



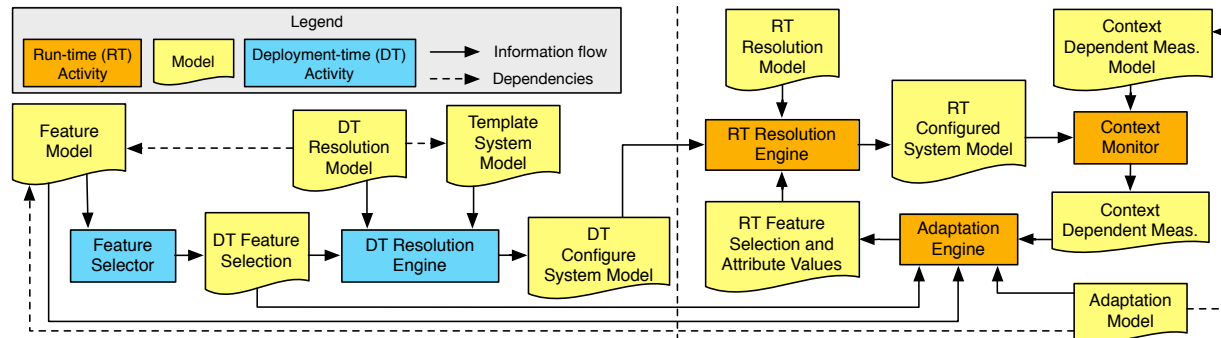
Addressing Deployment-time and Run-time Variability in Robotics Software Systems

Luca Gherardi

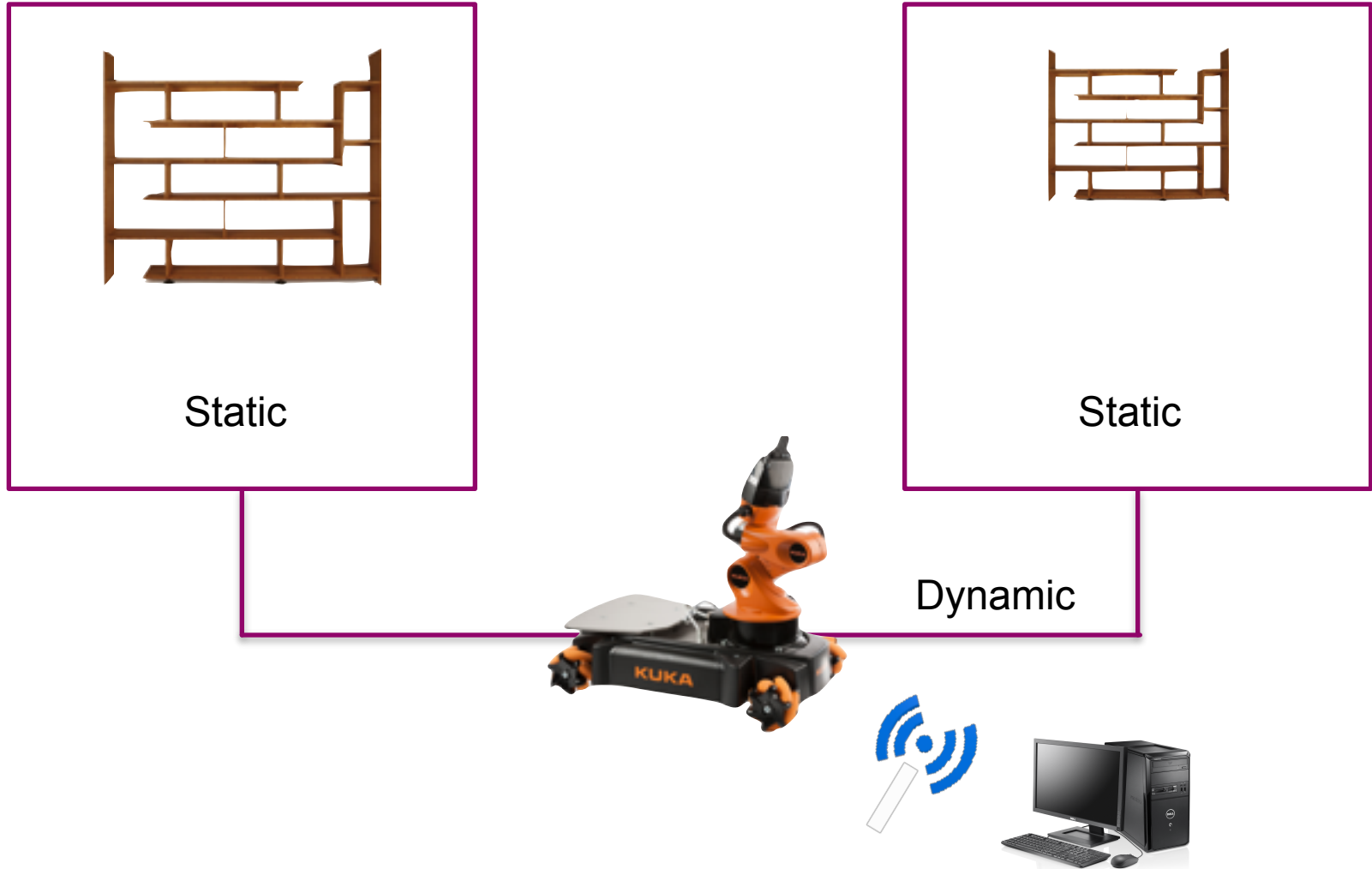
Bergamo - October 20th, 2014

Approach overview

- Problem
 - Huge amount of variability makes hard the configuration of robotics software architectures
 - Run-time variability resolution (aka run-time adaptation) is often hard-coded in the functional components
- Goal
 - Model variability and context information
 - To configure and adapt software architecture
 - Based on deployment-time requirements and state of the context



