Research Activities at the *Software & Systems Engineering* Lab

Prof. Andrea D'Ambrogio

University of Roma TorVergata
Roma (Italy)

dambro@uniroma2.it

www.sel.uniroma2.it

Outline

- Software and Systems Engineering Lab
- Research contributions
  - Model-driven Approaches for:
    - Performance Engineering of Business Processes (BPs)
      - PyBPMN: a language to specify QoS properties of BPs
      - PyBPMN-driven method to predict performance and reliability properties of BPs
    - Simulation Systems Engineering
      - Bridging the gap between MDE and DS
        - The conventional approach
        - The SimArch approach
  - Model-based Interface Specification for Systems Integration

Software Engineering Laboratory

where Systems, Software and Simulations meet up!

Università di Roma Tor Vergata

www.sel.uniroma2.it

- hosted at the Department of Enterprise Engineering of the University of Rome Tor Vergata
- Research Topics
  - software and systems performance engineering
  - model-driven software and systems engineering
  - business process management
  - distributed simulation
  - software and systems quality

Projects

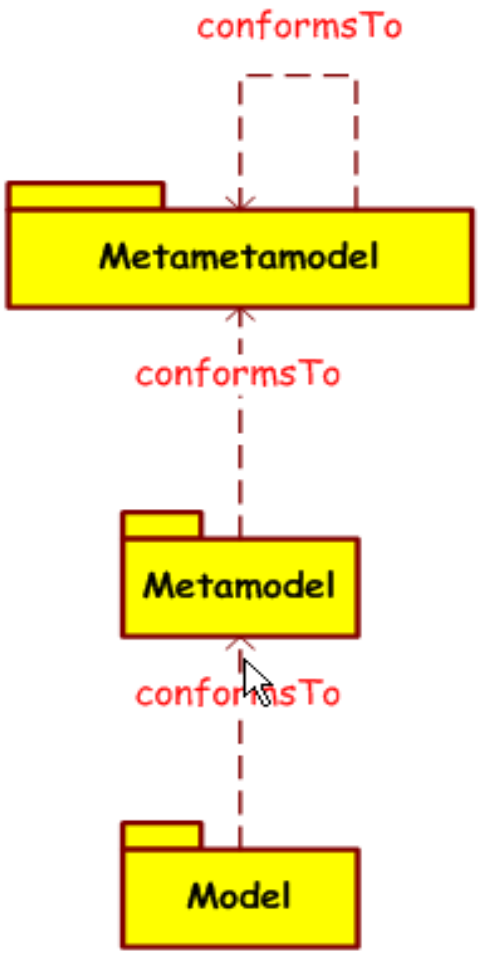
- Methods for the engineering and evaluation of system performance and dependability
  - GAAS *Generic Approach to ATM Systems* - EC DG XII
  - DAAS *Dependable Approach to ATM Systems* - EC DG XIII
  - PAMPAS *Preliminary Approach for Modelling Performance of ATM Systems* – EUROCONTROL
  - *Automated building of predictive models for performance validati*on – MIUR FIRB
  - SS&PSW *Methods for the development of dependable complex software platforms* - MAP-SELESO
- Strategies and tools for system validation
  - EVAS *EATMS Validation Strategy* – EUROCONTROL
  - VALERY *Study for the Development of a Validation Data Repository* – EUROCONTROL
  - EPVDR *Enhanced Prototype Validation Data Repository* - EUROCONTROL
- Methods and tools for model-driven systems engineering
  - OATA *Overall ATM Target Architecture* – EUROCONTROL
  - *SysML-based Model-driven System Development* – Elettronica SpA
- Software projects cost estimation and verification
  - *Software Acquisition Assessment* - ENAV
- E-government information systems
  - *INAIL Information System Quality Assessment* - INAIL
  - *Adequacy Assessment of Computing Facilities and Network Services* – ICE
  - *Requirements Engineering for Public Lighting Energy Efficiency* – ISIMM-ENEA
- Distributed and web-based simulation
  - *Integration of HLA and Web Services for web-based and distributed simulation* – MIUR FIRB
  - *HRAF: EDLS Distributed Simulation Federation and Model-driven Engineering Framework Development* – ESA-GMV
  - *MASTER: Modeling and Simulation as a Service for Training and Experimentation* – Italian MoD National Plan for Military Research

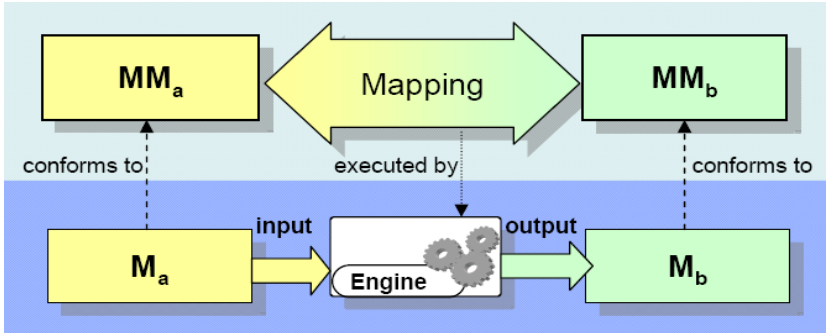
Other projects/activities

- FP7 DAEMONS (DEcentralized, cooperative, and privacy-preserving MONitoring for trustworthiness)
  - *publish&subscribe* approaches for implementing the coordination middleware
- ESA (European Space Agency) Summer of Code in Space 2013
  - ICML (Interface Communication Modeling Language)
- ProSys (POR FESR Lazio)
  - Adaptive Business Process Management System
- ALADDIN (Autonomous Learning Agents for Decentralised Data and Information Networks)
  - Agent-based M&S [software: SimJADE, DisSimJADE]
- euHeart (in Virtual Physiological Human)
  - Model Databasing [software: AMDB]
- Galileo
  - Architectural Modelling
- Space Situational Awareness
  - Data Policy modelling, definition and verification
- GMES-PURE
  - GMES Partnership for User Requirements Evolution
- Jason-CS / EPS-SG
  - Requirements Management

