annotations in models to a complete document
- add human-centered prosa / docu / explanations

Future Work:

human readable model annotations will be presented at the appropriate views within the toolchain (do not read a separate WIKI)

Output: Complete Document



SmartLaserObstacleAvoid

Components

Description of SmartLaserObstacleAvoid

The **SmartLaserObstacleAvoid** component implements a simple reactive obstacle avoidance.

Class Reference: **SmartLaserObstacleAvoid**

License	LGPL
Version	1.0.0
sourceforge SVN Repository	SmartLaserObstacleAvoid

Services

Required-Ports

- **SmartPushNewestClient1**

  Used to receive scans from the laser ranger. Typically connected to **SmartLaserLMS200Server** or SmartPlayerStageSimulator.

commPattern	SmartPushNewestClient
serverName	Description
wireable	true
serviceName	
commObject	**CommMobileLaserScan**

Example 3: DSL „Documentation“

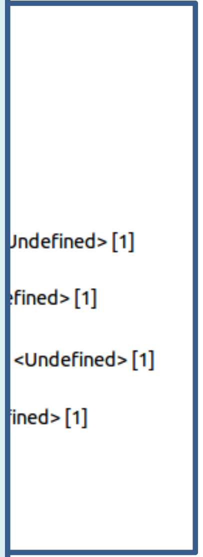
graphical models are used by textual models (stepwise annotations)

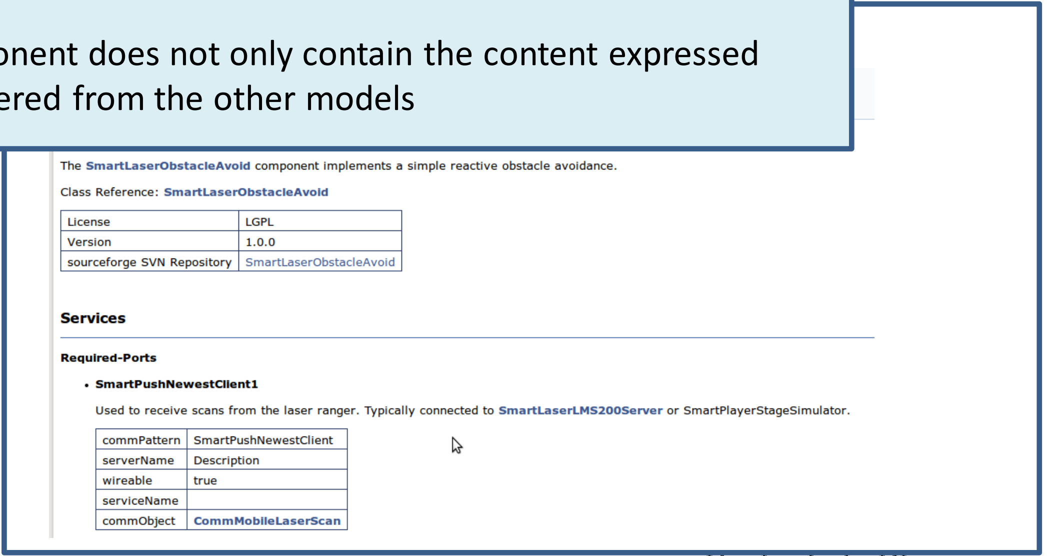
Textual models reference graphical models:

Workflow:
- component developer models component hull as graphical model
- someone else (typically another role, e.g. technical writer) opens document editor and provides the documentation for the “outside view” of this component
  - he is being assisted in this job by the DSL as the auto completion mechanism suggests those port, states etc. that still need to be documented
  - behind the scenes, the editor refers to the graphical model in order to come up with its suggestions
  - behind the scene, the very same mechanism of referencing the graphical component model prevents from modifying the component hull from within the role of the technical writer
- the generator composes out of the text elements of the documentation, the models (graphical model of the component, used communication objects, used parameters etc.) the document (currently, the final document is html with Doxygen as intermediate representation)
  - the html documentation describing the black box view of a component does not only contain the content expressed via the documentation DSL, but also all relevant information gathered from the other models

complete document
- add human-centered prosa / docu / explanations

Future Work:
human readable model annotations will be presented at the appropriate views within the toolchain
(do not read a separate WIKI)





The SmartLaserObstacleAvoid component implements a simple reactive obstacle avoidance.