A possible scenario



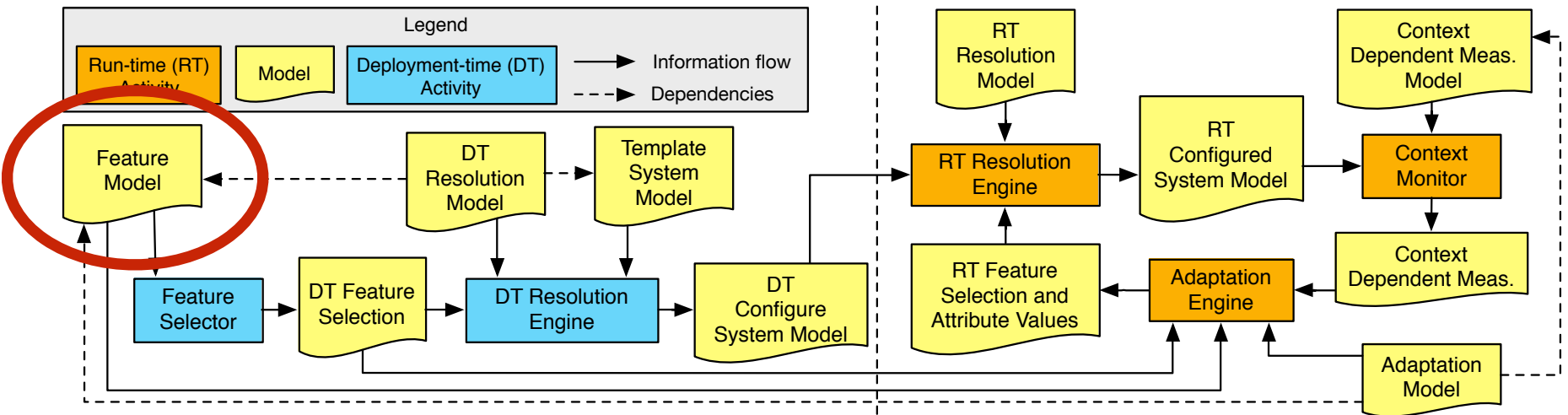
Deployment-time Variability Resolution

Luca Gherardi and Davide Brugali

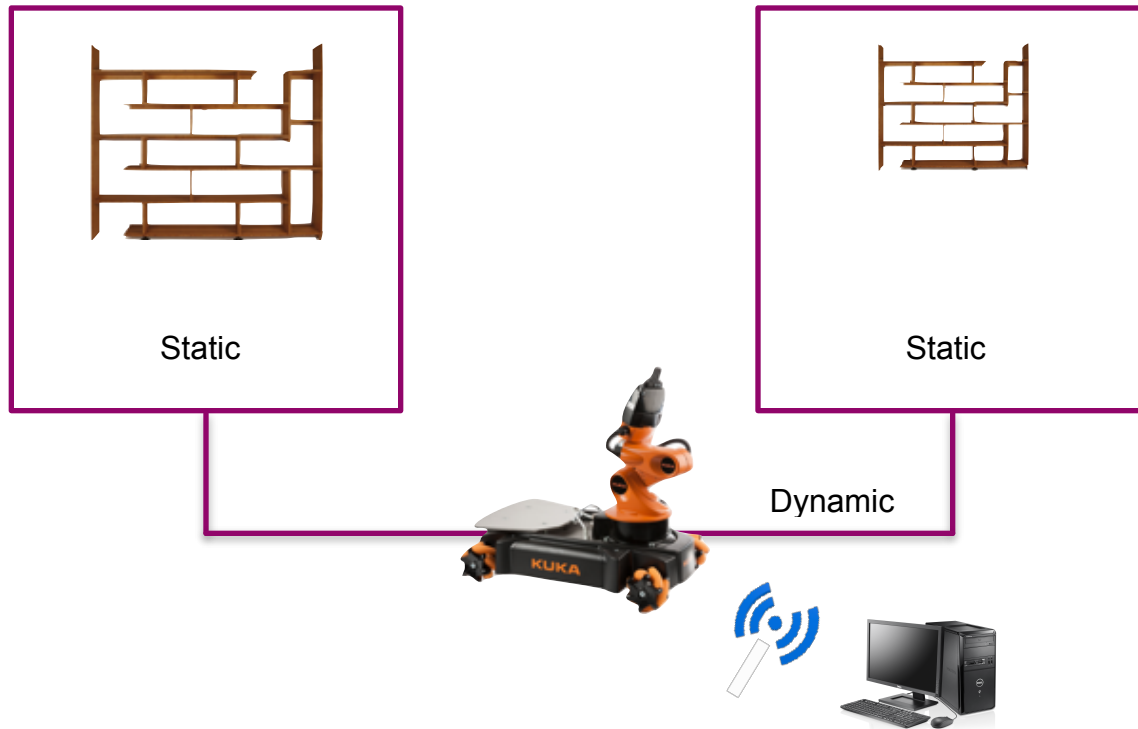
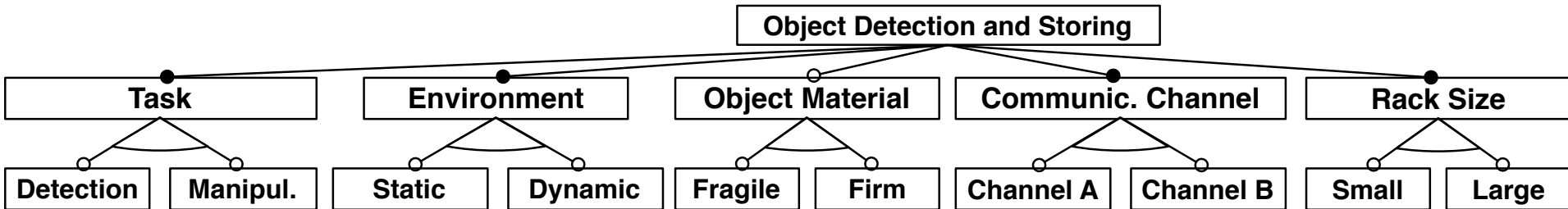
Configure architecture based
on the user's requirements



