Robotics MDE Workshop

Leuven, 11th February 2013

Separation of Roles: Challenges for MDSD

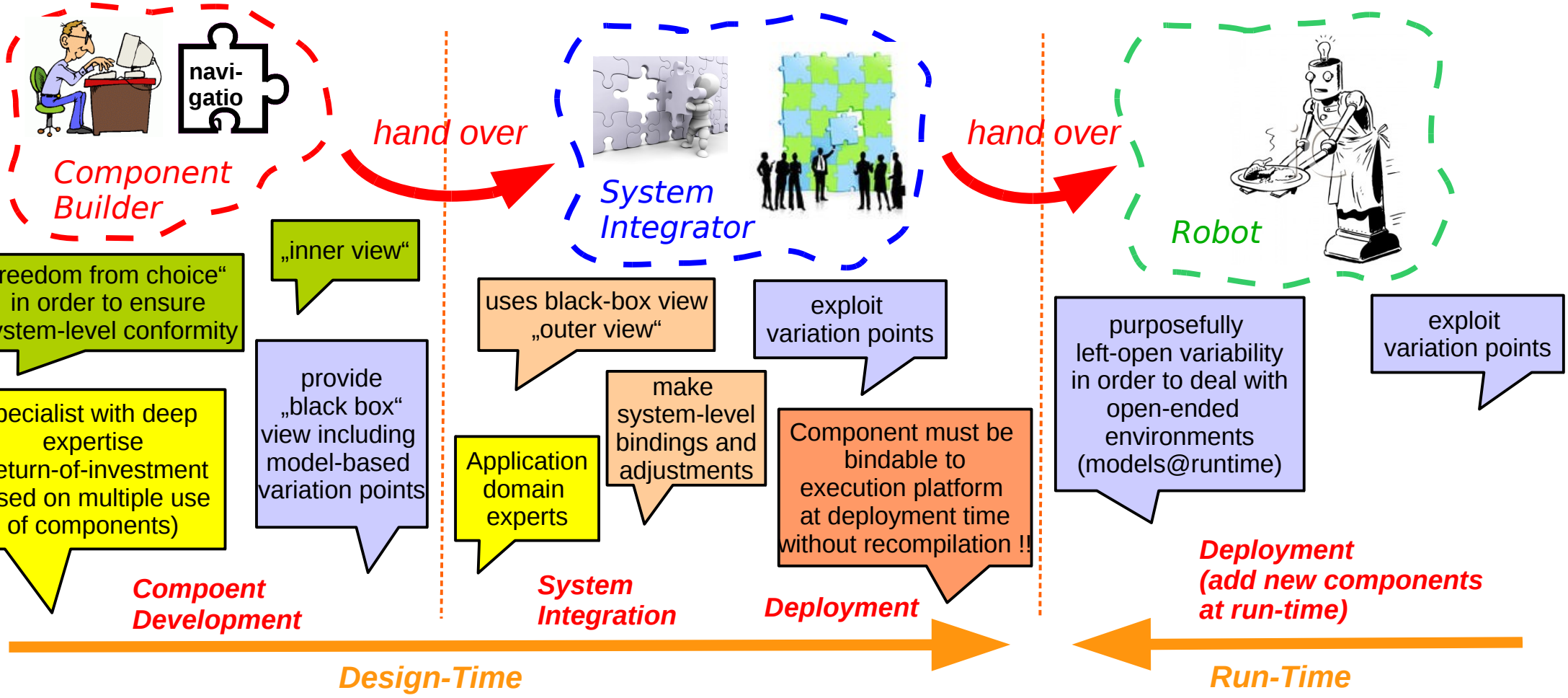
SmartSoft MDSD and its Transformations: Meta-Model, PIM, PSM, PSI

M.Sc. Alex Lotz

Prof. Dr. Christian Schlegel

Separation of Roles: Challenges for MDSD

- *Use models for the entire life-cyle of the robot*
- *Models are refined step-by-step until finally they become executable*
- *Separate inside view (component builder) from outside view (system integrator)*
- *Separate stable execution container from implementational technologies (middleware, OS)*
- *Variation points: design-time (component builder, system integrator), runtime (robot)*
  - *Explicitly model variability for late binding (by system integrator and even by the robot at runtime)*



stepwise refinement: (1) add more and more information, (2) bind more and more variability

Separation of Roles: Challenges for MDSD

Goal: Robotics Business Ecosystem
=> requires separation of roles
=> how to achieve this?

MDSD as solution technology
=> needs and priorities guided by separation of roles

MDSD for robotics / Challenges:
- Stepwise refinement instead of strictly linear MDA approach
  - Support deployment with late-binding of OS & middleware
- Variability modeling: design-time & run-time exploitation of variability
  - for QoS (quaility-of-service) in open-ended environments
  - to address non-functional properties
- provide role-specific support:
  - component developer, system integrator, robot, application domain expert, ...
- enable hand-over between these roles
  - black-box view, explicate and transfer constraints
- provide appropriate infrastructure
  - repositories for components & models with distributed access and versioning
  - shared and agreed meta-models and competition at implementational level
  - support also closed-source components in deployment to protect IP

MDE-support for the full life-cycle of a robotics system
- SW-component model, system model, hardware model, behavior model, etc. etc.
- Role models, Workflows and transformations

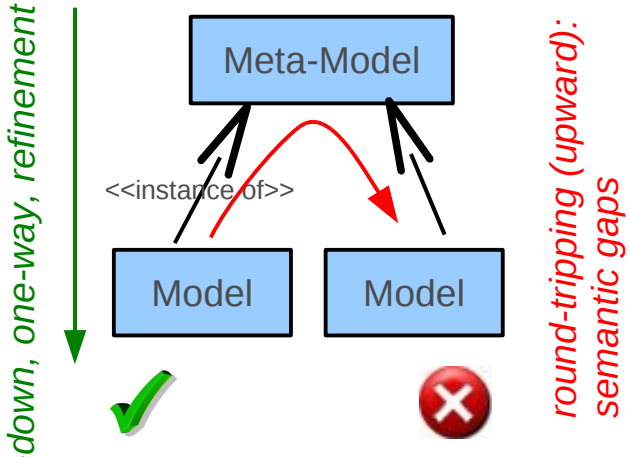
Separation of Roles: Challenges for MDSD

- Edward A. Lee
  - „Modeling languages that are not executable, or where the execution semantics is vague or undefined are not much better than TUML (The Truly Unified Modeling Language)“.
  - „We have to stop thinking of constraints as a universal negative!!!“
  - **„Freedom from Choice“ instead of „Freedom of Choice“**

- Robotics systems
  - **OMG MDA approach PIM => PSM => PSI is too linear**
  - In robotics systems,
    - parts of the hardware model (PSM, PSI) already needed at PIM level:
      (e.g., sensors and their mountings strongly influence algorithmic options)
      explicate / hand-over constraints from PIM to PSM to PSI
  - Separation of Roles
    - Component Developer: provide component to component shelf (not necessarily bound to middleware etc.)
    - System Integrator: picks-up component and binds it to target platform
    - Both: need to understand provided / required services

- **SmartMDSD covers a service-oriented component and system model with a focus on separation of roles and separation of concerns**
  - component, port semantics, lifecycle, system composition, deployment, runtime, variability modeling, etc.
  - component execution container
    - decouples inside view / outside view / OS / middleware, …
    - explicates variability and constraints for the various roles

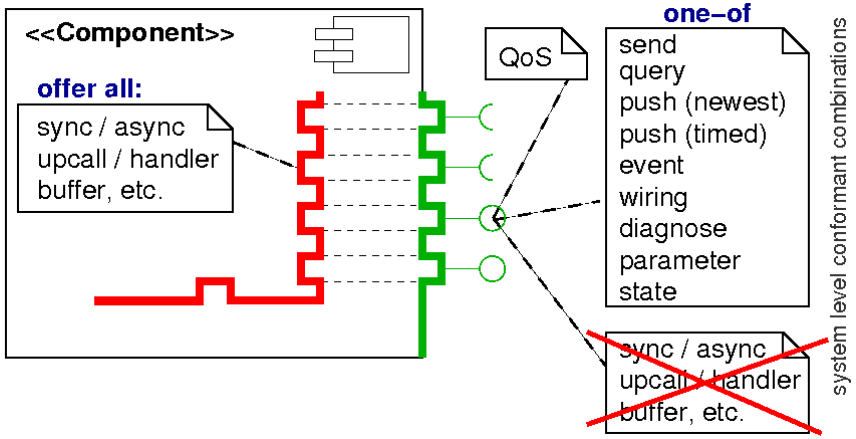
The SmartMDSD Toolchain: Assumptions...