Class Reference: SmartLaserObstacleAvoid

License	LGPL
Version	1.0.0
sourceforge SVN Repository	SmartLaserObstacleAvoid

Services

Required-Ports

- **SmartPushNewestClient1**

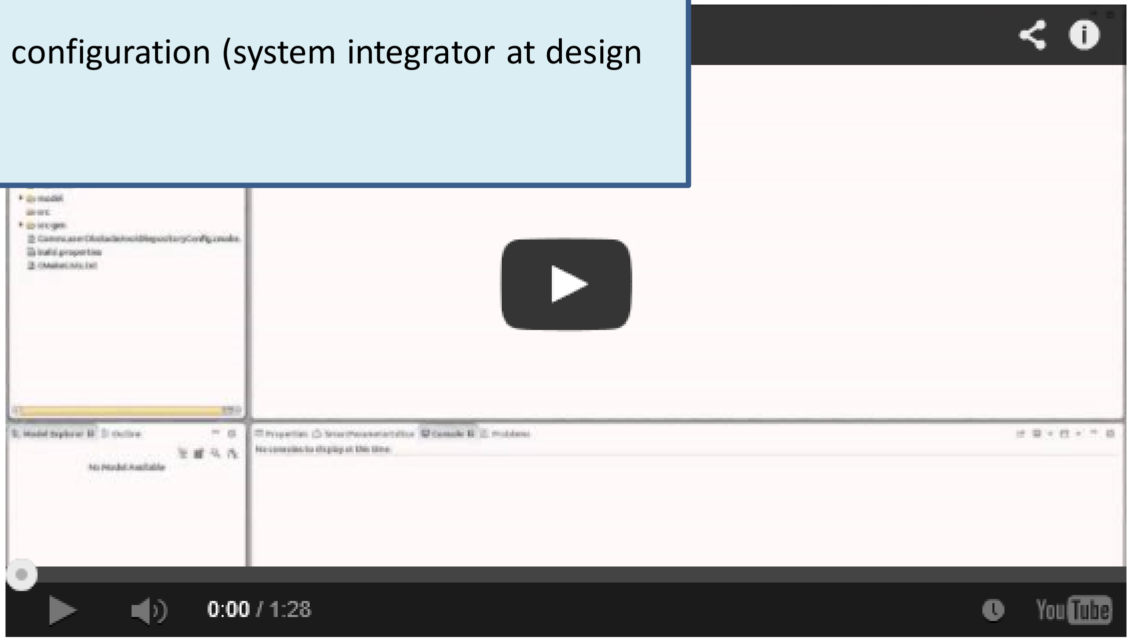
  Used to receive scans from the laser ranger. Typically connected to SmartLaserLMS200Server or SmartPlayerStageSimulator.

commPattern	SmartPushNewestClient
serverName	Description
wireable	true
serviceName	
commObject	CommMobileLaserScan

SmartMDSD Toolchain: Parameter Definition

Purpose: (Example DSL 2)

- see how a *parameter* for later component configuration (system integrator at design time, robot at run-time) is being defined
- please go to 00:34