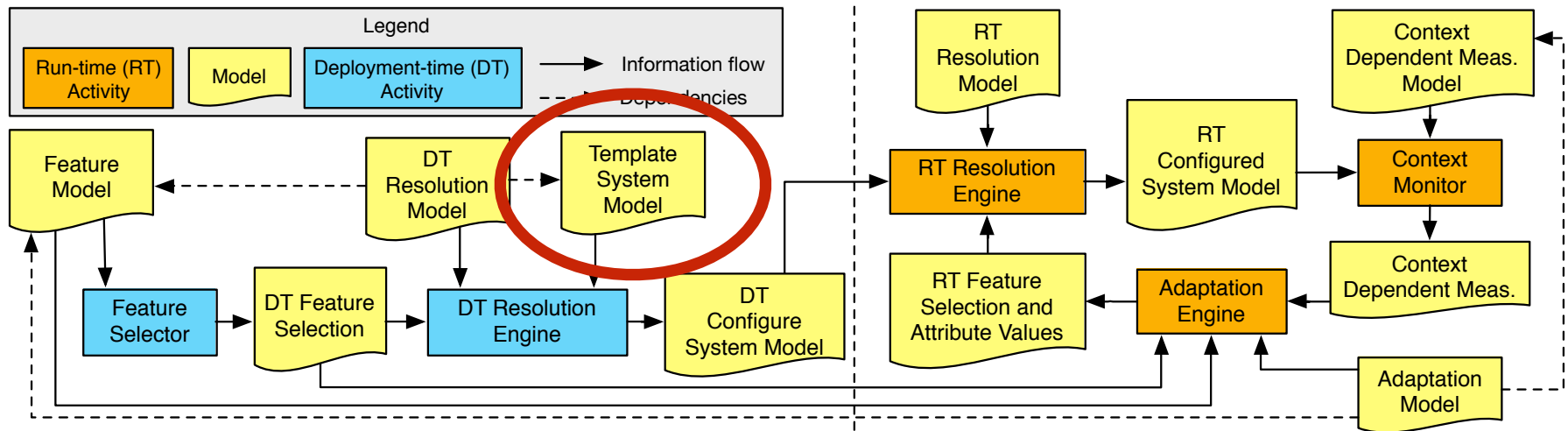
Feature Model



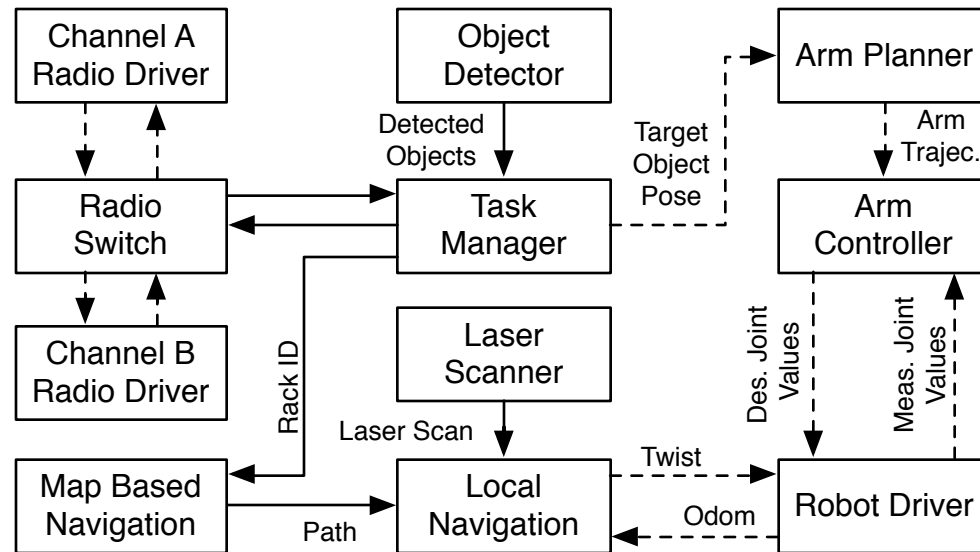
Feature Model



Template System Model

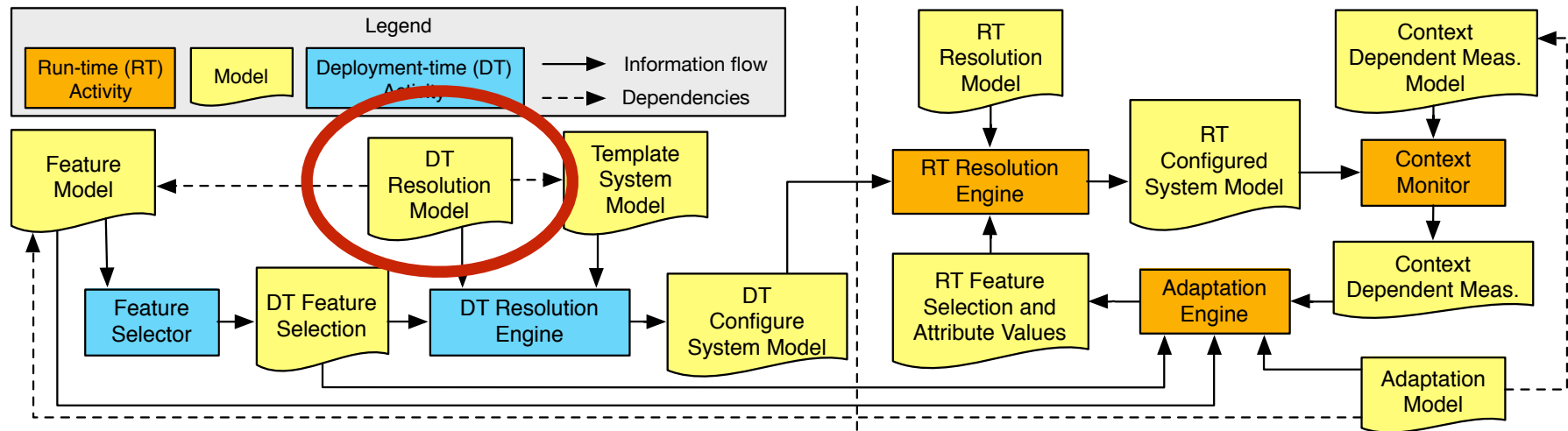


Template System Model

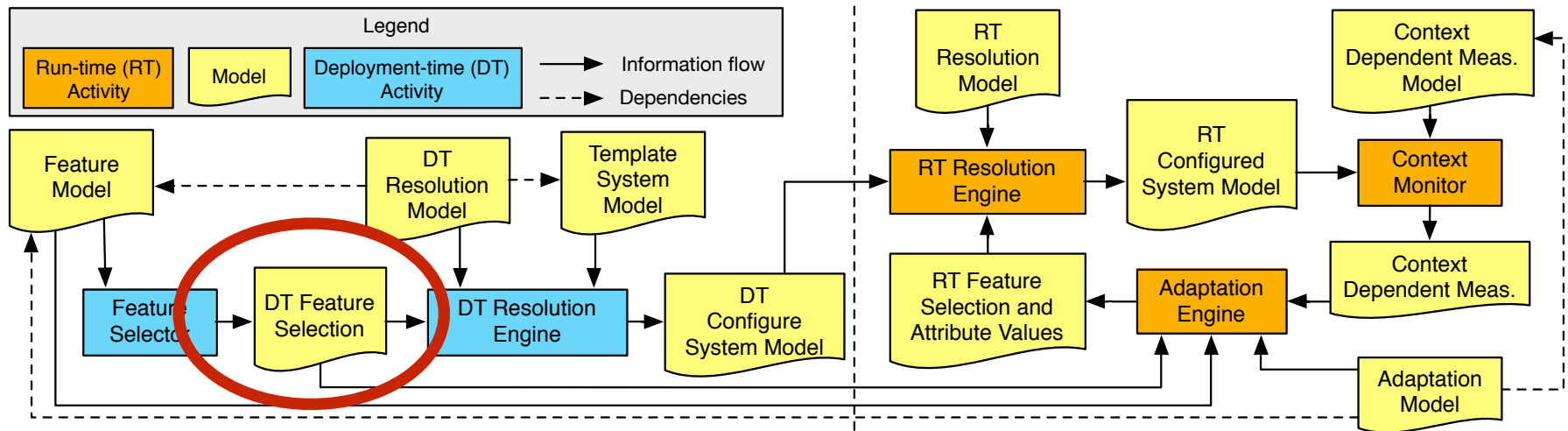


- 3 types of configurations
 - Change component's implementation
 - Change connections
 - Set properties