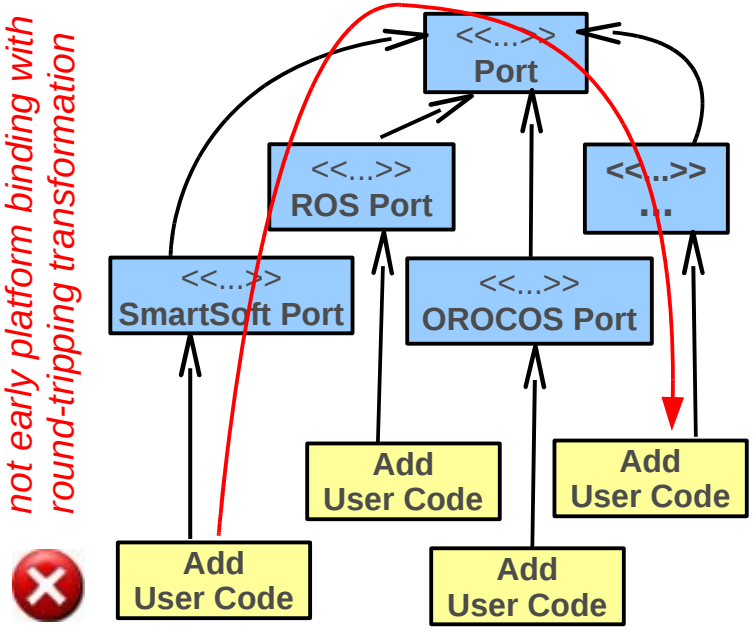
Transformation is enrichment (semantic gap), not just translation (syntactic level)

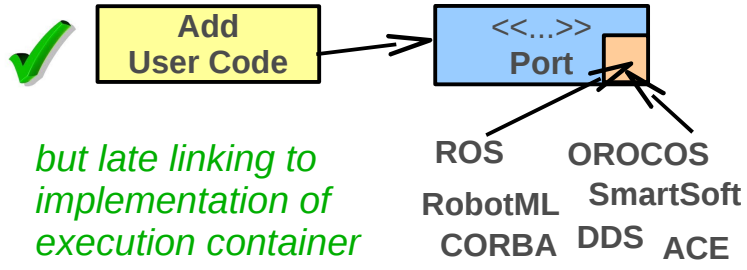
not variety outside where it affects system integration:
- *avoid complexity of combinatorial explosion of policies, mechanisms etc.*
- *ensure system level conformance (avoid distributed systems deadlocks etc.)*
- *avoid incompatible port variants of the same service*

but variety inside to ease job of developer:
- *give freedom to use desired access methods (sync, async, upcall, etc.)*
- *give freedom to install desired processing (passive, thread-pool, pipeline, buffers, etc.)*



not early platform binding with round-tripping transformation





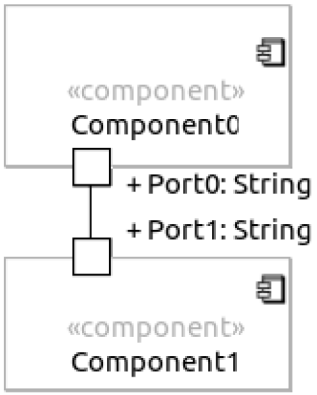
but late linking to implementation of execution container

explicate constraints (required sensor position, OS constraints etc.) introduced with user code

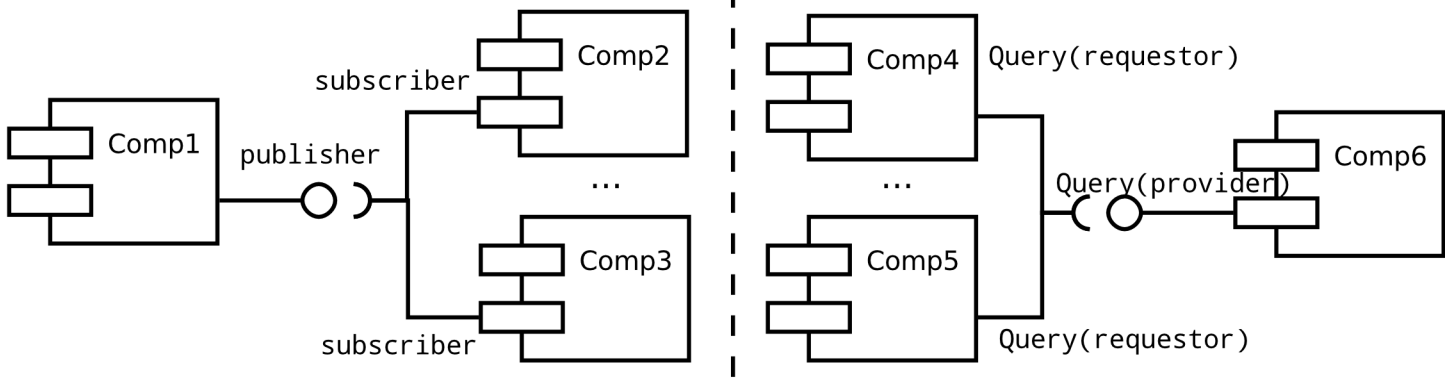
*The major focus is **not:** are the internals of the middleware are developed based on MDSD? but is on a workflow that supports separation of roles / concerns by means of MDSD*

The SmartMDSD Toolchain: Assumptions...

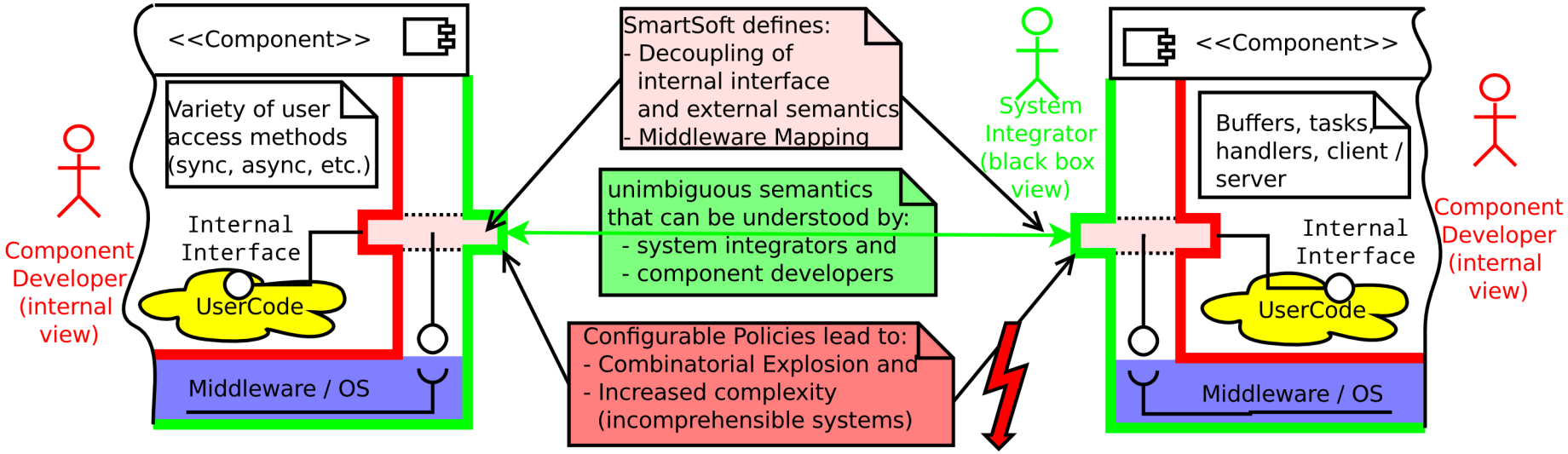
UML:



Robotics:



But: Communication Semantics? Protocols? Policies? QoS?
Interface (inside of component)? Internal Buffers/Handlers?

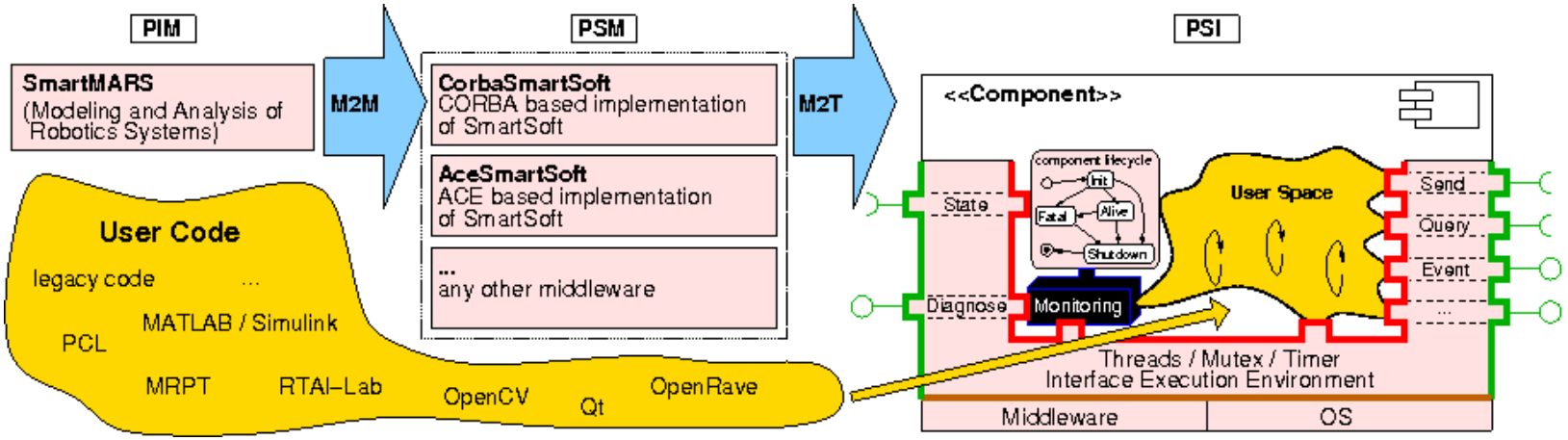


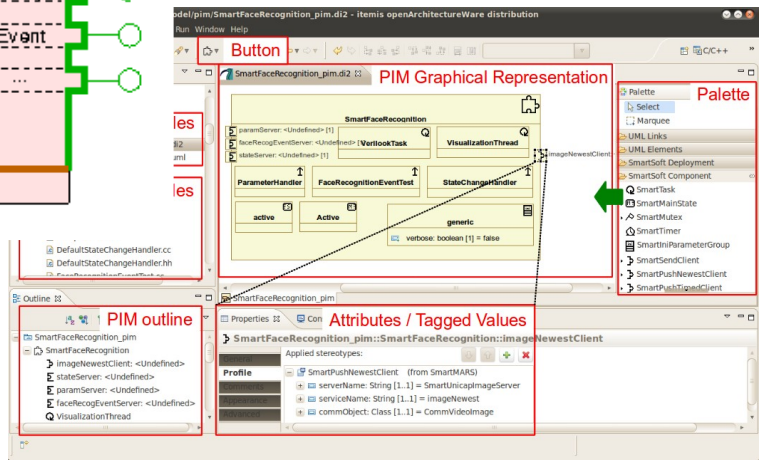
We decide on a small set of commnuication patterns, each binding a reasonable and consistent configuration and give them names!