http://youtu.be/2U4KxSgwtqY

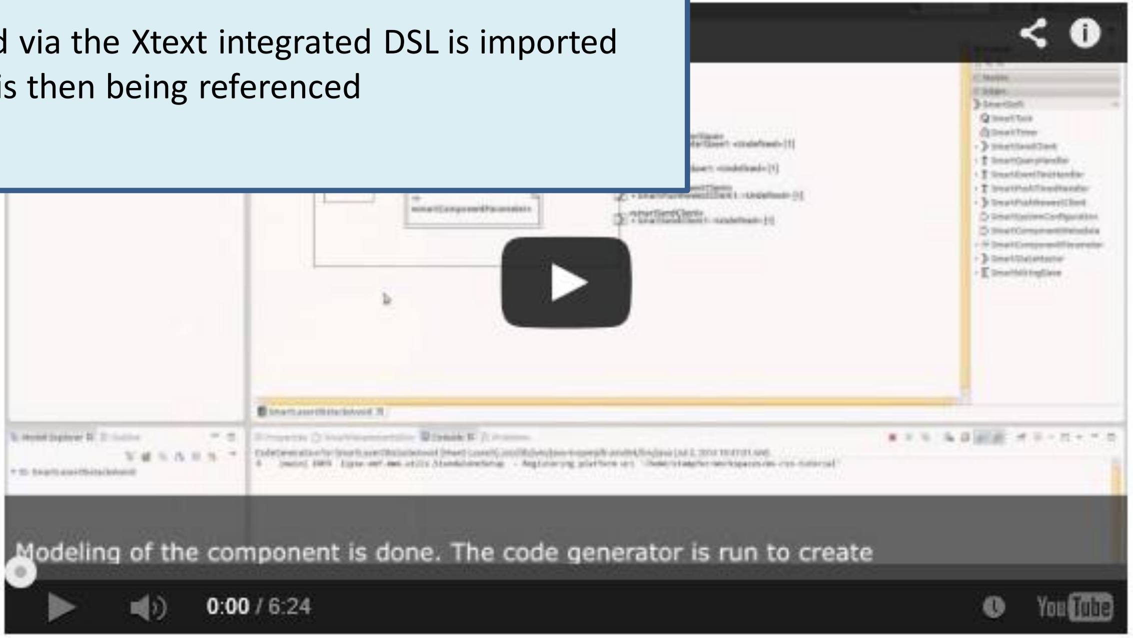
This video demonstrates the modeling of a parameter using the SmartMDSD Toolchain.

The parameter represents a configurable maximum velocity of a robot. This parameter can later be instantiated by components. The maximum speed can then be configured through the parameter service.

SmartMDSD Toolchain: Component Development

Purpose: (Example DSL 1)

- see how a communication object modeled via the Xtext integrated DSL is imported into the graphical UML model and how it is then being referenced
- please go to 01:50


Modeling of the component is done. The code generator is run to create
0:00 / 6:24

http://youtu.be/chyRCu4FCbs

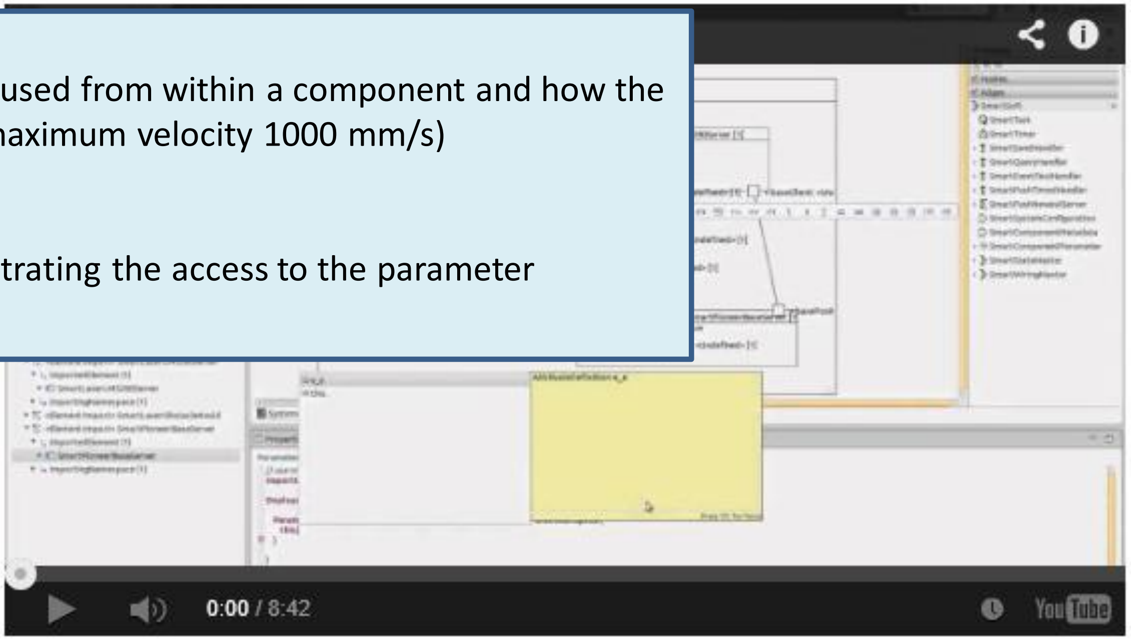
This video demonstrates the modeling and implementation of a component using the SmartMDSD toolchain. ***It thereby illustrates how to use the modeled parameter from within a component.***
The component receives laser scans. A simple obstacle avoidance algorithm outputs values for speed and direction. The component then limits the maximum speed according to a variation point (parameter "v_x", modeled in the previous video) before providing the navigation commands through one of its services. This parameter "v_x" can be configured during runtime of the component through its parameter service.