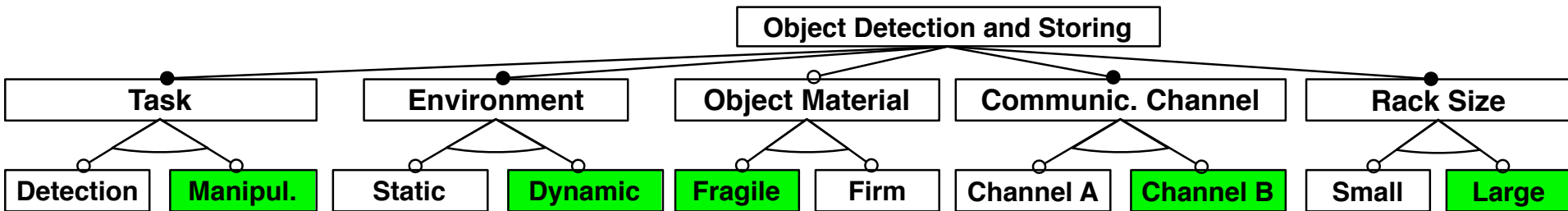
Deployment-time Resolution Model



Deployment-time Feature Selection



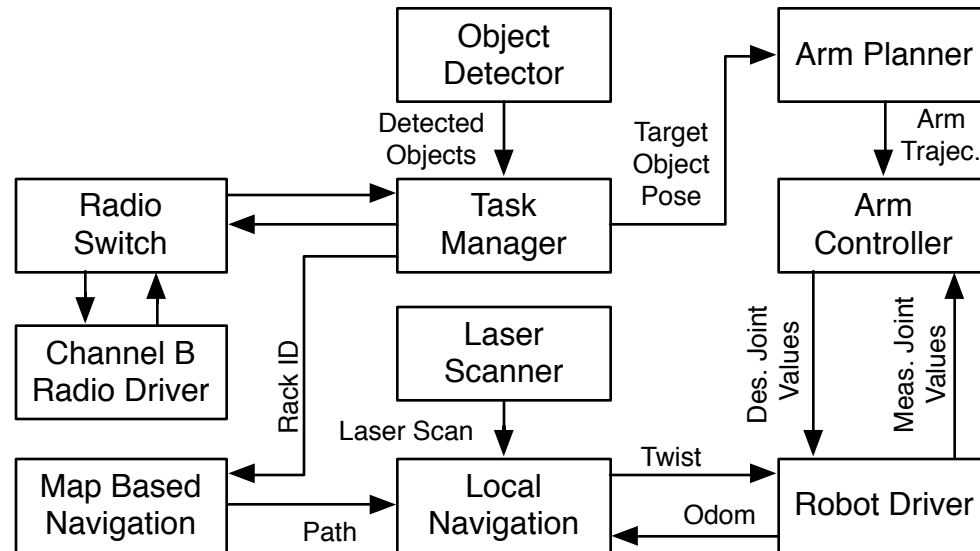
Deployment-time Feature Selection



- The feature selection is sent to the Deployment-time Resolution Engine



Deployment-time configured system model



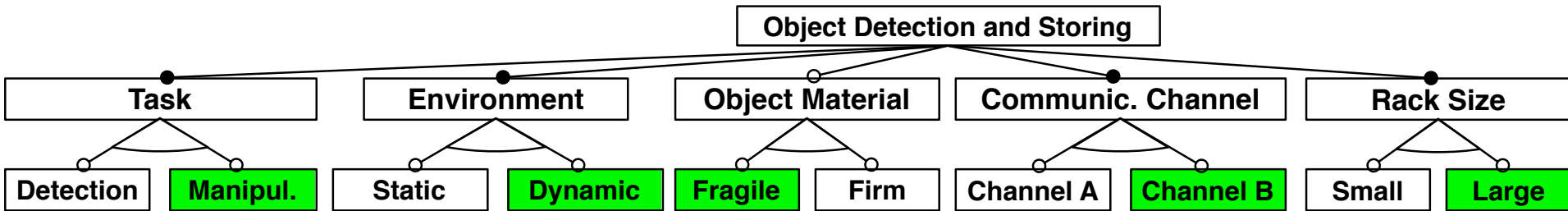
Run-time Variability Resolution

Luca Gherardi and Nico Hochgeschwender

Adapt run-time architecture based
on the state of the context



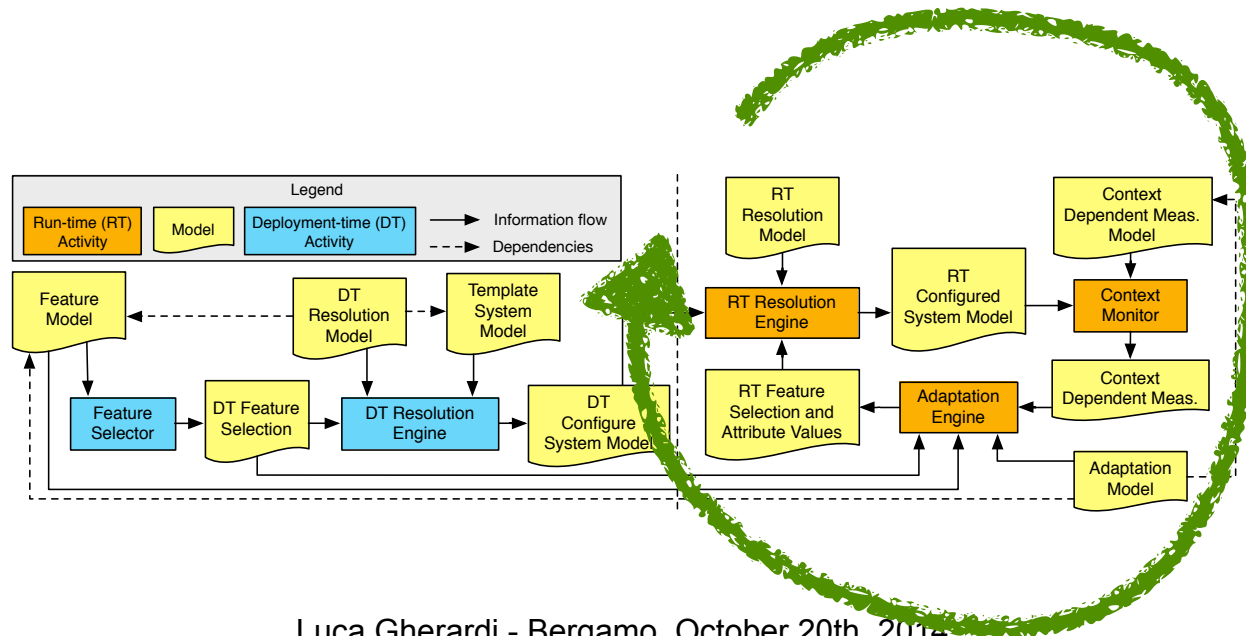
Deployment-time Feature Selection



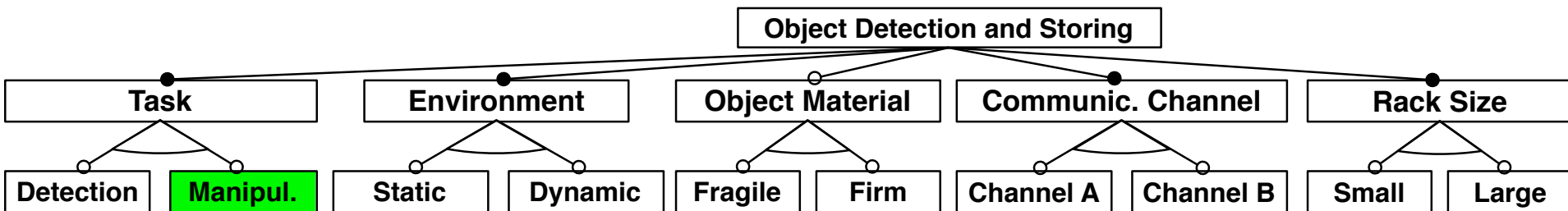
- Some of the features can be better chosen at run-time
 - More information about the context is available
- We have to model this information and reason about it