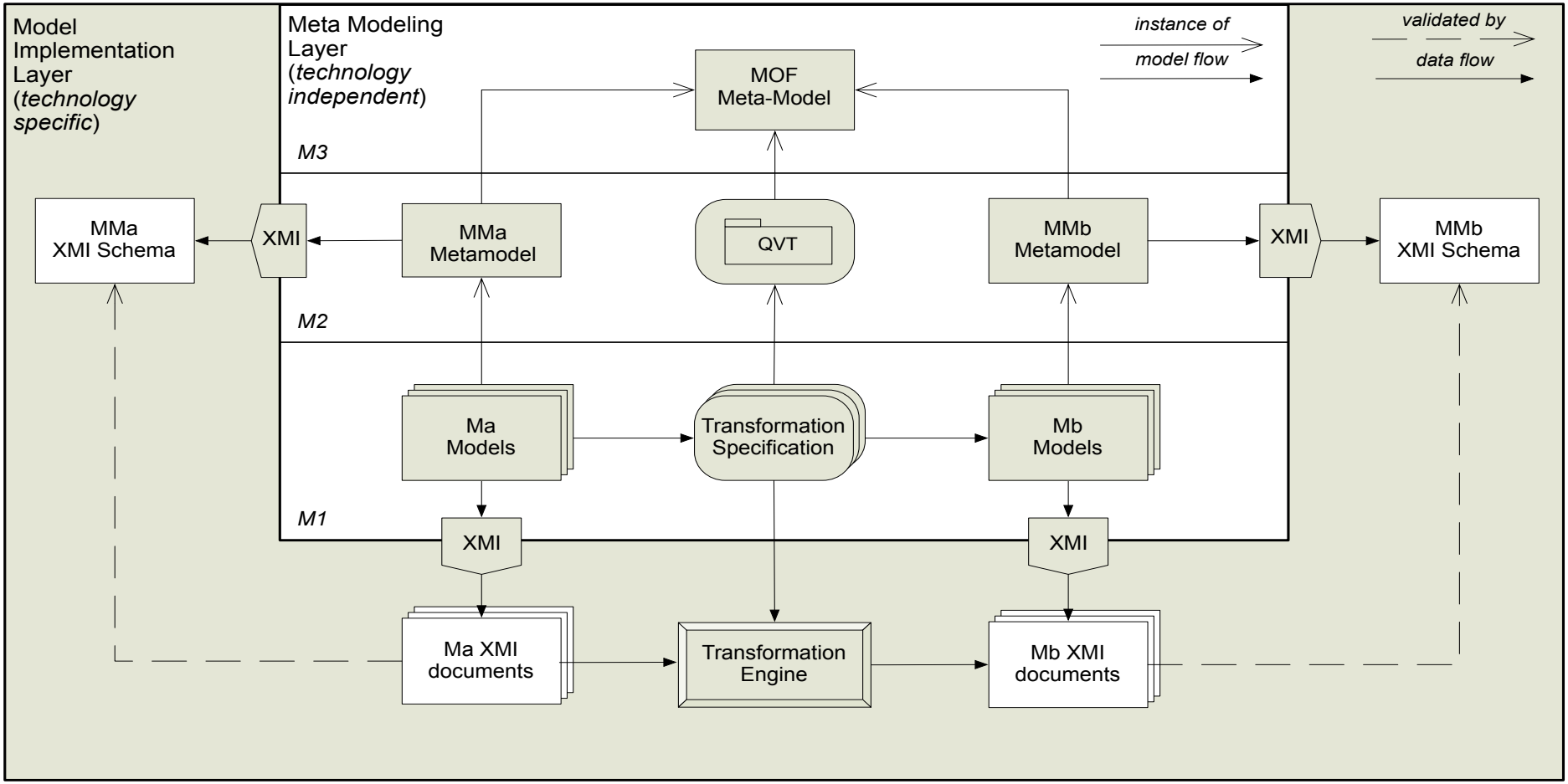
Model-driven Engineering (MDE)

- Enabler of automation
- Key elements
  - a language to specify metamodels (i.e., a metametamodel)
  - a language to specify model transformations
- Incarnations
  - MDA, MIC, Software Factories





MDA in a nutshell



Model-driven Approaches for *Performance Engineering* (in the BPM domain)

Business processes

- The term Business Process (BP) refers to the set of activities that companies and organizations carry out to provide services or produce goods

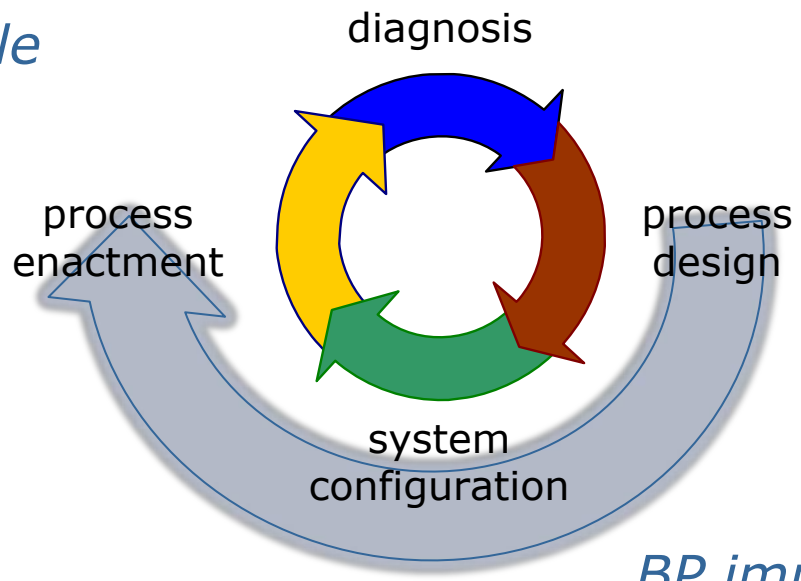
- A BP can be seen as a an orchestration of tasks, each one related to the automated or human resources in charge of its execution

➡ Our contribution focuses on fully automated BPs executed as orchestrations of software services

Business Process Management (BPM)

- The set of methods, techniques and software to design, enact, control and analyze operational processes involving humans, organizations, applications, documents and other sources of information *[van der Aalst et. al., 2003]*

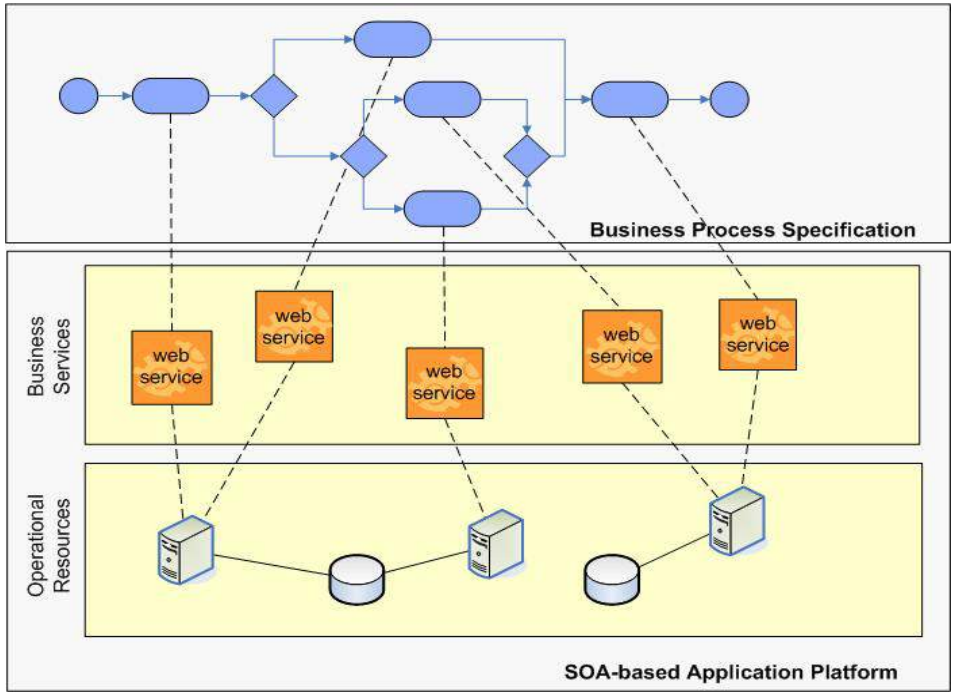
BP lifecycle



BP implementation cycle

Business processes and SOAs

- From an IT perspective, BPM is related to BP automation
  - SOA standards define a framework that allows the composition of atomic services to define and execute higher level business processes
  - Web services represent a set of technologies needed to define and invoke remote software services



The automated execution of tasks within a BP can be based on SOA standards

Our contribution

Objective

- Definition of an approach to *describe* and *analyze* the QoS of BPs by:
  - exploiting model-driven approaches
  - encompassing each stage of the BP implementation cycle, from the abstract design down to the execution

Contributions

1. ***Description perspective***: PyBPMN, a language to specify the *QoS properties* of BPs
2. ***Analysis perspective***: A PyBPMN-driven method to predict, at design time, performance and reliability properties of BPs

QoS properties: performance and reliability

Reliability → The ability of a process to perform correctly its required tasks in a given time interval