→

Now system integrators understand the system and component developers know how to use the services!

The SmartMDSD Toolchain: Standard Sequential Workflow According to MDA

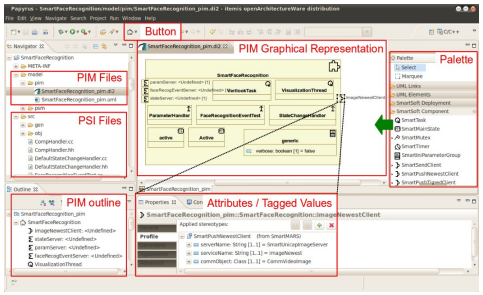




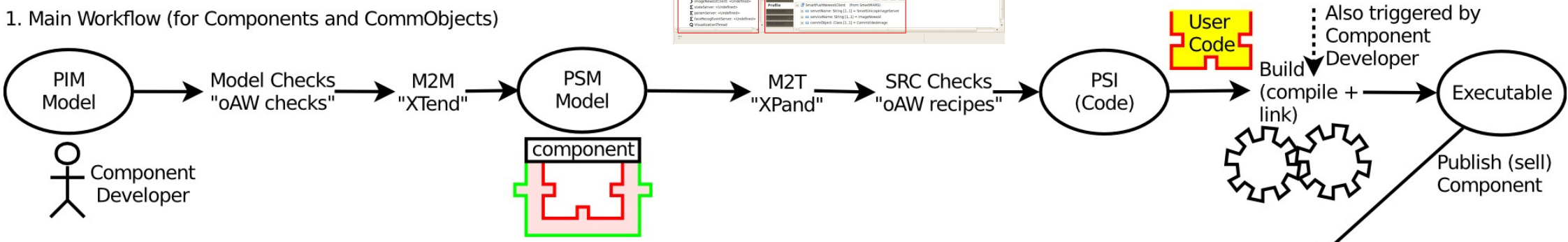
- ***OMG MDA is not perfectly fitting*** *(too linear) since we need stepwise refinements (more and more bindings of variability across the different and separated roles)*
- e.g. component builder
  - uses SmartMARS to specify component hull (stable internal structure and interface: ***red***)
  - generates code of component hull (***red*** **&** ***green*** interfaces provided as templates and via generation gap pattern) and adds user code (***yellow***)
- e.g. system integrator
  - exploits left open variability for adjusting settings of components
  - specifiies target platform during deployment step
  - deployment step adds implementation of execution container & links platform specific libraries (links together red, green, brown parts etc. as marked by ***pink coloring***)

=> we need **late binding of execution container** (middleware / OS) and variation points during deployment
=> we need to support selling **closed components** as object files (intellectual property) with late deployment
=> we (perhaps also) need to be able to compile several components into a single process (as with embedded systems)

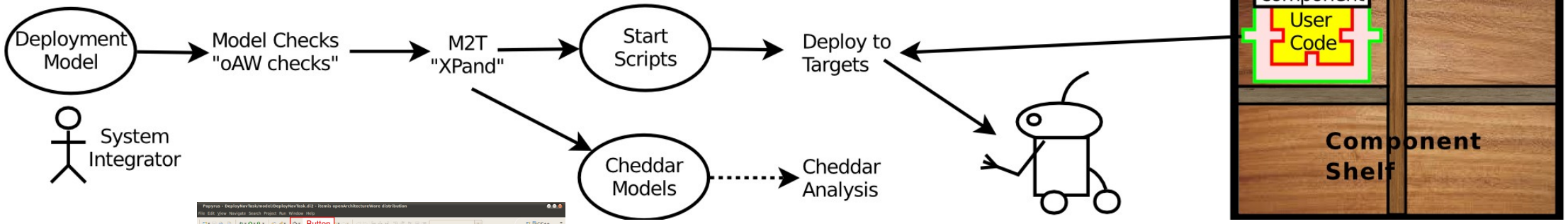
SmartMDSD Toolchain:
currently implemented traditional MDA workflow / automized steps

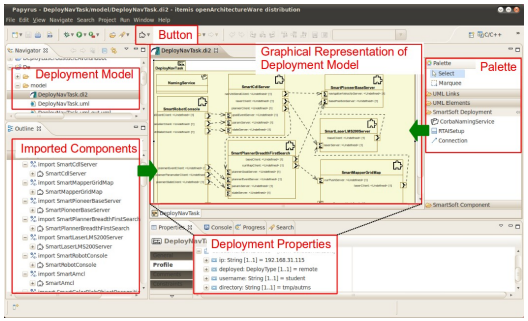


1. Main Workflow (for Components and CommObjects)



2. Deployment Workflow



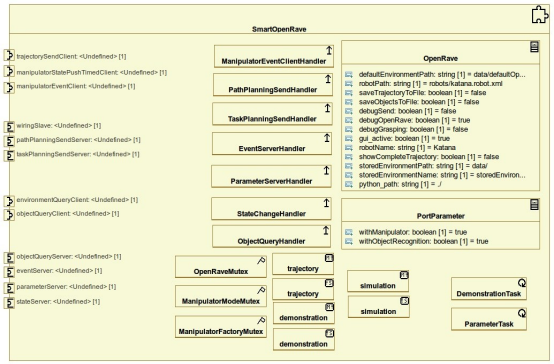


The SmartMDSD Toolchain:
Stepwise-Refinement Workflow / current work in progress

component developer

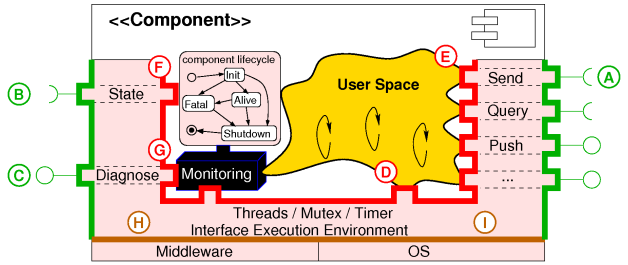
SmartMARS
component builder view
- build component model and generate component hull with explicated constraints (bindings, variability)
- add user code
- provide black-box view

Component Level: PIM => PSM => PSI



+

- **User: add user code**
- Toolchain:
  - generation gap pattern
  - templates



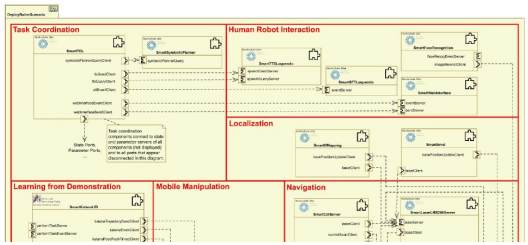
hand over

component model and object / binary
- models of valid bindings for variability
- constraints with respect to system integration (e.g. x86, RTAI, etc.)

system integrator

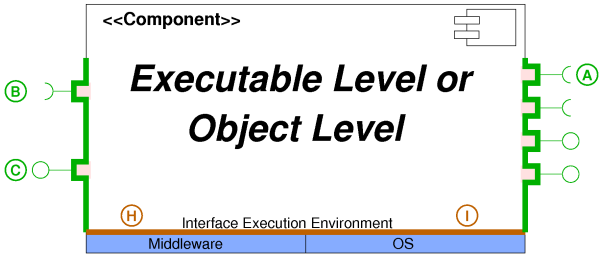
SmartMARS
system integrator view
- design deployment / system model
- bind variability for a balanced system integration
- use black-box component models
- use models of target platforms
- bind against middleware