Approach overview

- The run-time variability resolution is a feed-back loop continuously executed
- It is based on the architecture defined in the Deployment-time Configured System Model

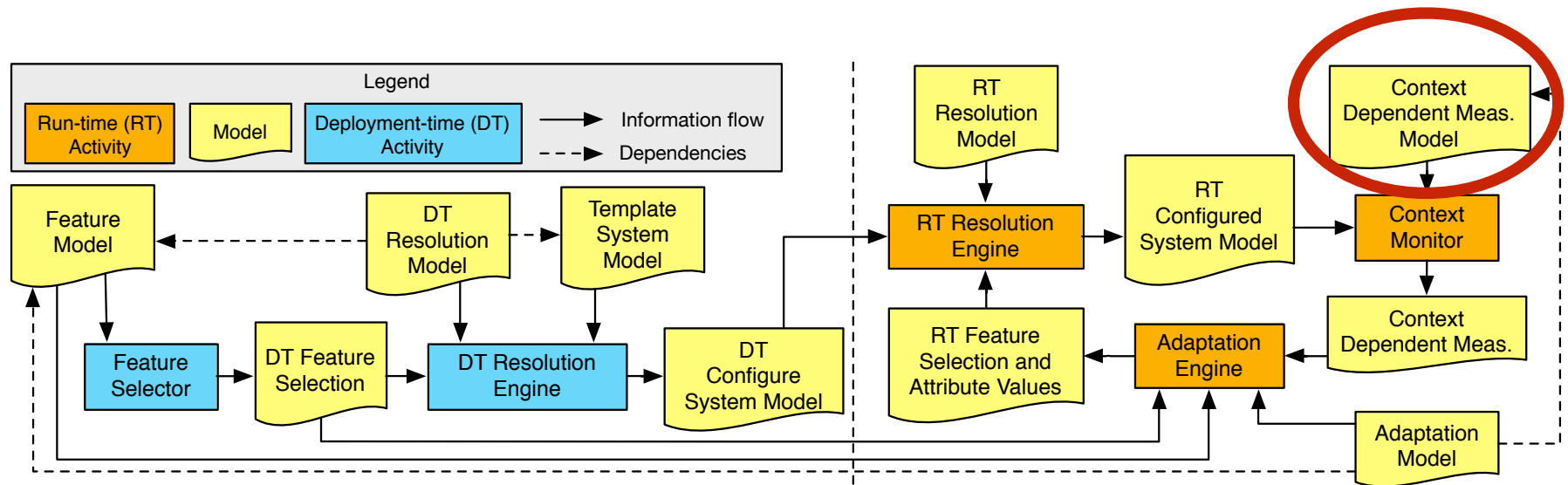


Deployment-time Feature Selection



Feature	Attribute	Value
Channel A	Latency	Low
Channel A	Power Consumption	High
Channel B	Latency	High
Channel B	Power Consumption	Low

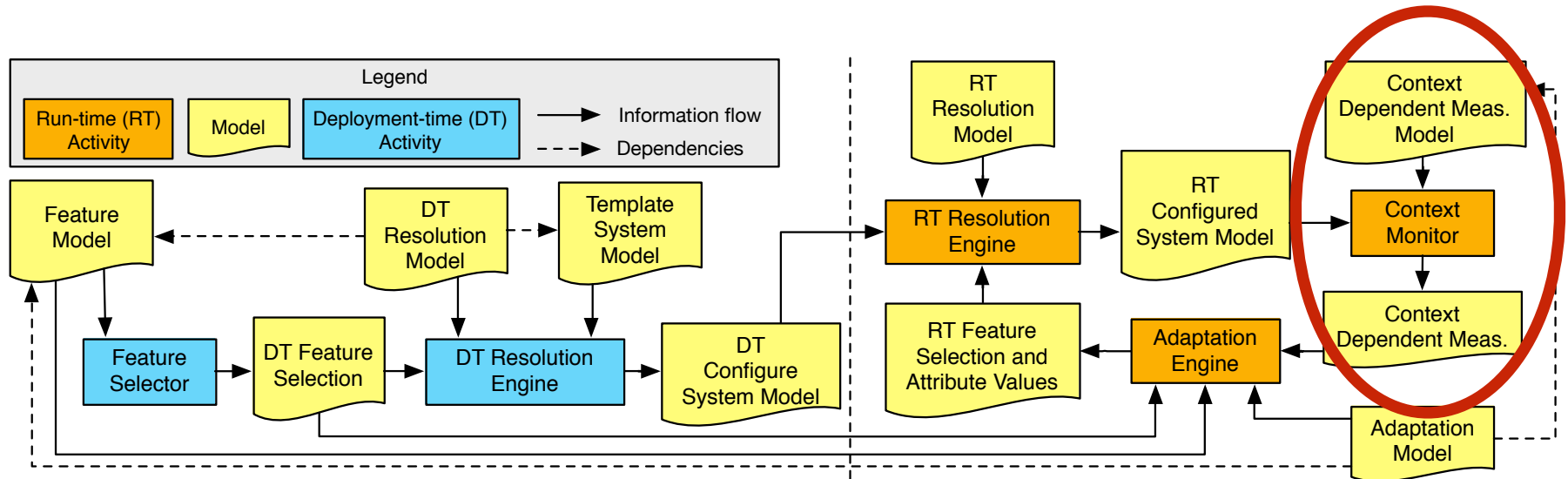
Context Dependent Measurement Model



Context Dependent Measurement Model

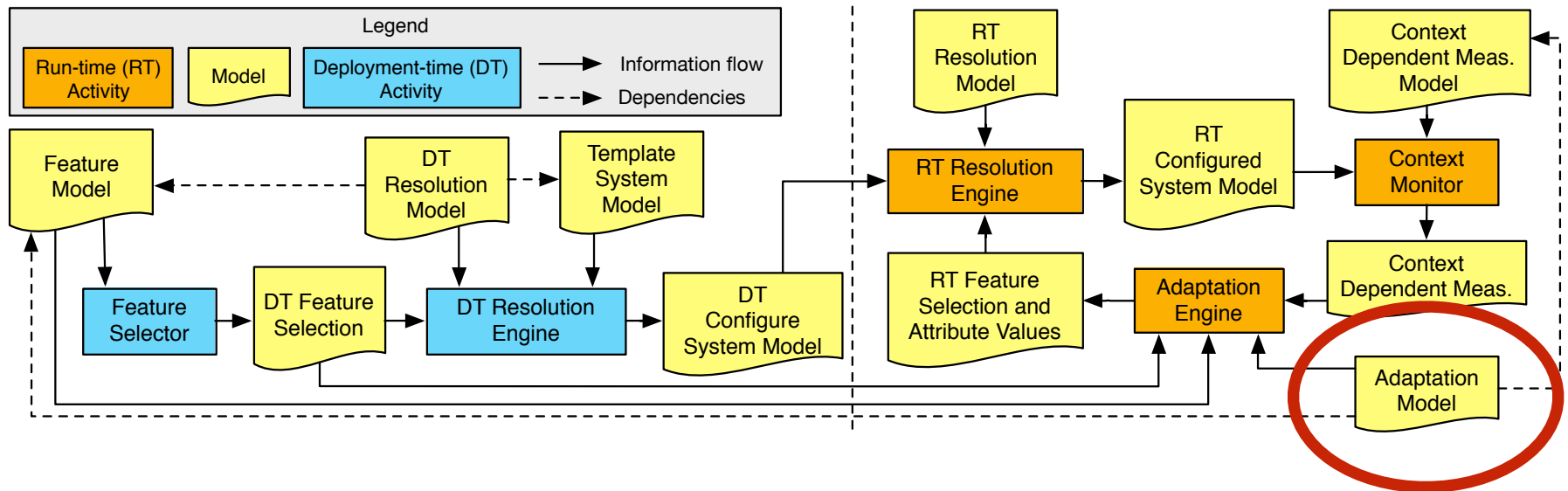
CDM	Component	Interace	Function
BattLev	Robot Driver	Battery Level	—
SemLoc	Map Based Nav.	Semantic Location	—
ObstDist	Laser Scanner	Laser Scan	getClosestObst
Material	Radio Switch	Object Material	—