Performance

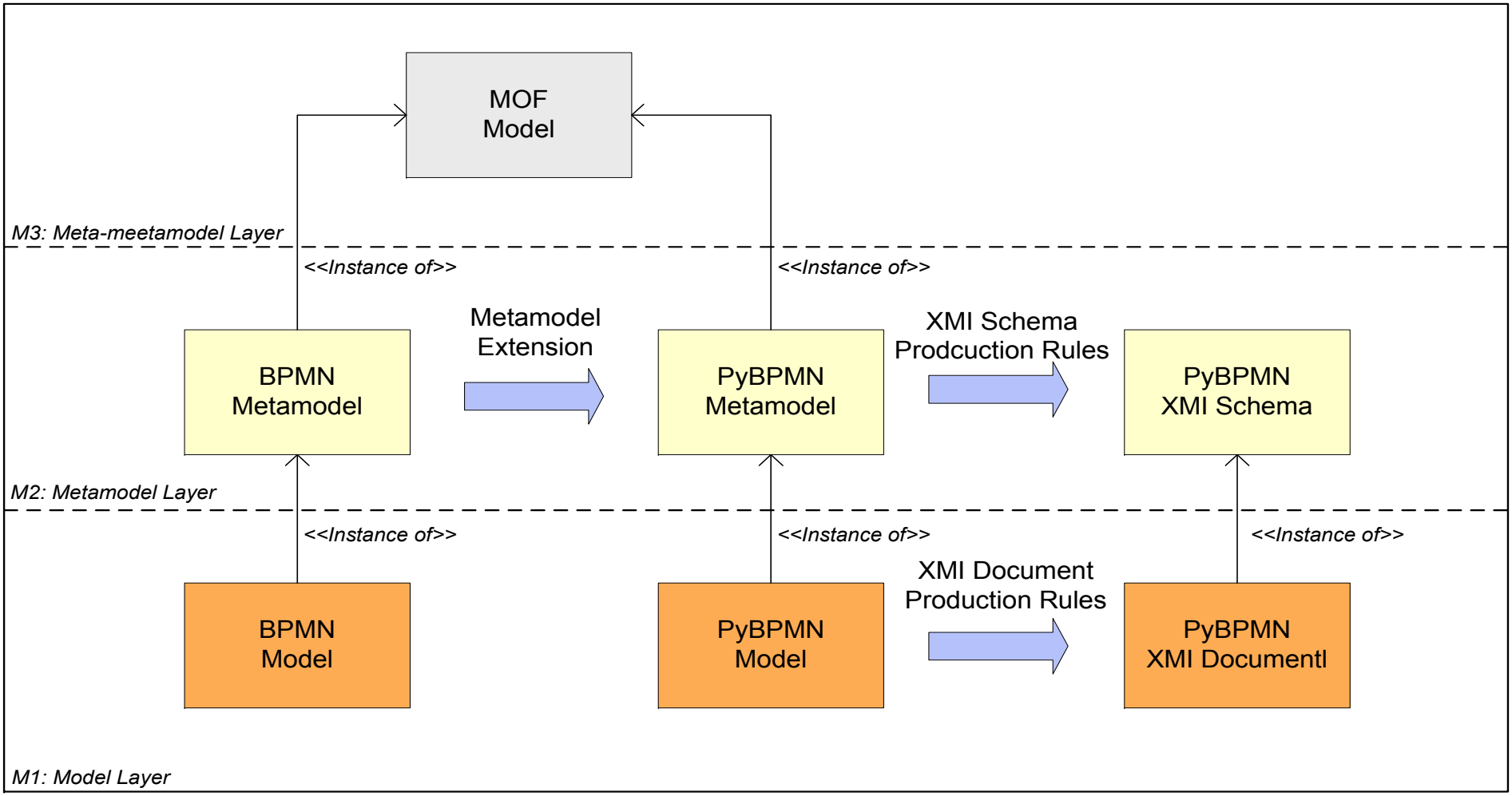
BP domain:
Key Performance Indicators (KPIs): a set of measures that focus on critical aspects of organizational performance

IT domain:
time-related properties such as throughput, response times and resource utilization

Modeling QoS properties of a BP: ***PyBPMN***

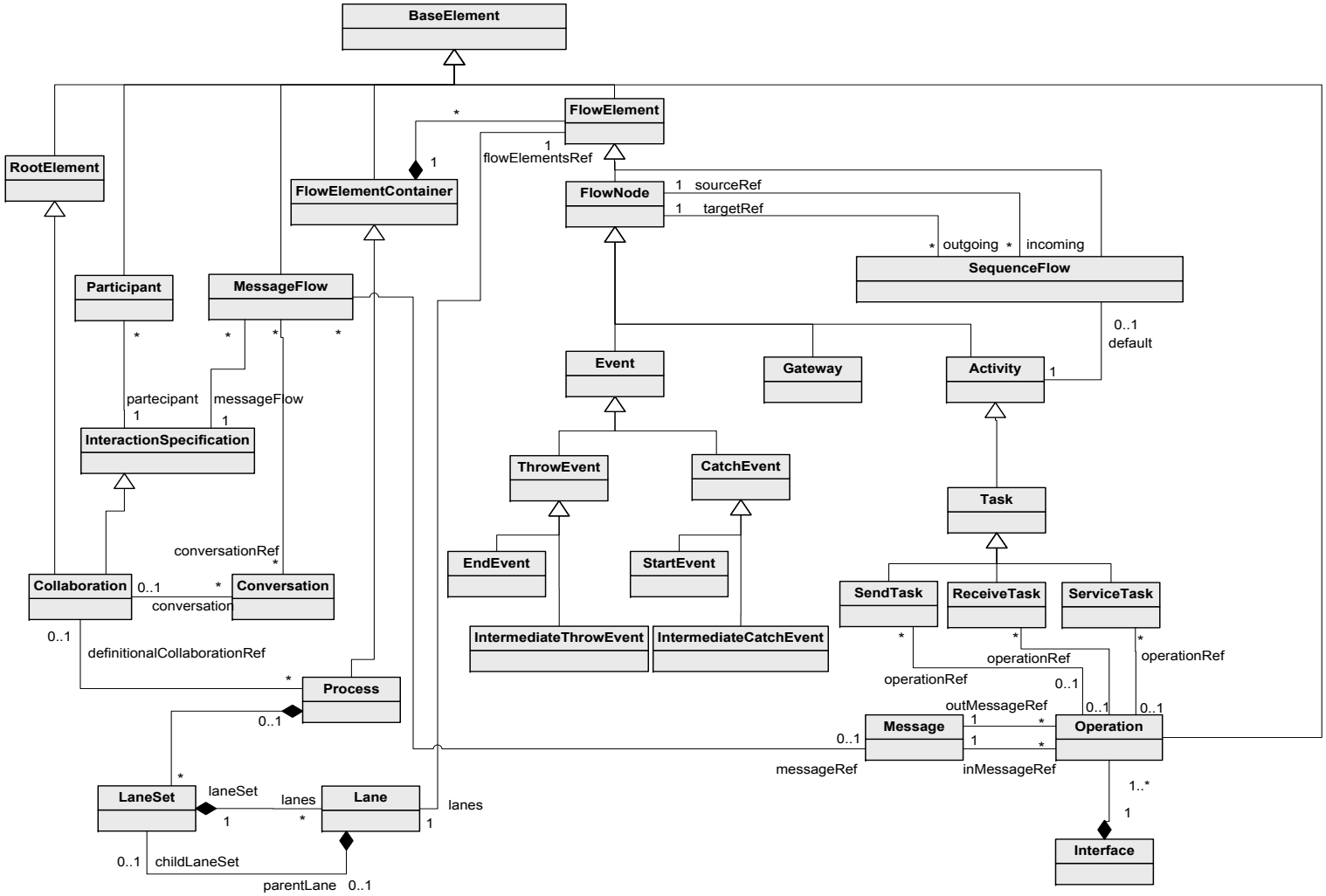
- We introduce Performability-enabled Business Process Modeling Notation (PyBPMN), a language to specify QoS properties of BPs
- PyBPMN has been designed as an extension of the Business Process Modeling Notation (BPMN), the standard language for business process modeling promoted by OMG
- According to MDA the extension process:
  - leverages on MOF (Meta Object Facility) and XMI (XML Metadata Interchange)
  - is based on a metamodel extension
- The extension specifically addresses:
  - Performance modeling: UML Profile for Modeling and Analysis of Real-Time Embedded systems (MARTE)
  - Reliability modeling: research contributions that add the description of reliability properties to MARTE *[Petriu, Bernardi and Merseguer, 2008]*