SmartMDSD Toolchain: System Configuration and Deployment

Purpose: (Example DSL 2)

- see how a predefined parameter is being used from within a component and how the parameter can be given an initial value (maximum velocity 1000 mm/s)
- please go to 03.05

- later on, you can see the source code illustrating the access to the parameter
- please go to 05:18 – 05:45



http://youtu.be/y-S33qaeNfI

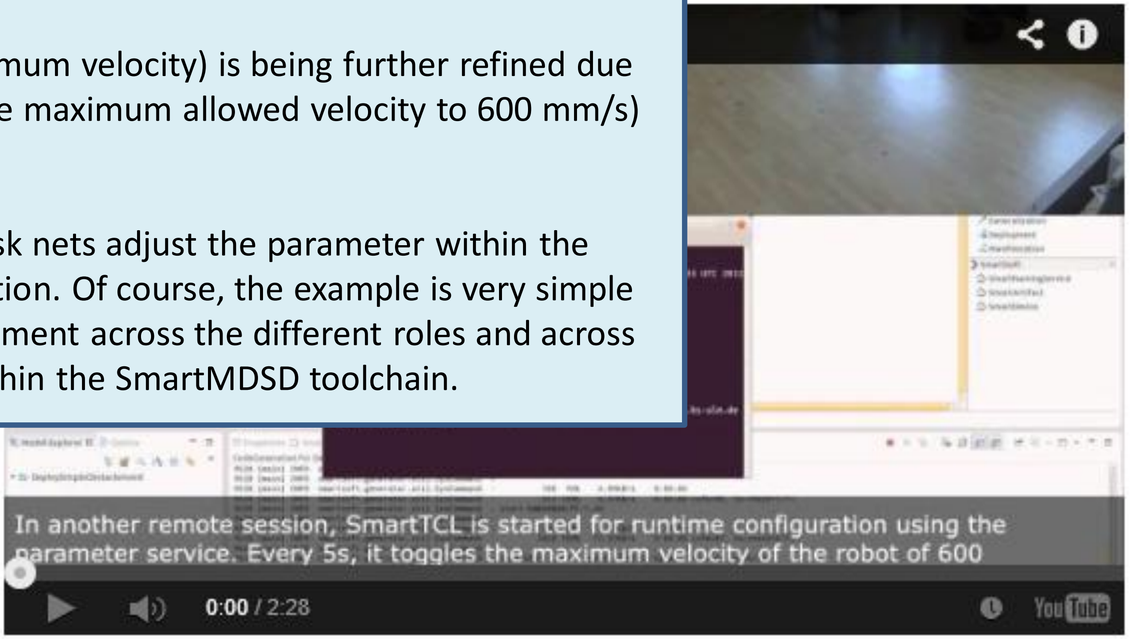
This video demonstrates the creation of system configuration and deployment model using the SmartMDSD toolchain.
The scenario: a robot shall drive and avoid obstacles. It reuses the (existing) components SmartLaserObstacleAvoid (see previous screencast), SmartLaserLMS200Server (laser ranger) and SmartPioneerBaseServer (robot). The system configuration model models the connection and configuration of components. The deployment model models the distribution of components on hardware.
According to system level needs, we restrict the maximum allowed velocity from 1000 mm/s (as is maximum capability of the component) to 600 mm/s (as is considered maximum for this application).