Context Monitor



- A ros-cpp node is automatically generated based on the Context Dependent Measurements Model
 - Listens for components outputs
 - Produces as output Context Dependent Measurements

Adaptation Model



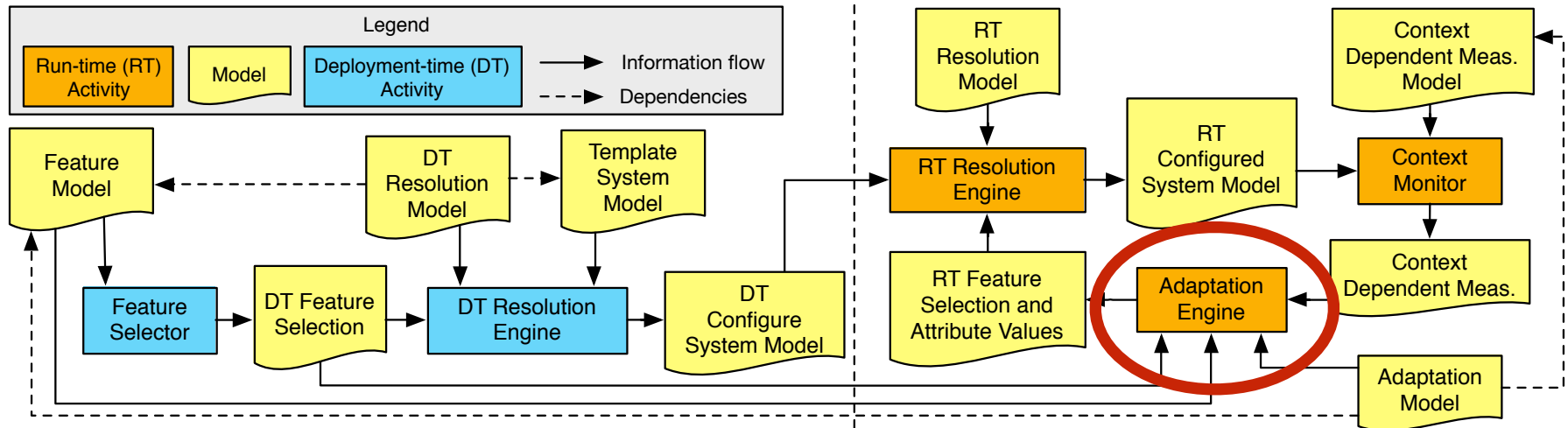
Adaptation Model

```
import caseStudy.featureModel
import caseStudy.contextDepMeasModel as cdm

rule RackSize:
  if( cdm.SemLoc == "RoomA")
    activate_feature(Small)
  else if( cdm.SemLoc == "RoomB")
    activate_feature(Large)
  else deactivate_feature((Large AND Small))

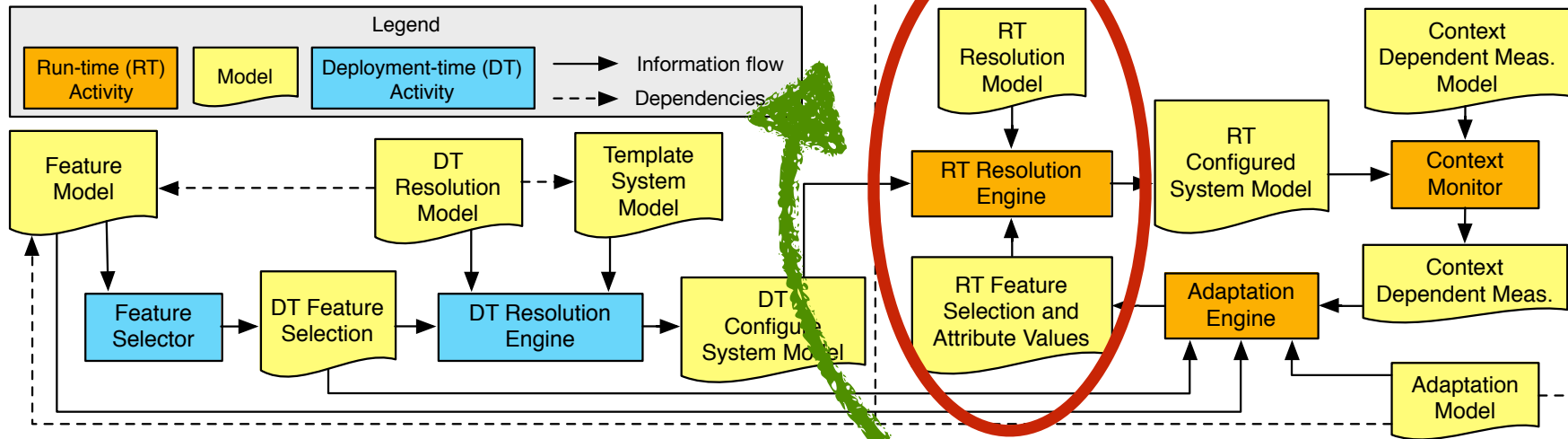
rule Radio:
  if( cdm.batteryLevel > "50" )
    select_feature_from_variants_of(Communic. Channel)
    where_attribute MIN(Latency)
  else select_feature_from_variants_of(Communic. Channel)
    where_attribute MIN(PowerConsump)
```