BPMN extension process



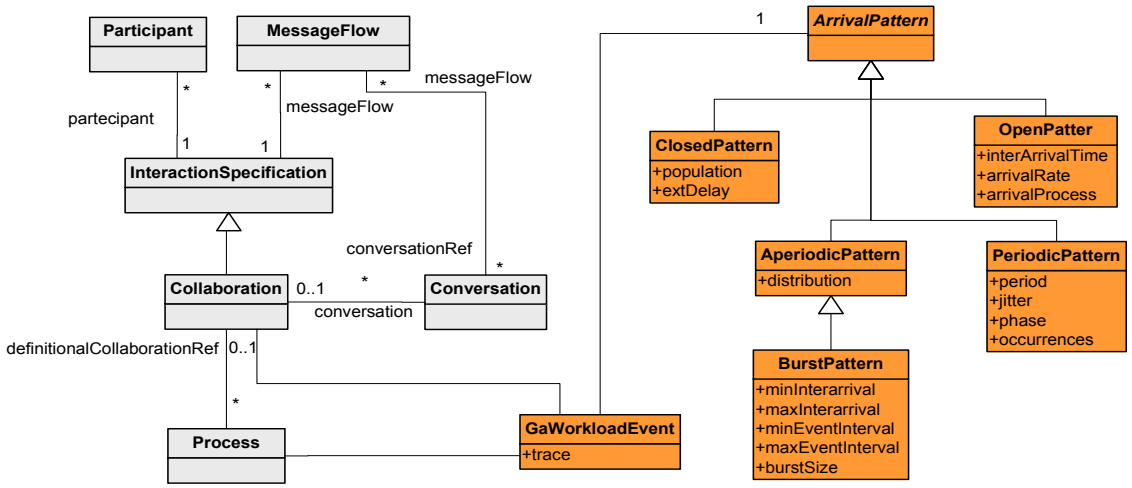
BPMN: Business Process Model and Notation