SmartMDSD Toolchain: Deployment and Run-Time Variability

Purpose: (Example DSL 2)

- see how the component level value (maximum velocity) is being further refined due to system level requirements (reducing the maximum allowed velocity to 600 mm/s)
- please go to 04:35

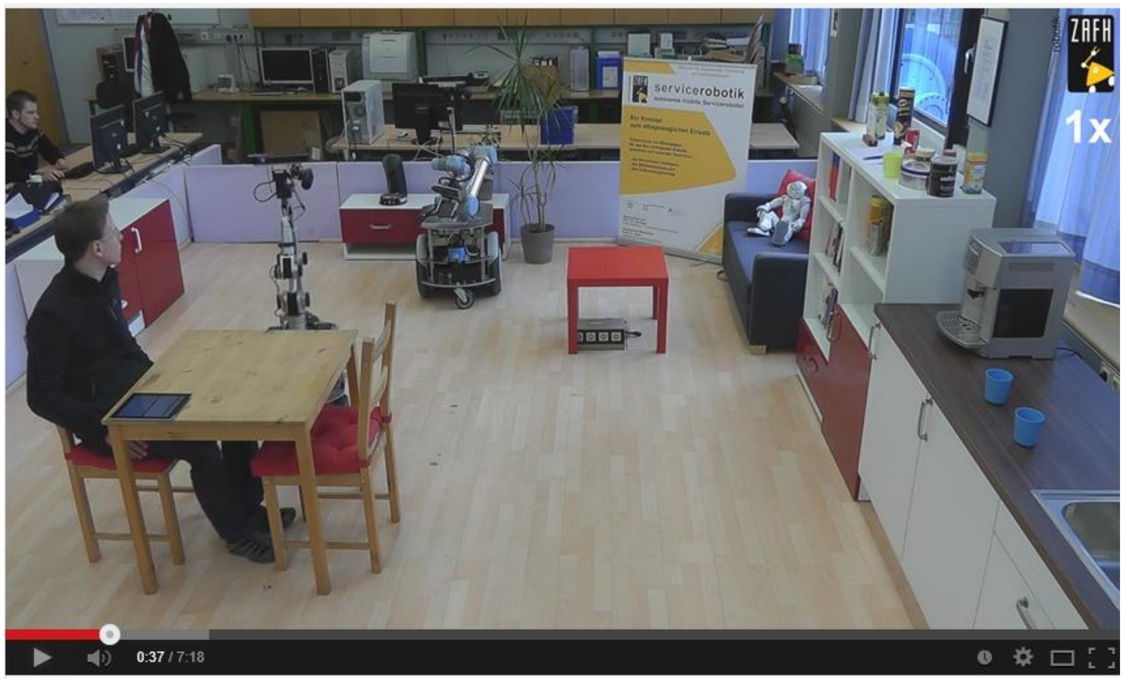
- later on, you can see how the run-time task nets adjust the parameter within the given limits according to the current situation. Of course, the example is very simple in order to illustrate the seamless management across the different roles and across design-time / run-time of a parameter within the SmartMDSD toolchain.


In another remote session, SmartTCL is started for runtime configuration using the parameter service. Every 5s, it toggles the maximum velocity of the robot of 600

http://youtu.be/OZcC4ipt_BM

This video demonstrates the deployment and execution of an application developed using the SmartMDSD toolchain.
The application (laser obstacle avoidance from a previous video) is deployed using SSH. A remote session on the robot is established in order to run it.
We show how we access the parameter during run-time. The robot will first drive with a maximum velocity of 600m/s (as has been configured as system configuration). Later, SmartTCL is used to change the maximum velocity of the component to 200 and back to 600 every 5s via the parameter service and the explicated variation point v_x.

Video: Real-World example



http://youtu.be/DjjNUPpj36E

The scenario shows the service robots "Kate" and "Larry" acting as butler. Kate takes orders from persons and hands over parts to Larry. While Kate makes a cup of coffee, Larry fetches the sugar dispenser from within a closed sideboard.

Links

- Portal
  - http://www.servicerobotik-ulm.de/
- Paper and Talks
  - http://www.servicerobotik-ulm.de/drupal/?q=node/15
- Videos
  - http://youtube.com/user/roboticsathsulm
- Software
  - http://www.servicerobotik-ulm.de/drupal/?q=node/7