Adaptation Engine



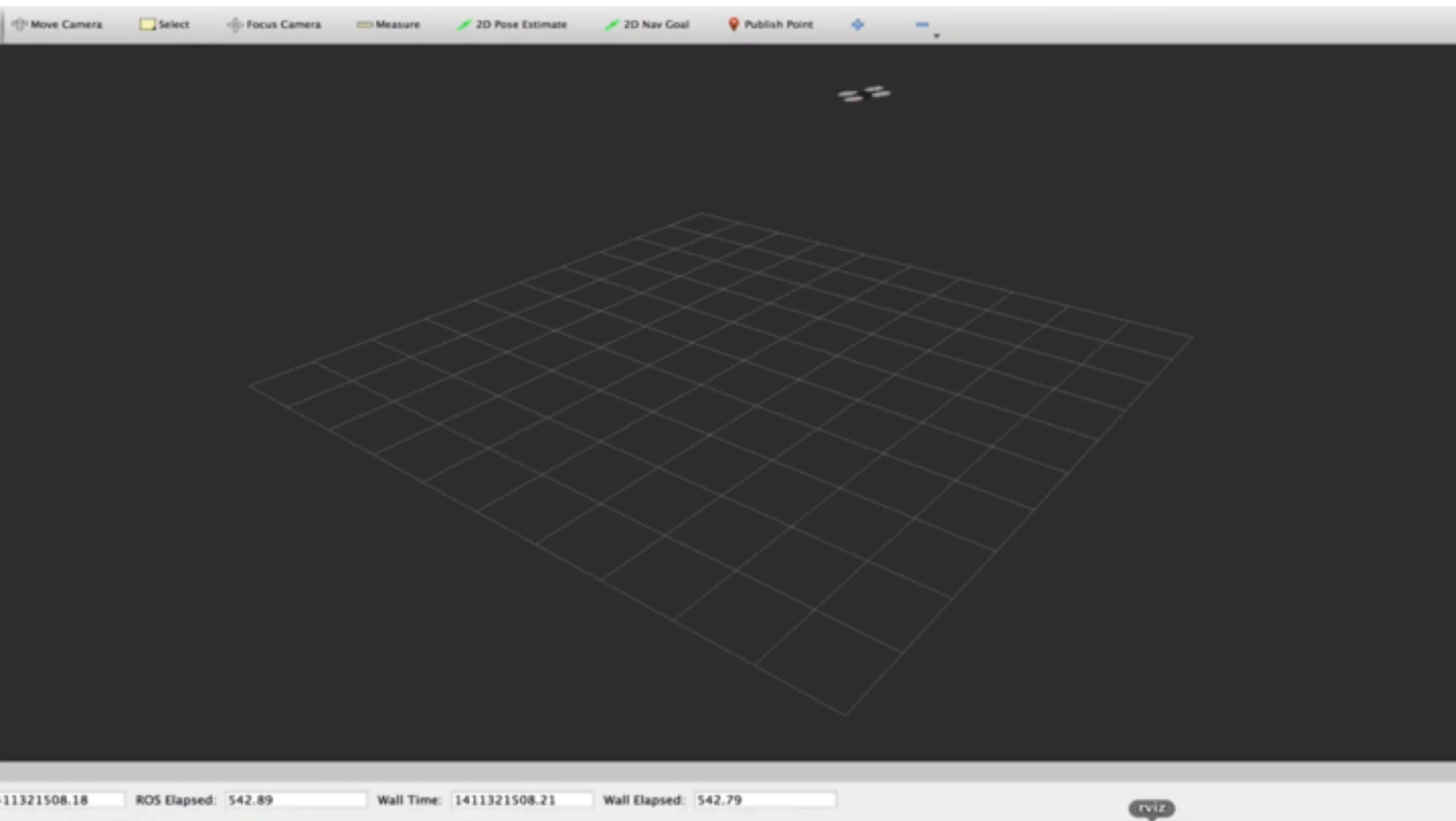
- A ros-java node is automatically configured based on the Adaptation Model
 - Listens for Context Dependent Measurements
 - Produces as output a selection of features

Run-time Resolution Engine



- 3 types of transformations
 - Start/stop components
 - Change connections
 - Set properties

A simulation example

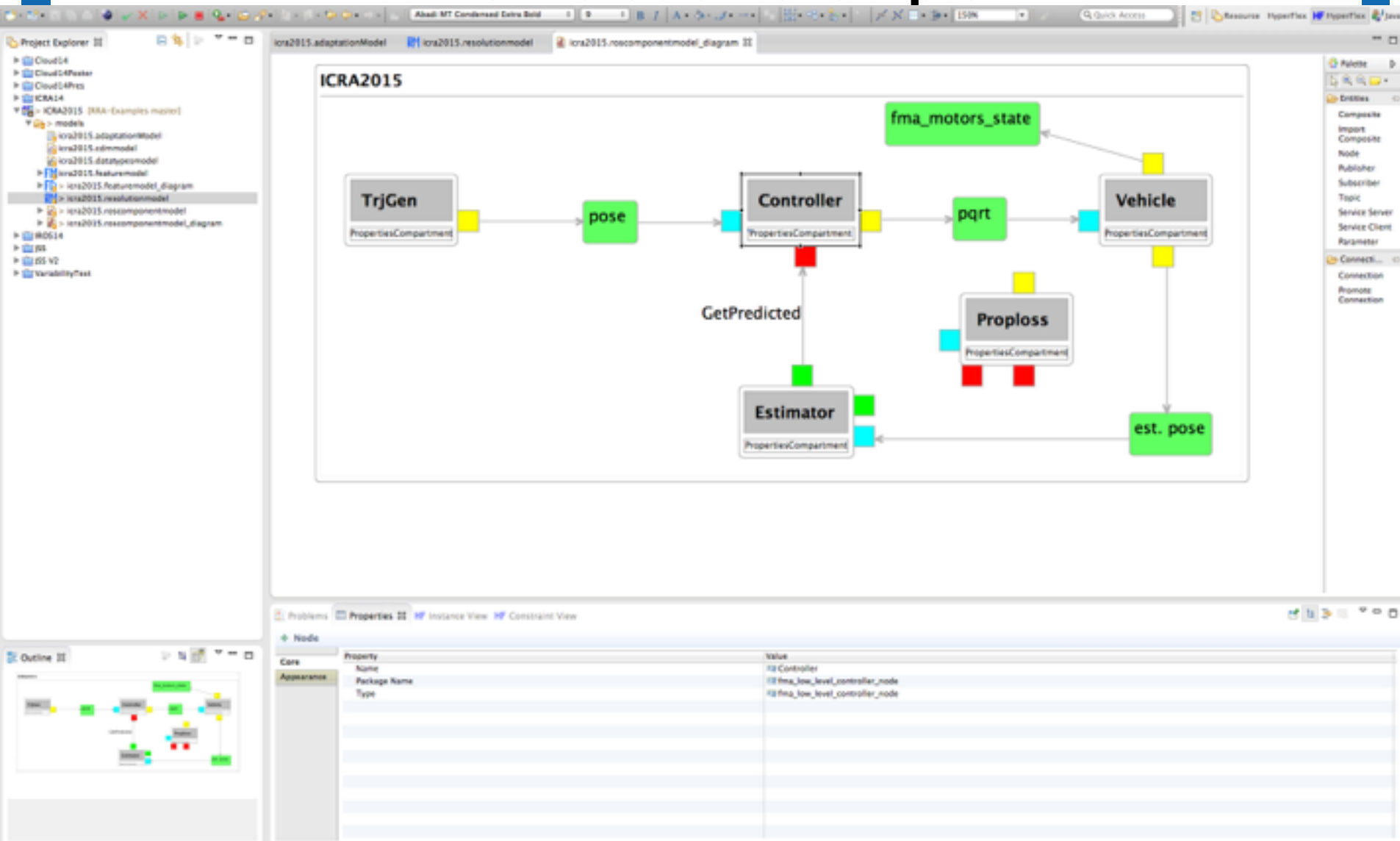


M. W. Mueller and R. D'Andrea. Stability and control of a quadro-copter despite the complete loss of one, two or three propellers.