BPMN is a standard for the high-level specification of business processes

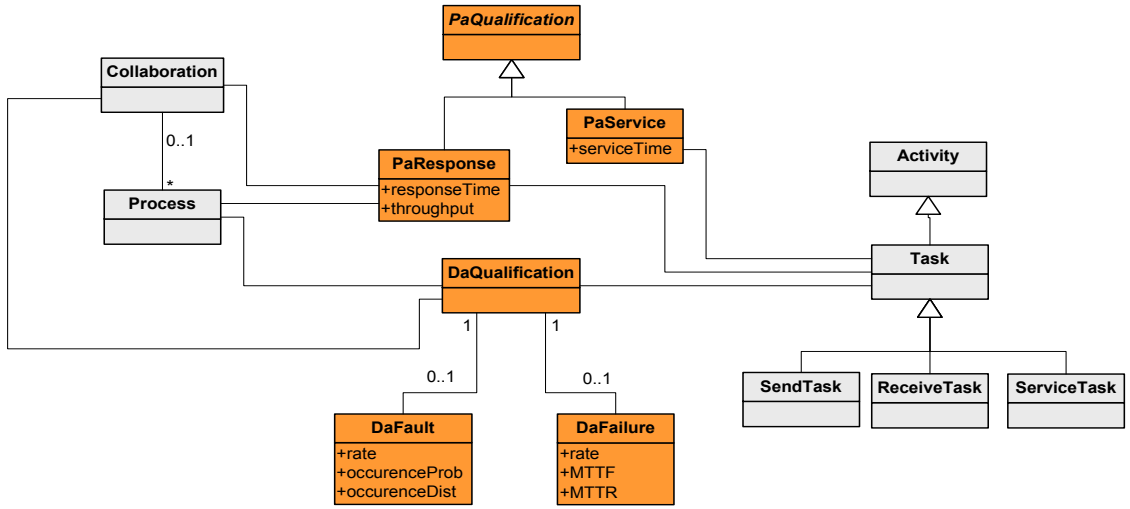


PyBPMN extension details

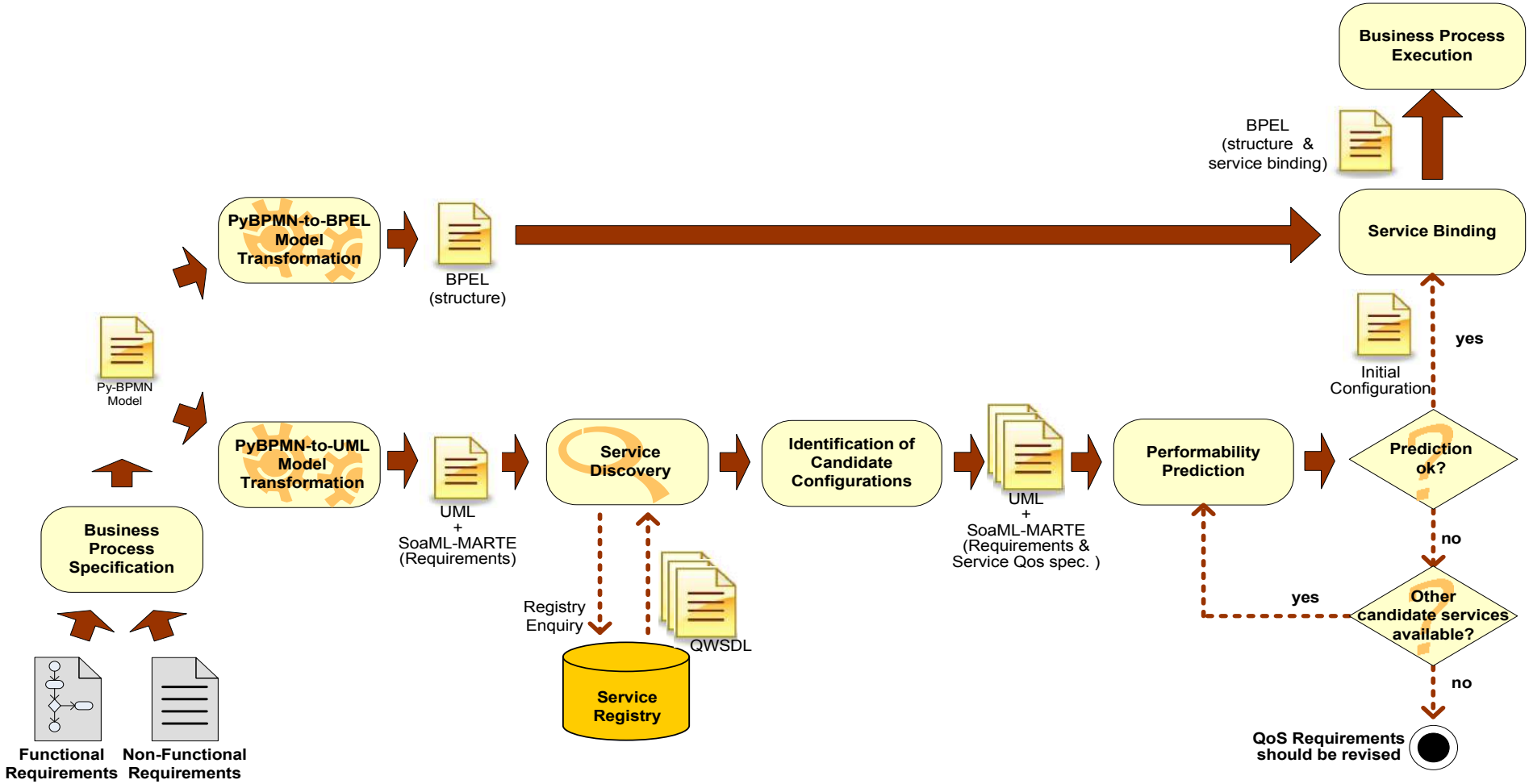
Workload characterization



Performance/reliability characterization



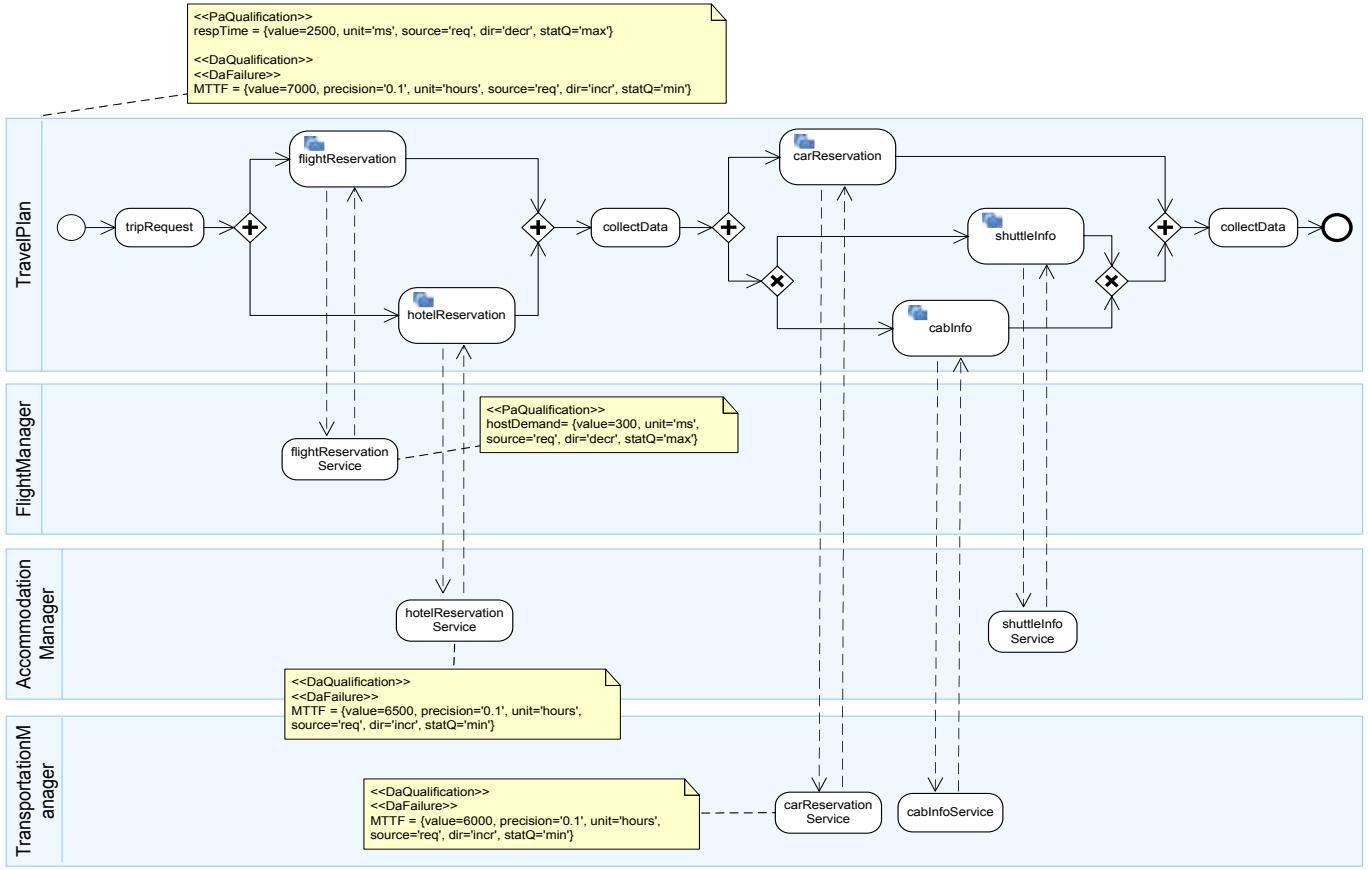
Model-driven method to predict QoS properties of BPs



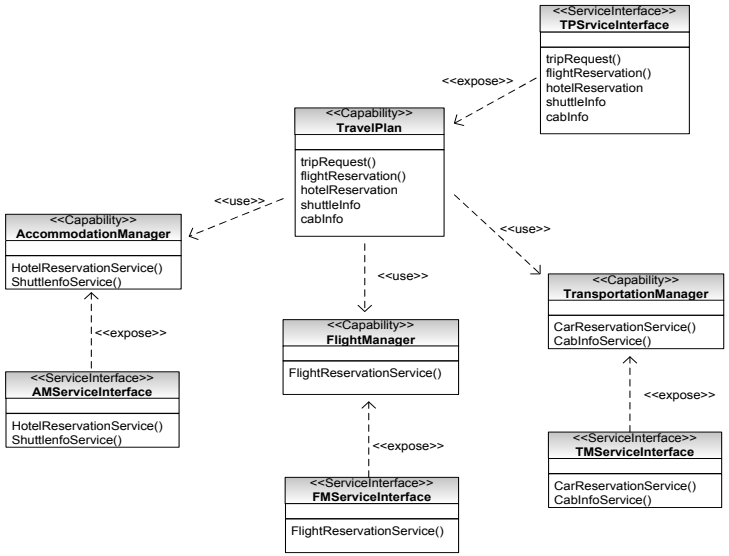
BP modeling by use of PyBPMN *Example*

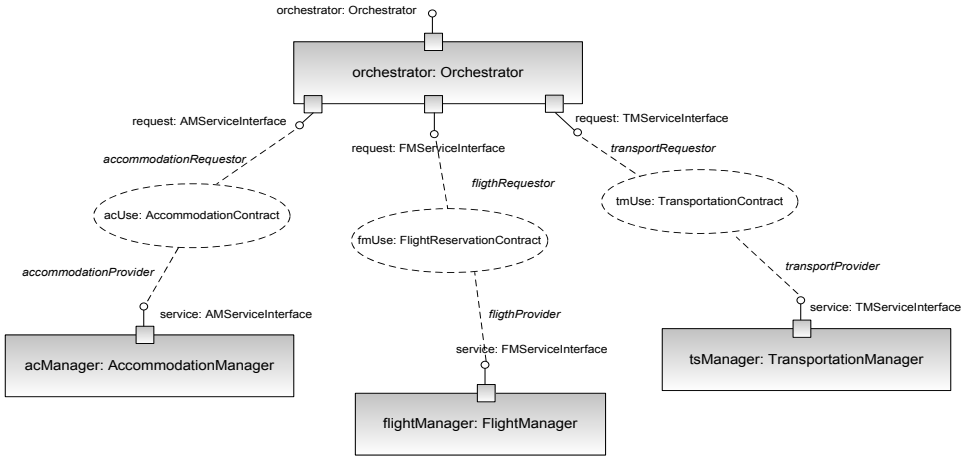
Let us consider a business process that provides a service for creating travel plans. The process makes use of the following services:

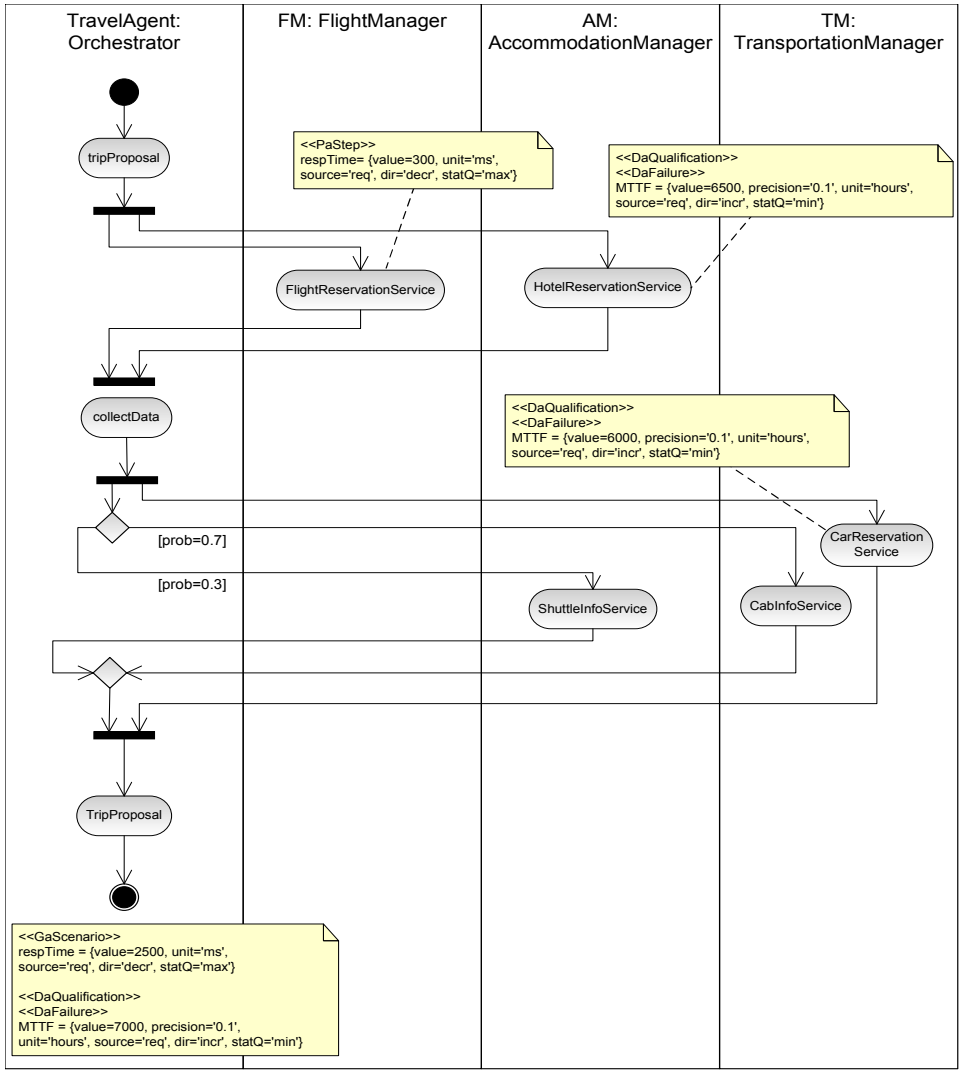
- Flight Manager (FM) service
- Accommodation Manager (AM) service
- Transportation Manager (TM) service



PyBPMN-to-UML model transformation
Example







Service discovery
Example

Service discovery retrieves the QoS-enabled descriptions of candidate services

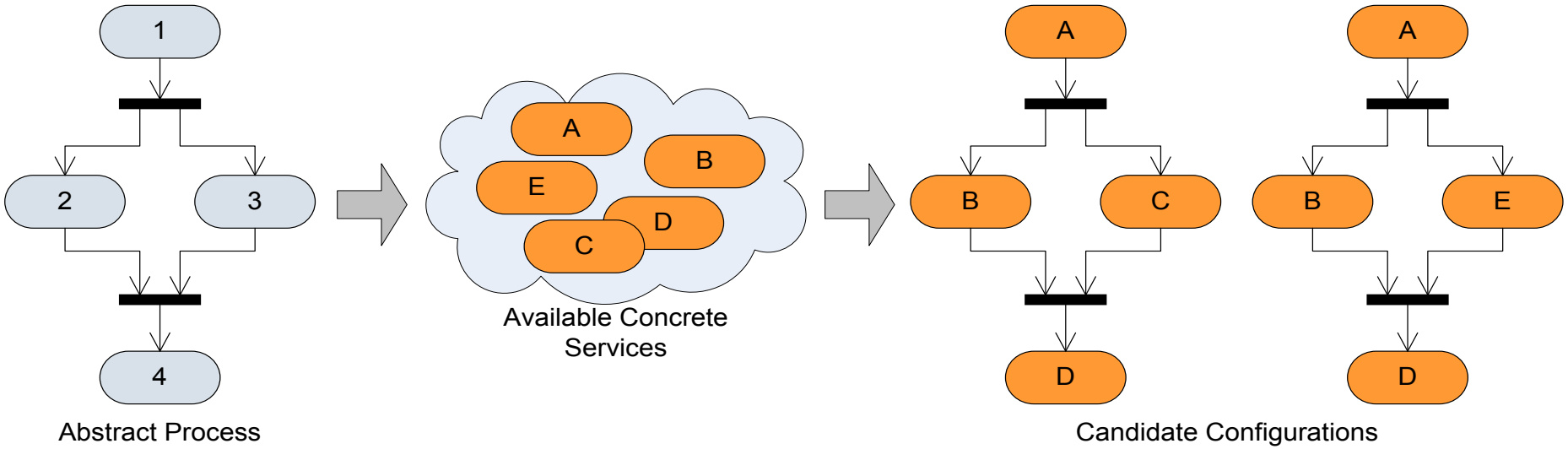
Parameter		TM_A	TM_B
Performance	*CarReservation time demand*	120 ms	90 ms
	CabInfo time demand	115 ms	84 ms
	Network bit rate	10 Mb/s	100 Mb/s
Reliability	*MTTF*	7900 hours	5100 hours

- TM_A provides better reliability properties
- TM_B provides better performance properties

Problem: no win-win, which one is to be selected?

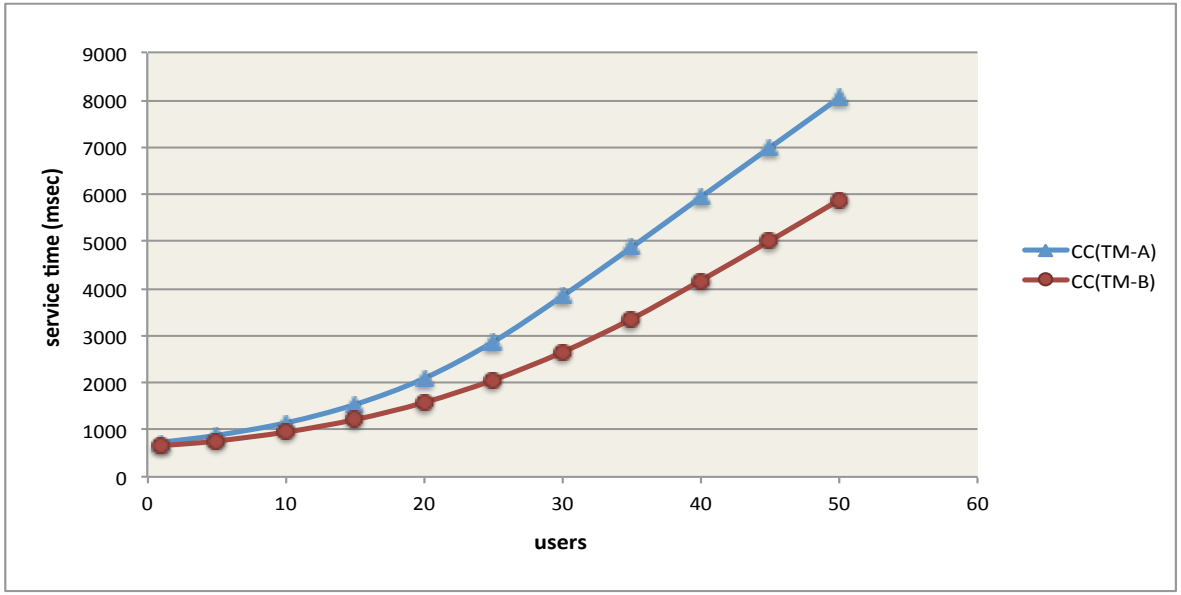
Identification of candidate configurations

- In general, the service discovery step gives as output more than a single concrete service for each abstract service
- Each possible binding leads to a *candidate configuration (CC)*
- Problem: how to select the *initial configuration* (IC) among the available CCs?