System Level: PIM => PSM => PSI



+

- Toolchain:
  link platform-specific libraries for execution container



hand over

Run-time model and executable
- models of run-time variability

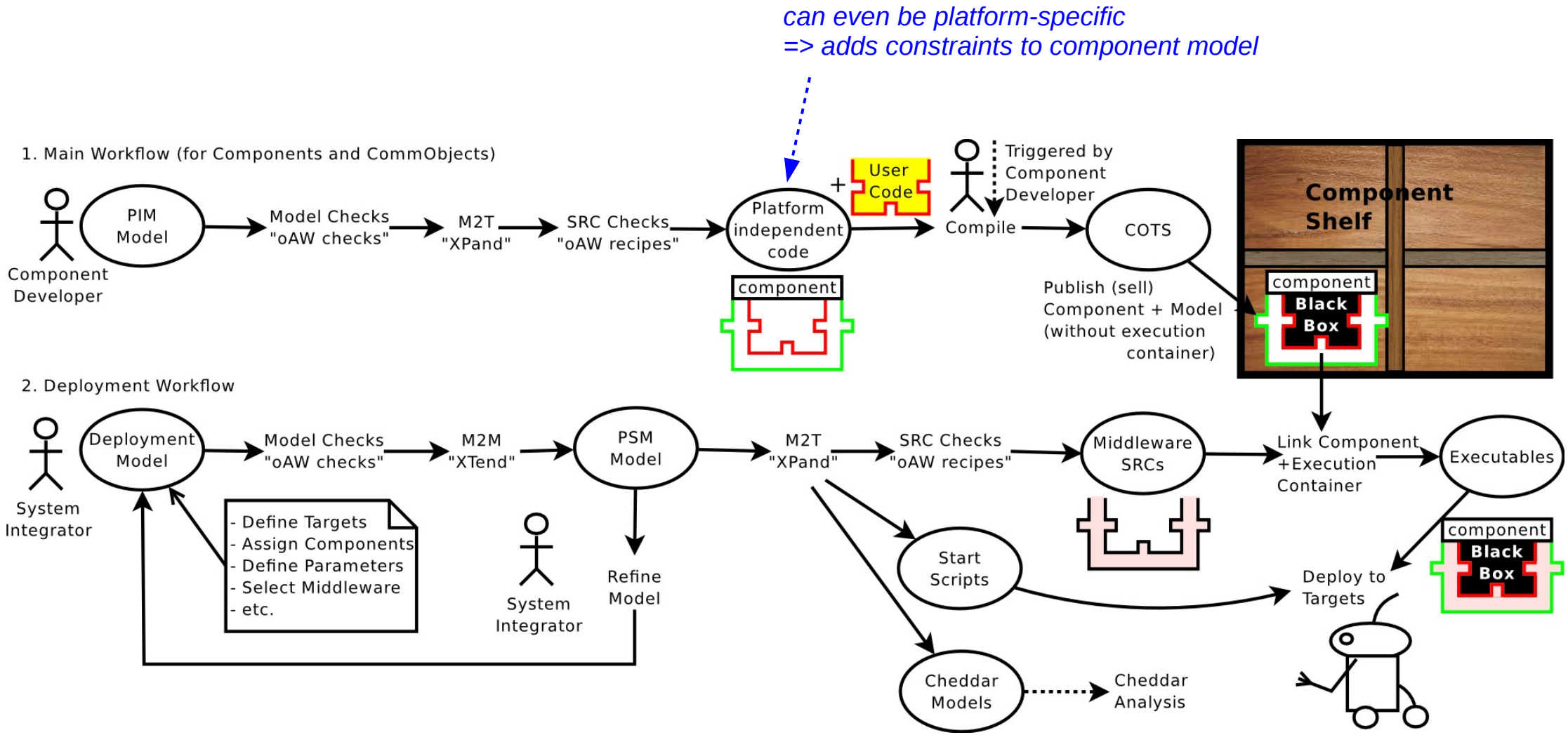
robot

SmartMARS
robot run-time view

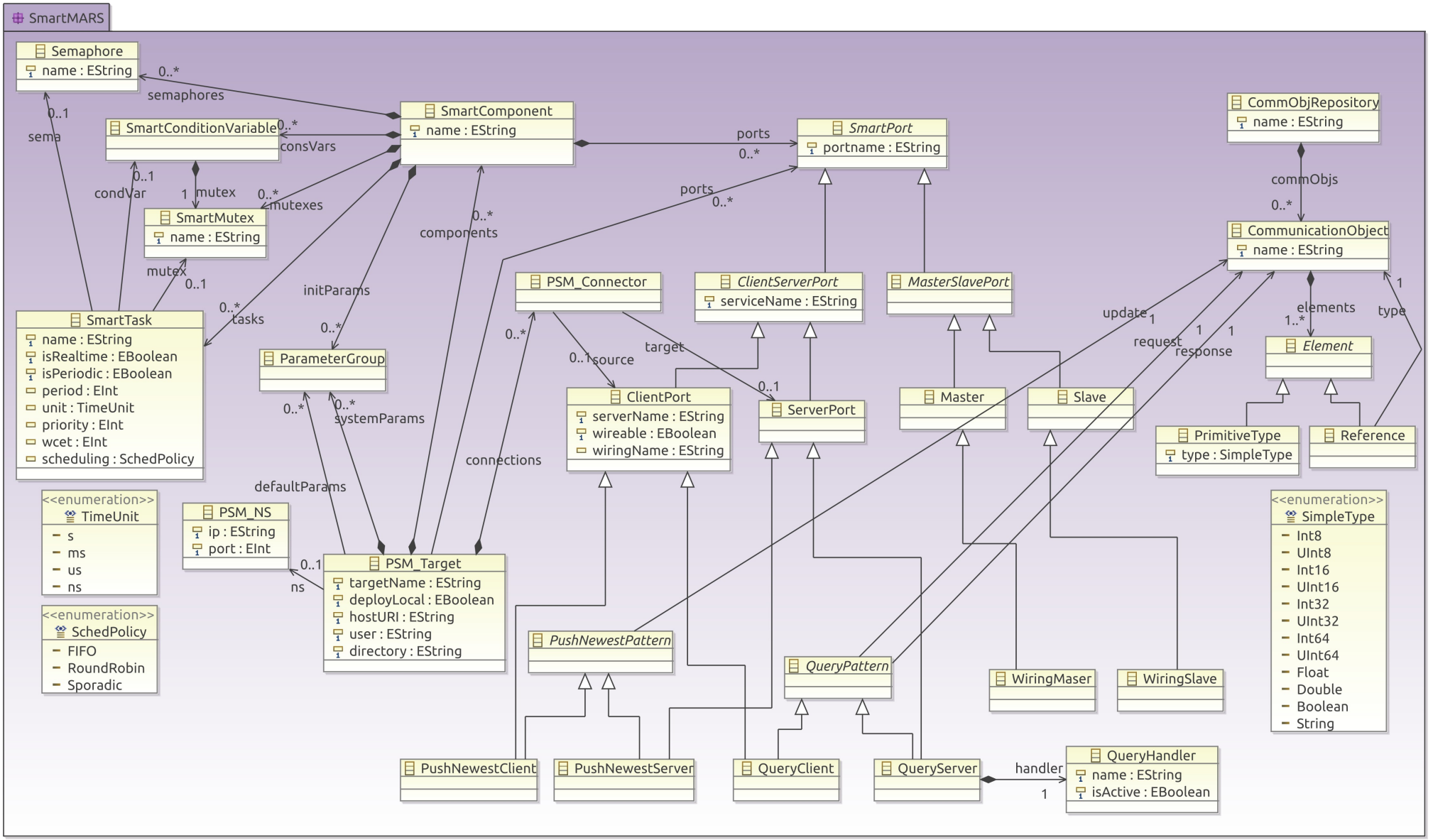
Run-Time Level: PIM => PSM => PSI

Run-time model + ***Executables***

SmartMDSD Toolchain:
Stepwise-Refinement Workflow / current work in progress