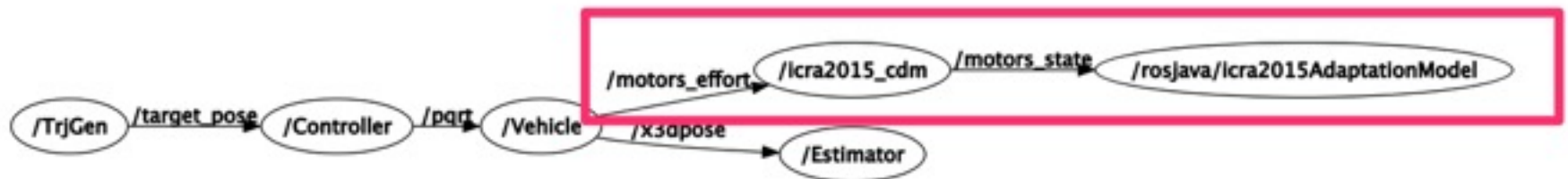
In *International Conference on Robotics and Automation (ICRA)*, 2014.



A simulation example



A simulation example



A simulation example

The screenshot displays an IDE interface with the following components:

- Project Explorer:** Shows a project structure with folders like Cloud14, Cloud14Pilot, Cloud14Phes, ICRA14, and ICRA2015. The ICRA2015 folder is expanded, showing sub-models like icra2015.adaptationModel, icra2015.cdmmodel, icra2015.datatypesmodel, icra2015.featuremodel, icra2015.featuremodel_diagram, icra2015.resolutionmodel, icra2015.resolutionmodel_diagram, icra2015.roscomponentmodel, and icra2015.roscomponentmodel_diagram.
- Source Editor:** Displays the code for `icra2015.adaptationModel`. The code includes imports for various models and a rule for `SafeRecover`.
- Properties View:** Located at the bottom, it shows a table with columns for Property and Value, currently empty.

```
import "icra2015.featuremodel";
import "icra2015.cdmmodel";
import "icra2015.datatypesmodel";
import "icra2015.resolutionmodel";
import "icra2015.roscomponentmodel";

feature model icra2015_fm;
template system model icra2015_csm;
resolution model icra2015_rm;
cdm model icra2015_cdm;
data types model icra2015_dtm;

name icra2015AdaptationModel;
period[ms] 0; // triggered by new measurements

rule SafeRecover {
  if(icra2015_cdm.motors_state > "3") deactivate_feature(icra2015_fm.ICRA2015.Proploss)
  else activate_feature(icra2015_fm.ICRA2015.Proploss)
};
```

A simulation example

The screenshot displays the Eclipse IDE interface for a ROS simulation. The **Project Explorer** on the left shows a project structure with several models under the `ICRA2015` package. The main editor shows the **Resource Set** for the `icra2015.resolutionmodel`, which includes a list of ROS service and topic connections. The **Properties** view at the bottom right shows the details for the selected object, `ROS Service Connection estimator_predict`.

Property	Value
Name	estimator_predict
Service Consumer	Node Srv Consumer estimator_predict_prosp
Service Producer	Node Srv Producer estimator_predict

Selected Object: ROS Service Connection estimator_predict

A simulation example

The screenshot shows the Eclipse IDE running a ROS simulation. The main window displays a 3D grid floor. A dialog box titled "Select the desired instance" is open, showing a list of ROS topics and services. The bottom left panel shows the "Node Graph" with a tree view of the simulation nodes. The bottom right panel shows a "PyQtGraph" plot window with a line graph.

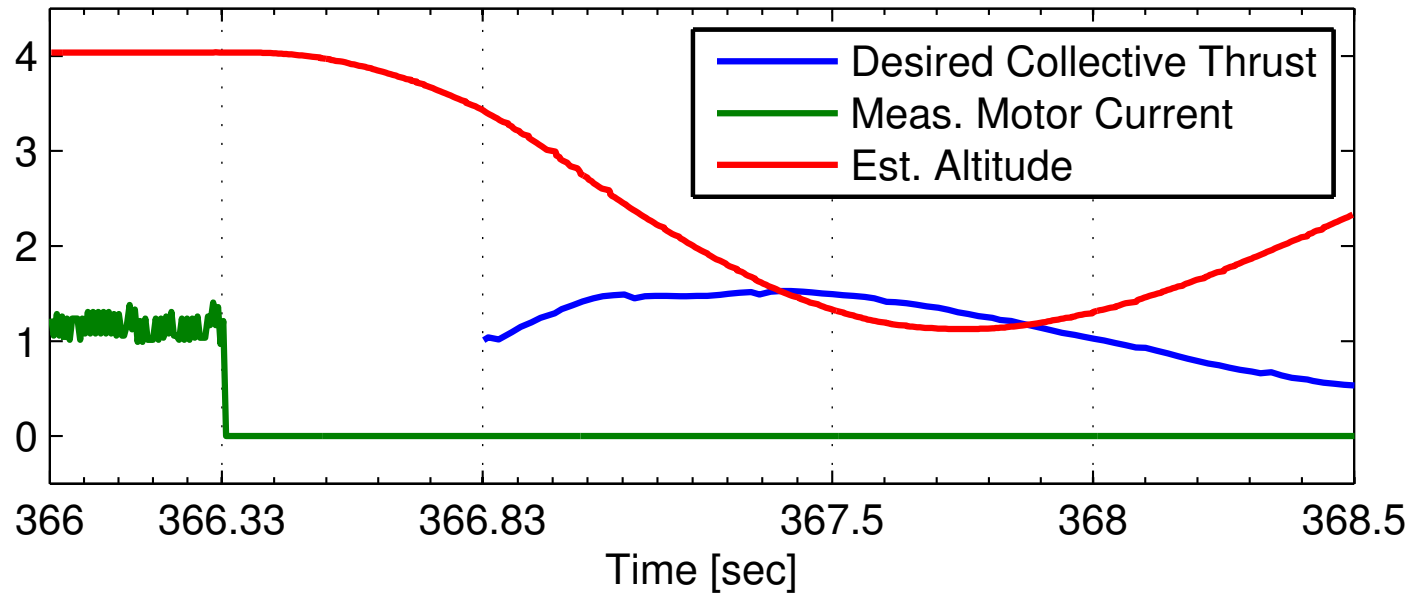
```
import "icra2015.featuremodel"
import "icra2015.cdmmodel"
import "icra2015.datatype"
import "icra2015.resolutionmodel"
import "icra2015.rescomparator"

feature model icra2015_fm;
template system model icra2015;
resolution model icra2015_fm;
cdm model icra2015_cdm;
data types model icra2015_fm;

name icra2015AdaptationModel;
period[ms] 0; // triggered

rule SafeRecover {
  if(icra2015_cdm.motors_state > "3") deactivate_feature(icra2015_fm, ICRA2015.Proploss);
  else activate_feature(icra2015_fm, ICRA2015.Proploss);
};
```

A simulation example



Thank you!
Any Questions?