The IC is selected by use of a **performability prediction algorithm**

Performance and Reliability predictions
Example



Performance Prediction

choose **CC(TM$_B$)** as IC

BP Implementation	MTTF
Candidate Configuration with TM$_A$	2163 hours
Candidate Configuration with TM$_B$	1918 hours

Reliability Prediction

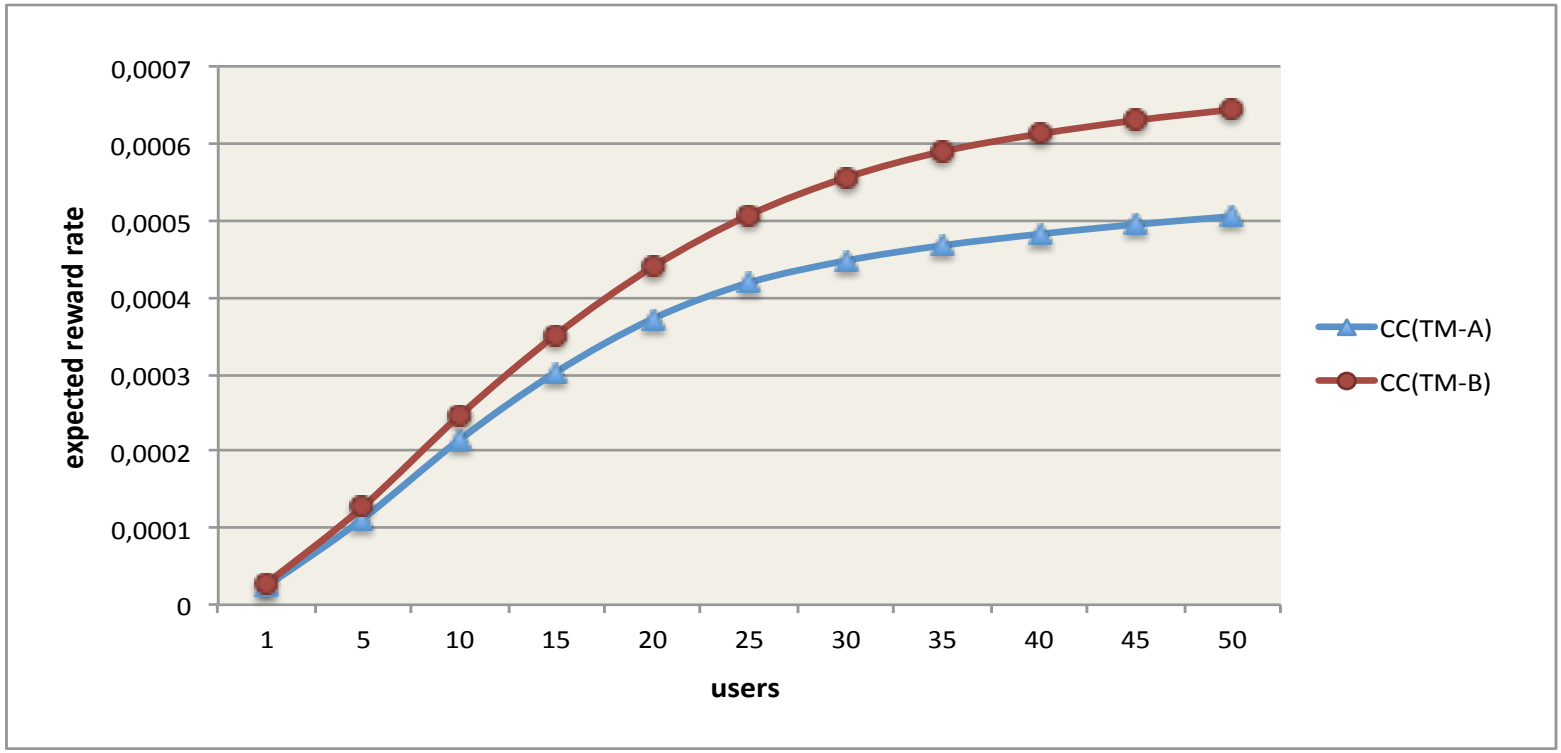
choose **CC(TM$_A$)** as IC

Performability prediction

- Performability is the joint analysis of performance and reliability
- Performance and reliability predictions are used to obtain the performability prediction, as follows:
- For each candidate configuration CC_k (k=1..n):
  1. CC_k is assumed to be the initial configuration (IC)
  2. The reliability prediction is used to evaluate $P(CC_i)$, the probability to be in CC_i starting from the assumed IC
  3. The performance prediction is used to obtain $T(CC_i)$ and assign it as a reward to CC_i
  4. The performability prediction is obtained in terms of the expected reward rate of IC, given by:

$$RW(IC) = \sum_{i=1}^{n} P(CC_i)\, T(CC_i)$$

Step 5.3: Performability prediction - Example



Performability Prediction → choose $CC(TM_B)$ as IC

Model-driven Approaches for *Simulation Systems Engineering*

Simulation for Systems Engineering

- The validation of complex systems from the early development phases (*lifecycle validation*) can be effort- and time-consuming
- *Modeling & Simulation (M&S)* is widely recognized as an effective and powerful tool for lifecycle validation of systems, but:
  - M&S methods must scale with growth and evolution of complex systems and ecosystems (e.g., *SoS* or *ULS*)

Our contribution

- How to enable M&S methods that take into account the peculiar complexity/scalability/evolvability of complex systems?

The proposed solution exploits model-driven approaches for the effortless development of complex distributed simulation systems

Useful definitions

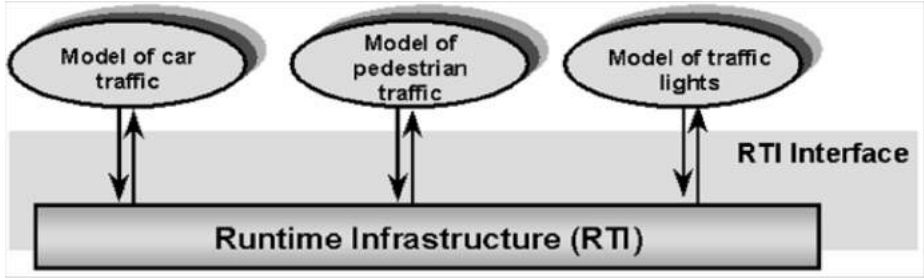
- System Under Study (SUS)
  - the system that has to be simulated to get insights into or to predict its behaviour
  - typically specified at development time by use of *system models*
- Simulation Engineering:
  - the set of activities to be carried out first to build a *simulation model* of the SUS and then to implement it into a *simulation system*, i.e., a software system that "executes" the model onto a given centralized or distributed platform.
- Local Simulation (LS) System
  - A simulation system deployed onto and executed by a single host
- Distributed Simulation (DS) system
  - A simulation system that consists of a set of sub-systems deployed onto and executed by a set of geographically distributed hosts