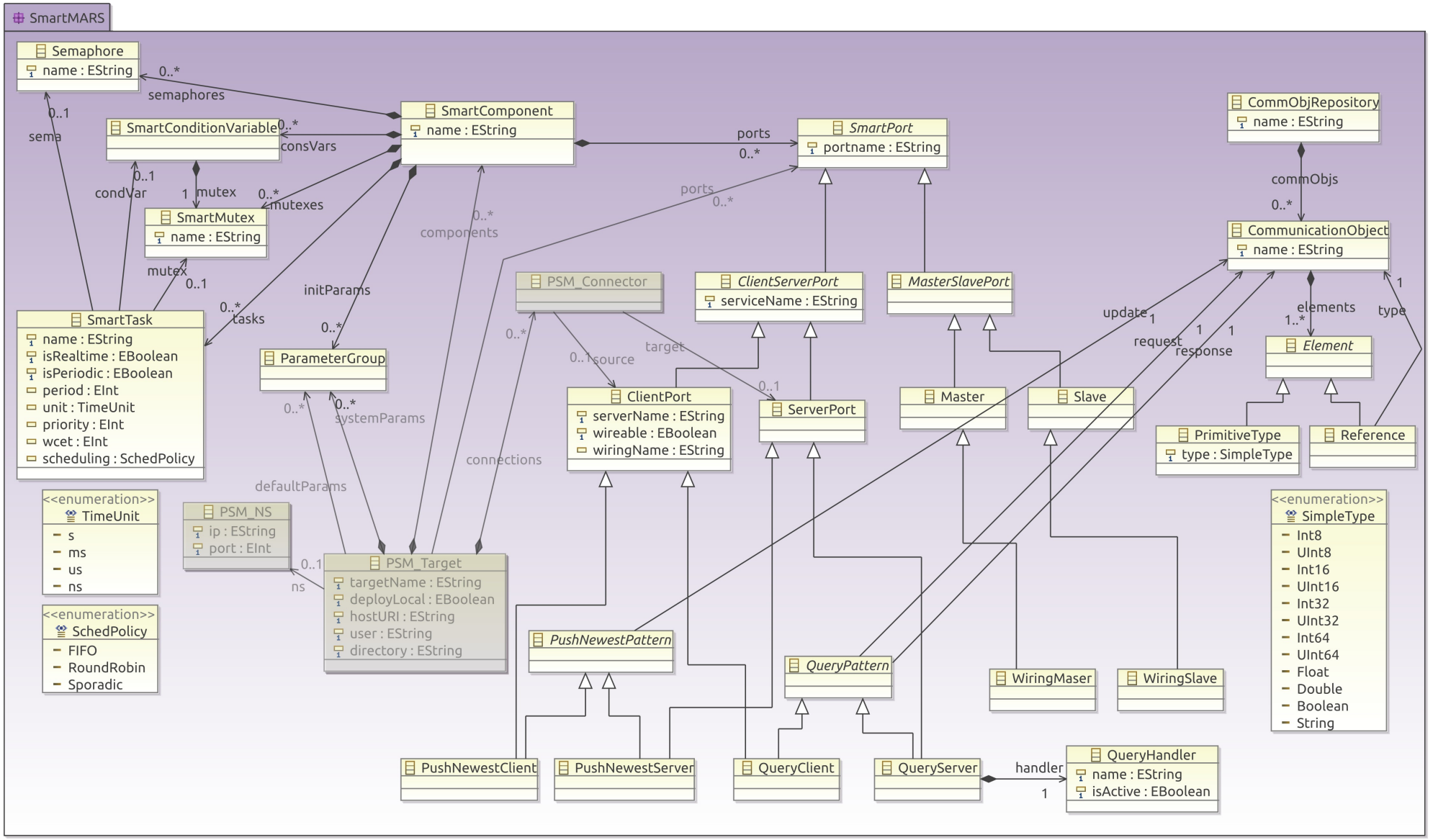


Excerpt of SmartMARS as Ecore diagram

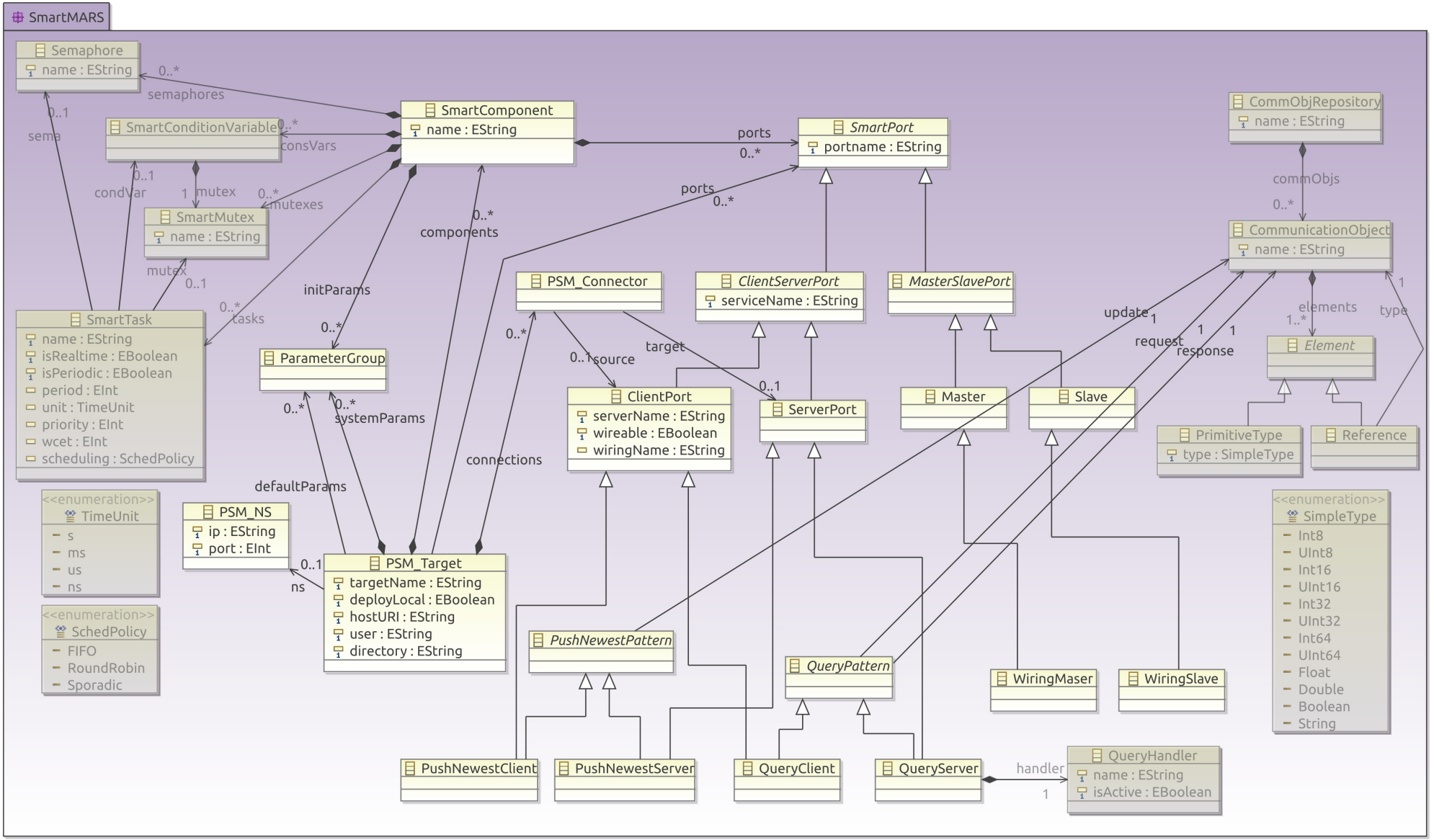


- Graphical or textual editor needs additional annotations on top of Ecore
- Methods (Ecore: operations) are not explicated in this diagram
- Not all communication patterns are shown in this diagram (event, diagnose, state, etc.)

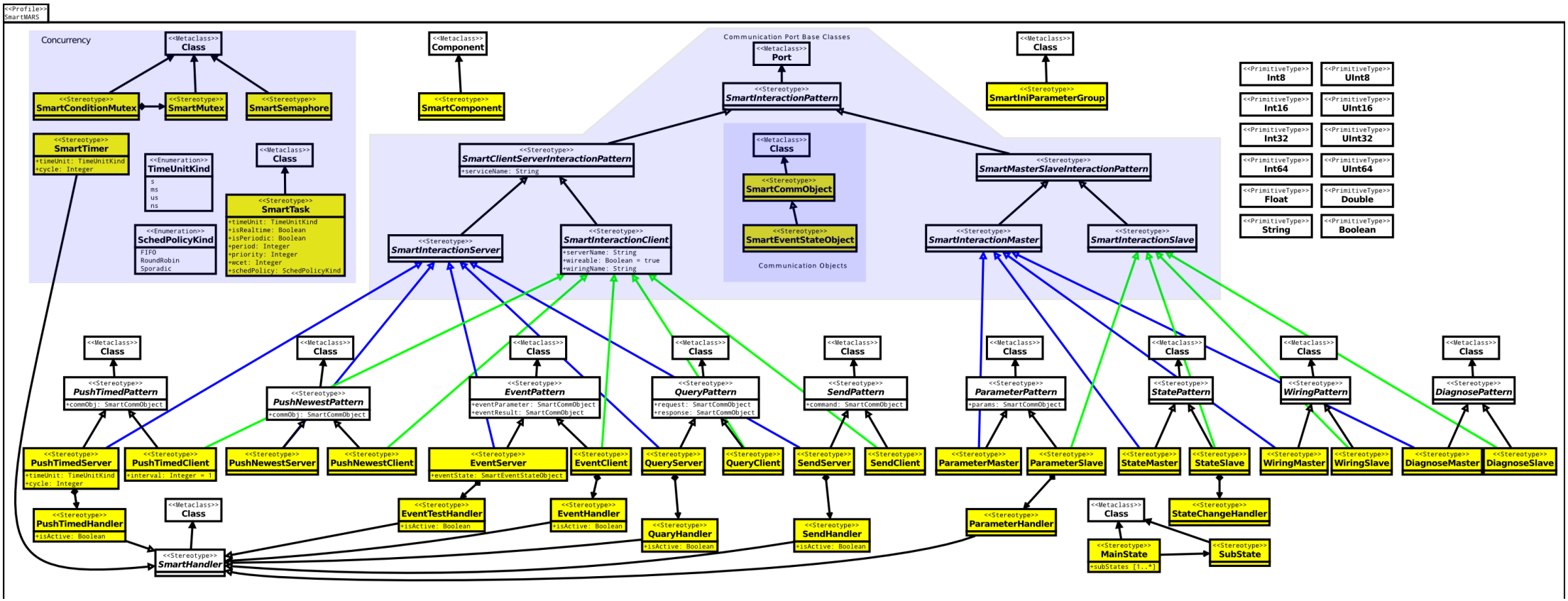
Excerpt SmartMARS / PIM: Component Builder View / Ecore



Excerpt SmartMARS / PSM: System Integrator View / Ecore

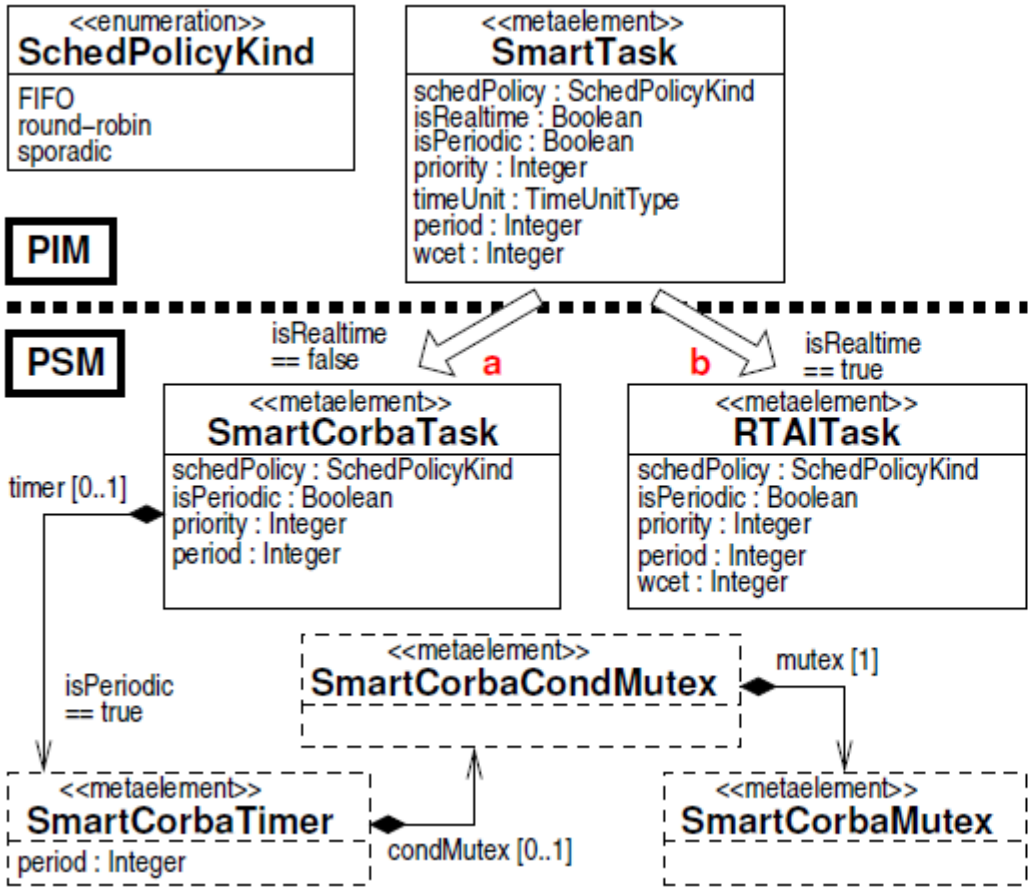


SmartMARS / PIM:
Component Builder View as UML profile

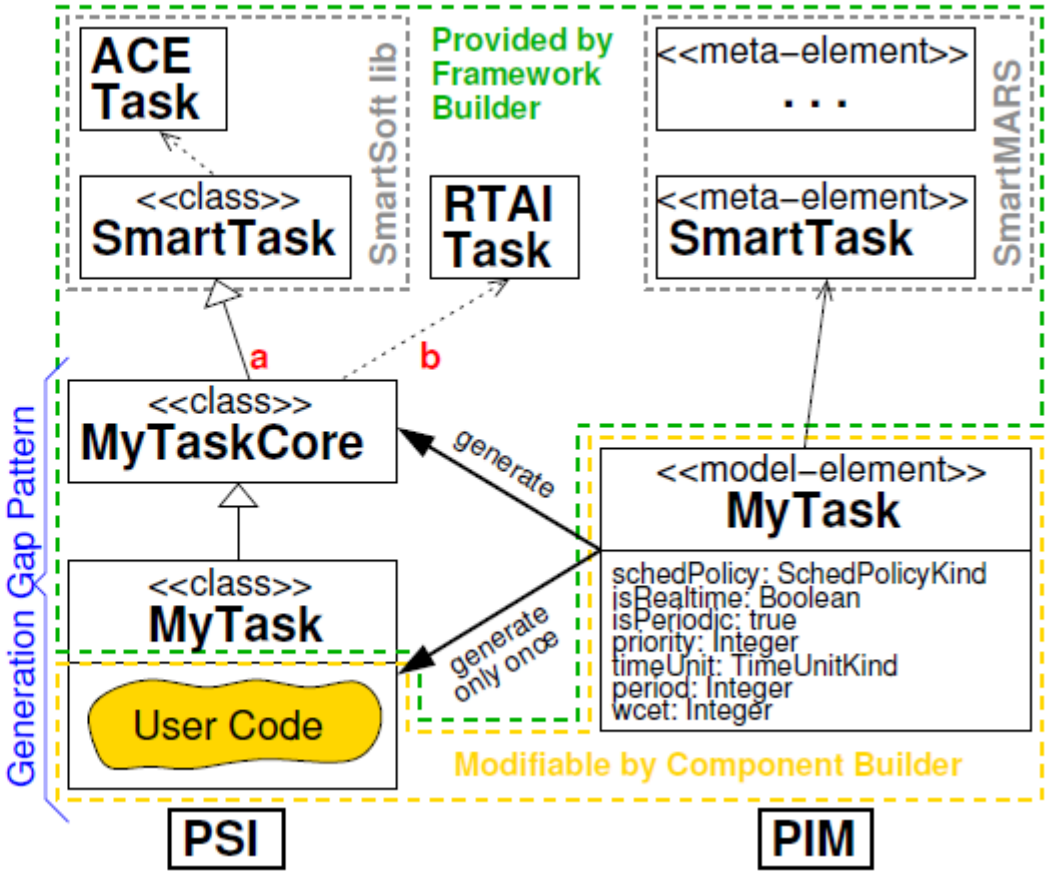


- Implementation as UML profile gives graphical editor for free
- SmartMARS does not depend on UML

Model-Driven Software Development
Model Transformation + Code Generation



Transformation PIM into PSM



Generation Gap Pattern / Technical View:

- *stable user interface [e.g. MyTask] even when platform is changed*
- *platform-specific internals / internal implementations are added transparently*

Model-Driven Software Development
PIM to PSM / SmartTask / isRealtime

task_mutex.ext

```
create uml::Class this addSmartTask(SmartMARS::SmartTask tsk, uml::Component cmp) :
    cmp.packagedElement.add(this) ->
    this.setName(tsk.name) ->
    if( tsk.isRealtime == true) then
    {
        this.applyStereotype("CorbaSmartSoft::RTAITask") ->
        setTaggedValue(this, "CorbaSmartSoft::RTAITask", "isPeriodic", tsk.isPeriodic) ->
        setTaggedValue(this, "CorbaSmartSoft::RTAITask", "wcet", tsk.wcet.toSecond(tsk.timeUnit.name)) ->
        setTaggedValue(this, "CorbaSmartSoft::RTAITask", "period", tsk.period.toSecond(tsk.timeUnit.name)) ->
        setTaggedValue(this, "CorbaSmartSoft::RTAITask", "priority", tsk.priority)
    }
    else
    {
        this.applyStereotype("CorbaSmartSoft::SmartCorbaTask") ->
        setTaggedValue(this, "CorbaSmartSoft::SmartCorbaTask", "isPeriodic", tsk.isPeriodic) ->
        setTaggedValue(this, "CorbaSmartSoft::SmartCorbaTask", "wcet", tsk.wcet.toSecond(tsk.timeUnit.name)) ->
        setTaggedValue(this, "CorbaSmartSoft::SmartCorbaTask", "period", tsk.period.toSecond(tsk.timeUnit.name)) ->
        setTaggedValue(this, "CorbaSmartSoft::SmartCorbaTask", "priority", tsk.priority) ->
        if( tsk.isPeriodic == true ) then
        {
            setTaggedValue(this, "CorbaSmartSoft::SmartCorbaTask", "timer", cmp.addTimer(tsk.name, tsk.period, tsk.timeUnit.name))
        }
    };
```

Xtend Transformation Rule (M2M):
PIM to PSM model transformation of the SmartTask depending on the attribute “isRealtime”

Model-Driven Software Development
PSM to PSI

smartTask.xpt — PSM → PSI Template

```
«DEFINE TaskUserSourceFile FOR CorbaSmartSoft::Task-»
«FILE this.getUserSourceFilename() writeOnce-»
«getCopyrightWriteOnce()»
#include "«this.getUserHeaderFilename()»"
#include "gen/«((CorbaSmartSoft::SmartCorbaComponent)this.eContainer()).getCoreHeaderFilename()»"

#include <iostream>

«this.getName()»::«this.getName()»()
{
    std::cout << "constructor «this.getName()»\n";
}

int «this.getName()»::svc()
{
    // do something -- put your code here !!!
    while(1)
    {
        «IF this.isPeriodic == true-»
        std::cout << "Hello from «this.getName()»  - periodic\n";
        smart_task_wait_period();
        «ELSE-»
        std::cout << "Hello from «this.getName()»\n";
        sleep(1);
        «ENDIF-»
    }
    return 0;
}
«ENDFILE»
«ENDDEFINE»
```

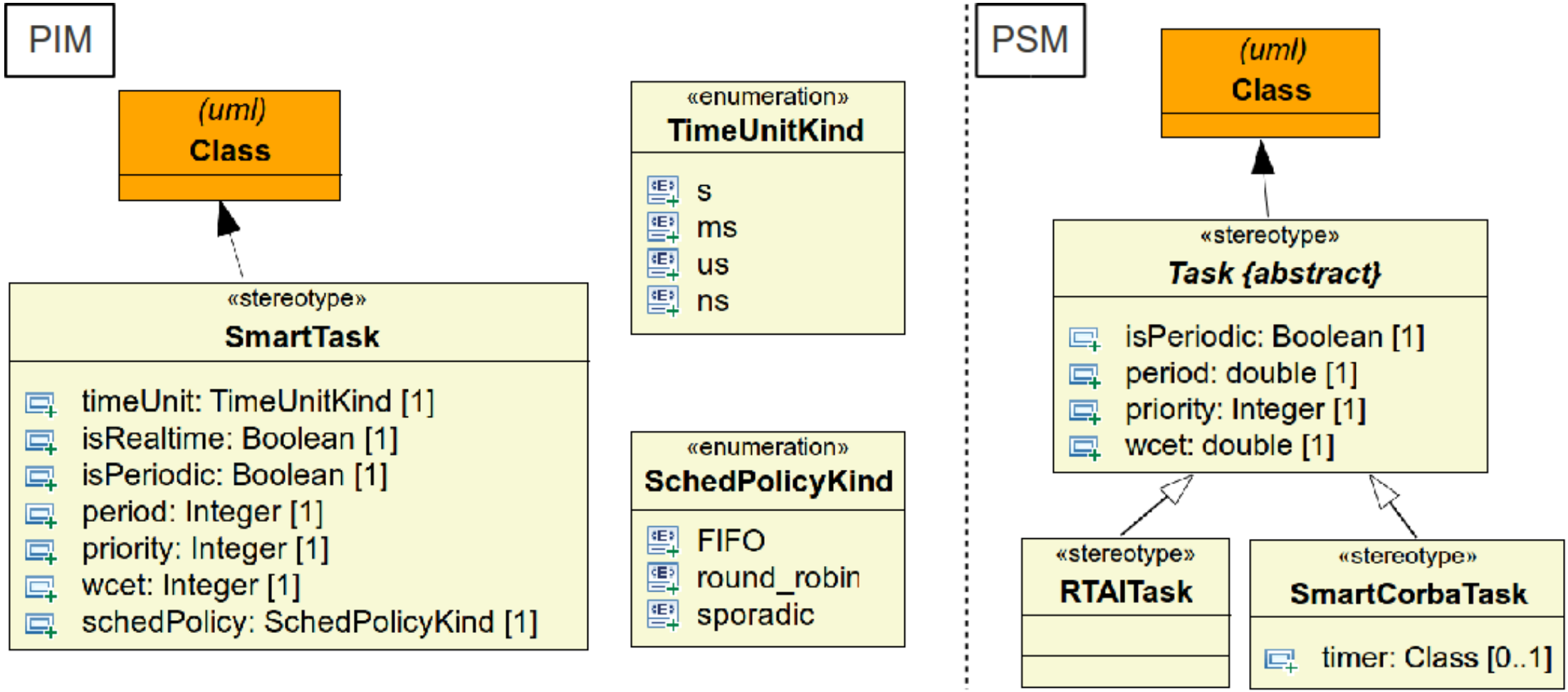
ServoTask.cc — PSI (user code .cc file)

```
#include "ServoTask.hh"
#include "gen/SmartServo.hh"

#include <iostream>

ServoTask::ServoTask()
{
    std::cout << "constructor ServoTask\n";
}

int ServoTask::svc()
{
    // do something -- put your code here !!!
    while (1)
    {
        std::cout << "Hello from ServoTask  - periodic\n";
        smart_task_wait_period();
    }
    return 0;
}
```

Xpand / Xtend Transformation (M2T): PSM to PSI model transformation

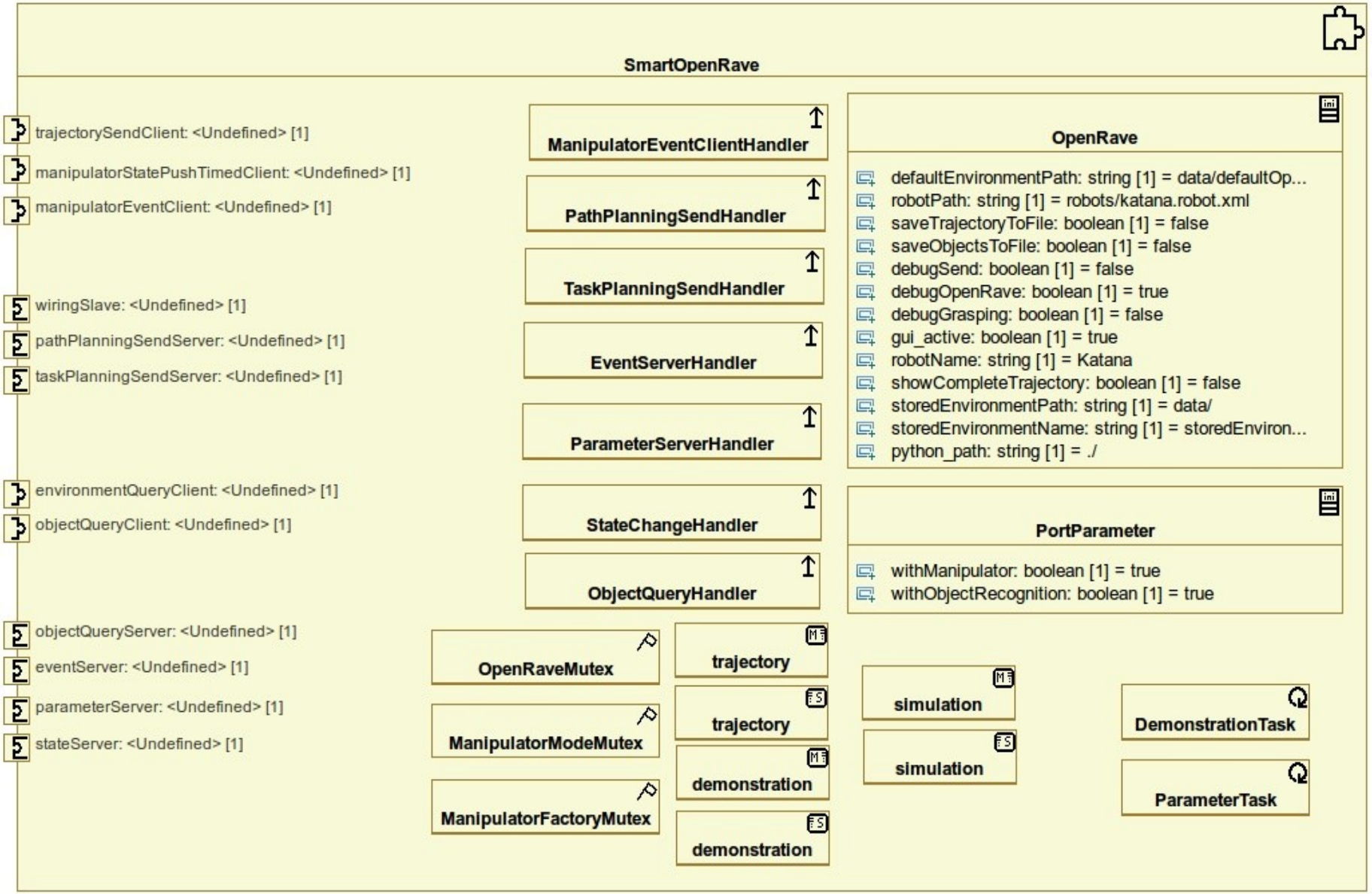
Additional Slides

Model-Driven Software Development
SmartMARS UML Profiles (PIM, PSM)



excerpts of UML Profile created with Papyrus UML (left PIM, right PSM)

SmartOpenRave Component



SmartOpenRave Component

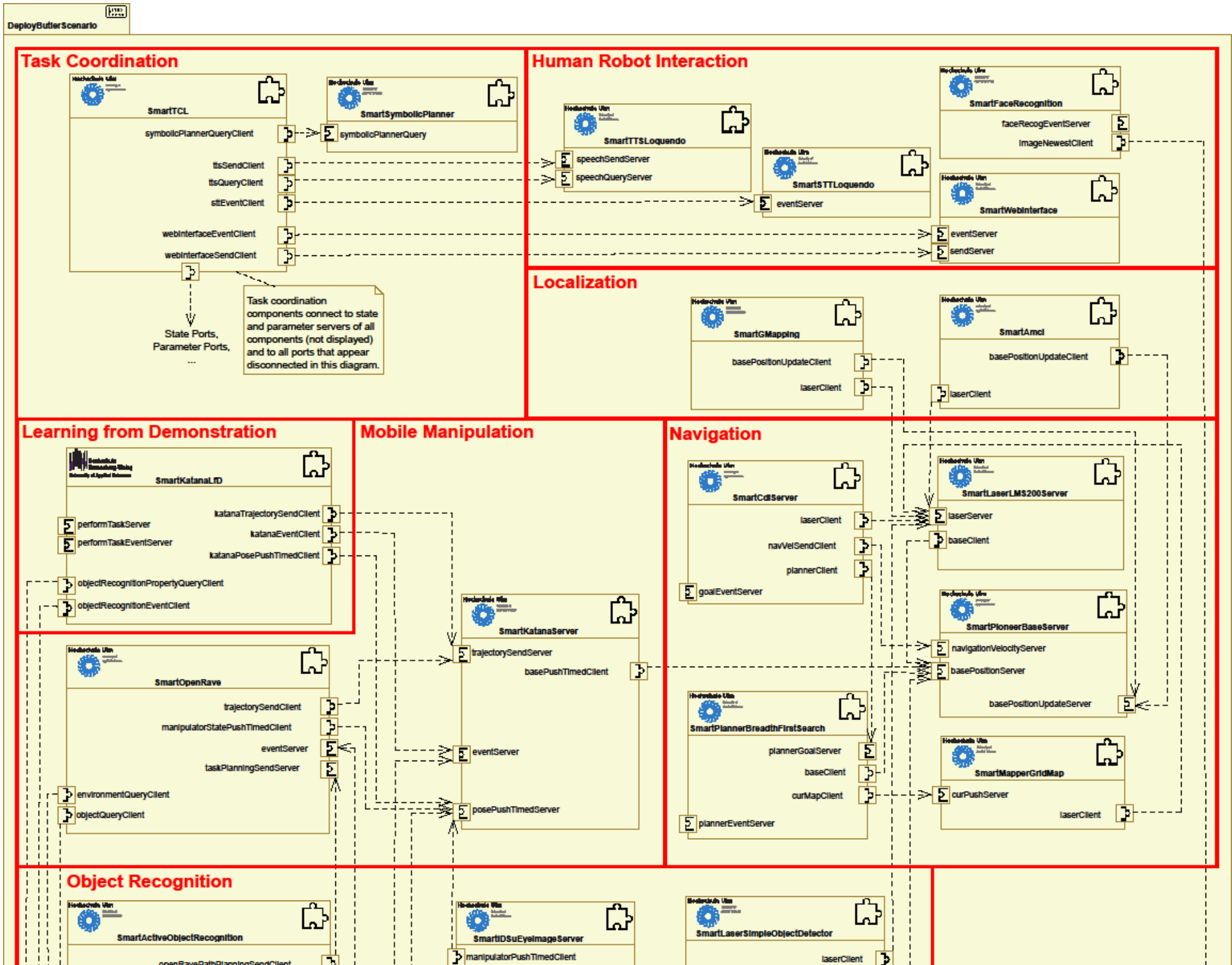
States

High level description of states. See Services for a detailed description how individual ports behave in specific states.

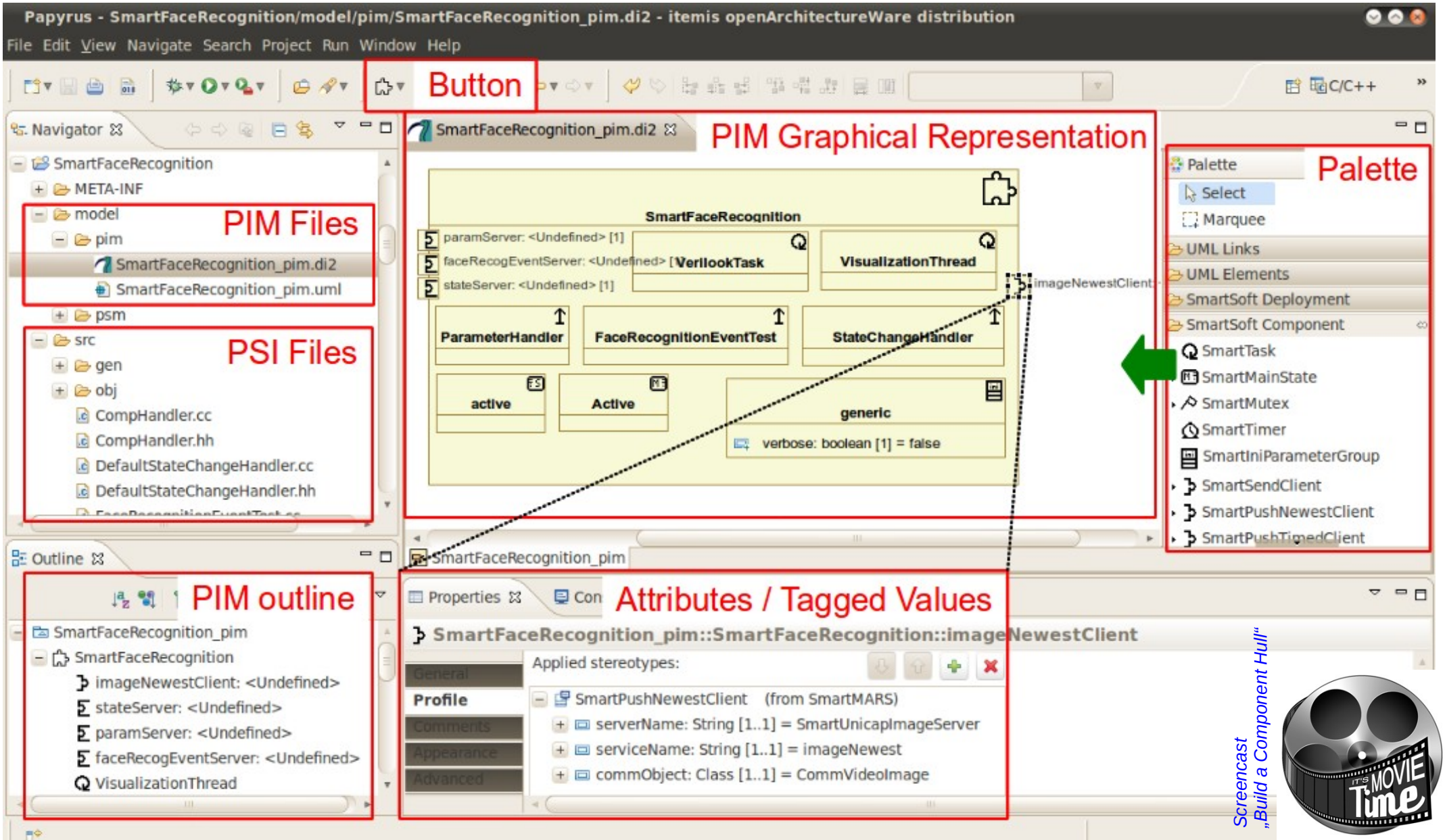
neutral:	The component does not perform any planning or IK calculation. It accepts parameters.
trajectory:	The component can plan paths or plan higher level tasks like grasping an object and place it somewhere.
demonstration:	The component just synchronizes the modeled manipulator with the real manipulator. This state is mainly for testing purpose.
simulation:	The component does not send any trajectory to the real manipulator. It computes all IK solutions and plans path as in "trajectory" state.

Parameters

ENV_CLEAR:	The scene is reset to its default as loaded initially from the ini-configuration.
ENV_LOAD_OBJECTRECOGNITION(?envid):	The environment with id ?envid is loaded from the object recognition. The environmentQueryClient is used to get the environment from the object recognition.

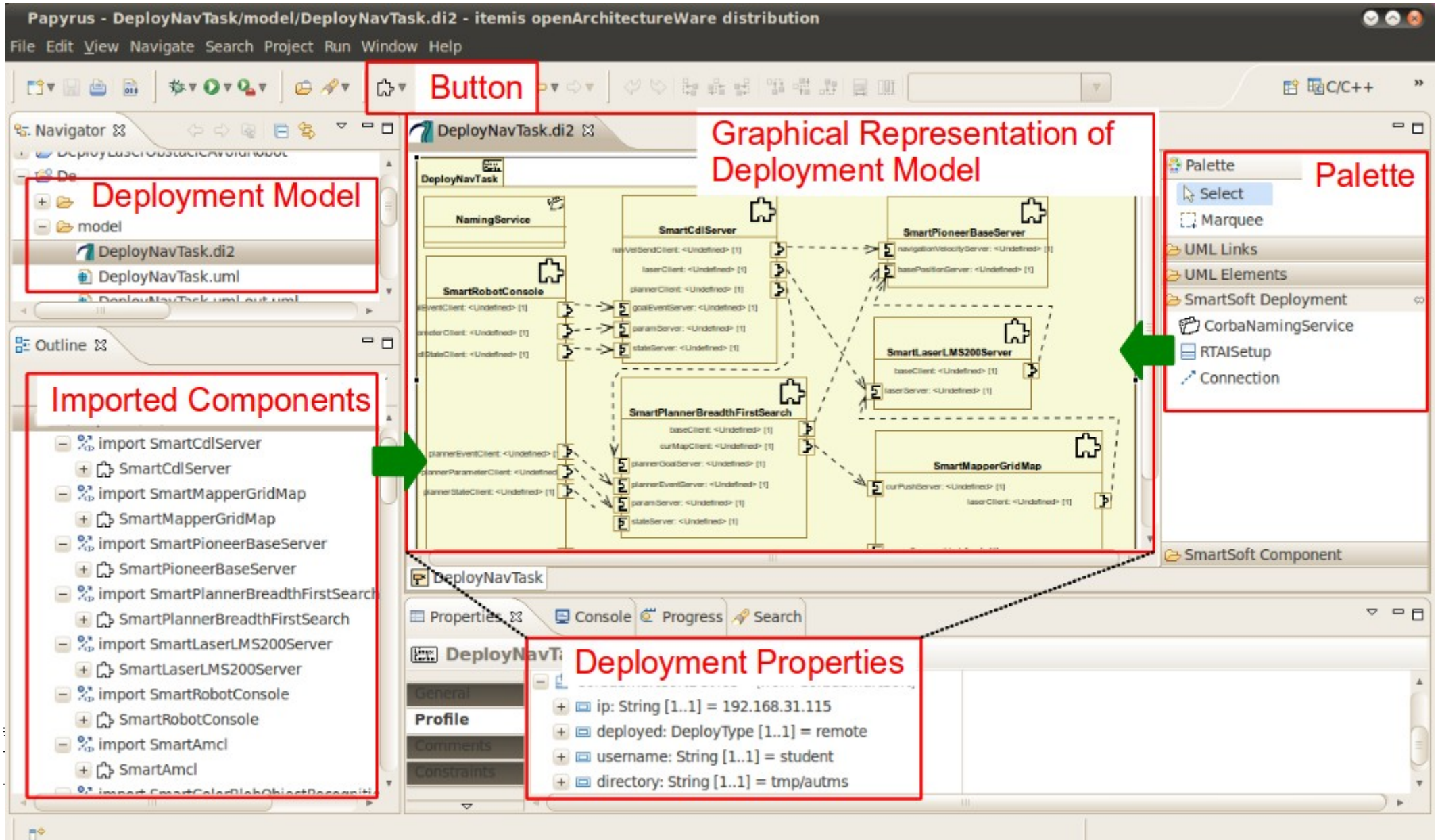


Model-Driven Software Development: Component Builder View

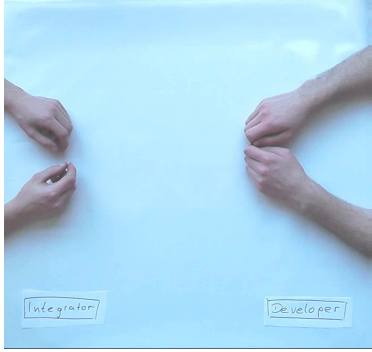


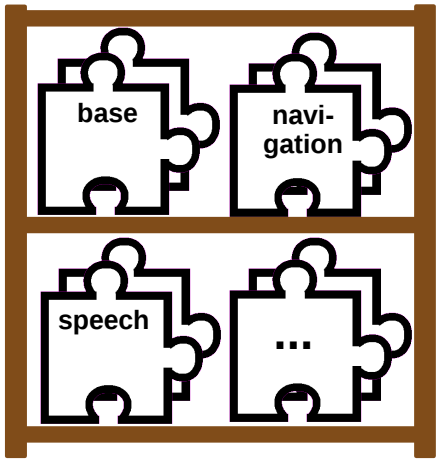
Screencast „Build a Component Hull“

Model-Driven Software Development: System Integrator View



Model-Driven Software Development: System Integrator View





Component Shelf
Reusable Components

System Level Properties / Bindings / Conformance Checks