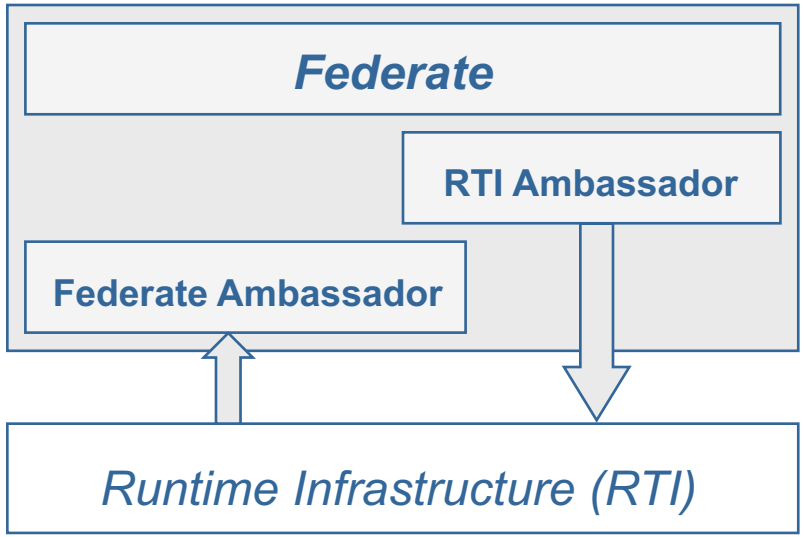
Distributed Simulation (DS)

- The term *distributed* is interpreted in the sense of traditional distributed computing (e.g., based on the C/S paradigm)
- *Goal*
  - synchronize and coordinate remote simulation programs
- *Benefits*
  - Geographical distribution
  - Integrating simulators from different manufacturers
  - Reusability
  - Load balancing
  - Fault tolerance

High Level Architecture (HLA)

- *IEEE standard 1516*
- Main elements
  - *Federate*: a remotely-accessible simulation sub-system
  - *Federation*: the overall DS system, composed of a set of Federates
  - *RTI*: provides communication and coordination services to the Federates that join into a Federation





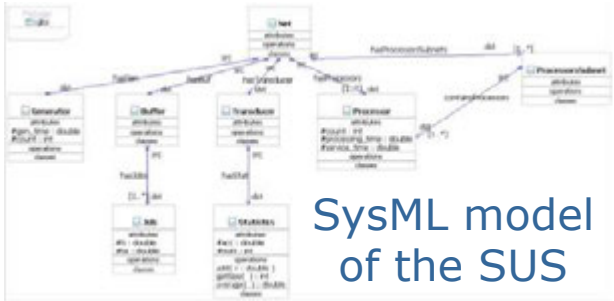
MDE and DS
Opportunities

- MDE (Model Driven Engineering) is finding increasing acceptance in the development of complex systems:
  - enabler of reuse
  - high degree of automation
- DS systems are inherently complex:
  - intrinsic concurrency
  - required interoperability
  - intricacies of currently available DS platforms
  - the *Green Elephant* risk
- MDE provides a promising approach for supporting the development of DS systems of higher quality at largely reduced time, effort and cost

MDE and DS
Challenges

- On the DS side:
  - code-centric approaches
  - development process:
    - not standardized (only FEDEP/DSEEP recommendations)
    - often not starting from scratch
    - often requiring the integration of legacy subsystems
  - interoperability is only dealt with at syntactic level
  - support for *simulation-in-the-loop* is limited
- On the MDE side:
  - model-centric approach
  - tool support for defining and orchestrating model transformations is still very limited
  - modeling languages strongly influenced by UML

MDE for DS system development
from *cogitative* to *generative* approaches



SysML model of the SUS

where

$DEVS_{subs} = (X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta)$

$X = \{\}$

$Y = \{1\}$

$S = \{\text{"sixty"}, \text{"forty"}, \text{"five"}, \text{"surface1"}, \text{"surface2"}\} \times R_{0,\infty}^{+}$

$\delta_{int}(\text{"sixty"}, \sigma) = (\text{"surface1"}, 60)$

$\delta_{int}(\text{"siurface1"}, \sigma) = (\text{"forty"}, 25)$

$\delta_{int}(\text{"forty"}, \sigma) = (\text{"five"}, 5)$

$\delta_{int}(\text{"five"}, \sigma) = (\text{"surface2"}, \infty)$

$\lambda(phase, \sigma) = 1$

$ta(phase, \sigma) = \sigma$

////////////////////////////////////

The phone call modification is

$\delta_{ext}(phase, \sigma, e, x) = (\text{"five"}, 5)$ if phase != "surface1", "surfac2", or "5"

$= (phase, \sigma - e)$ *otherwise*

$\delta_{con}(phase, \sigma, e, x) = \delta_{ext}(phase, \sigma, e, x)$ //pay attention to external event (call)

Simulation Model

```
package gbp;

import genDevs.modeling.digraph;
import genDevs.simulation.coordinator;

//stub della classe tipata come CoupledCoordinator che viene
utilizzata
//per avviare il simulatore in modalità centralizzata
//N.B. tale classe non viene utilizzata in modalità distribuita
(DEVSSOA)
public class Net extends digraph {

        private static final long serialVersionUID = 1L;

        //Root Coordinator
        protected coordinator coordinator;

        public Generator hasGen;

        public Buffer hasBuf;

        public Processor hasProcessore[];

        public Transducer hasTransducer;

        public Net() {
                super("Net");
                initialize();
        }
        public static void main(String[] args) {
                //se il simulatore si utilizza in modalità
centralizzata
                //è necessario instanziare l'oggetto coordinator
        }
}
```

DS system code

Bridging the gap between MDE and DS

- Two approaches
  1. The **conventional** one
     - applies a conventional MDA process to the development of DS systems
     - based on top-down refinement
     - the platform is the DS standard (+ its implementation)
  2. The **simulation-enabled** (or **`SimArch`**) one
     - introduces the **`SimArch`** technology to facilitate the model-driven development of DS systems
     - based on bottom-up abstraction
     - the platform is the domain-specific language of the simulation model

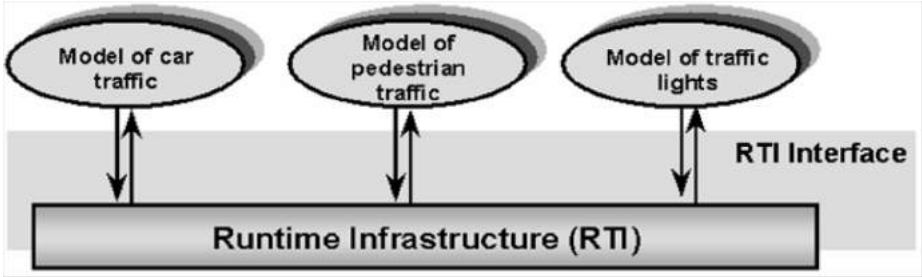
1. The conventional approach

- It is based on the standard MDA process
- Obtains the benefits of MDE approaches
- Requires:
  - The appropriate marking of the system PIM (by use of the Model-View-Controller pattern)
  - The choice of a specific DS infrastructure (e.g., HLA)
  - The introduction of an UML extension (Profile) for annotating UML models with DS infrastructure details
  - The specification of a PIM (SUS model) to PSM (simulation model) *model-to-model* transformation
  - The choice of a given DS implementation
  - The specification of a PSM to code *model-to-text* transformation

HLACloud Framework

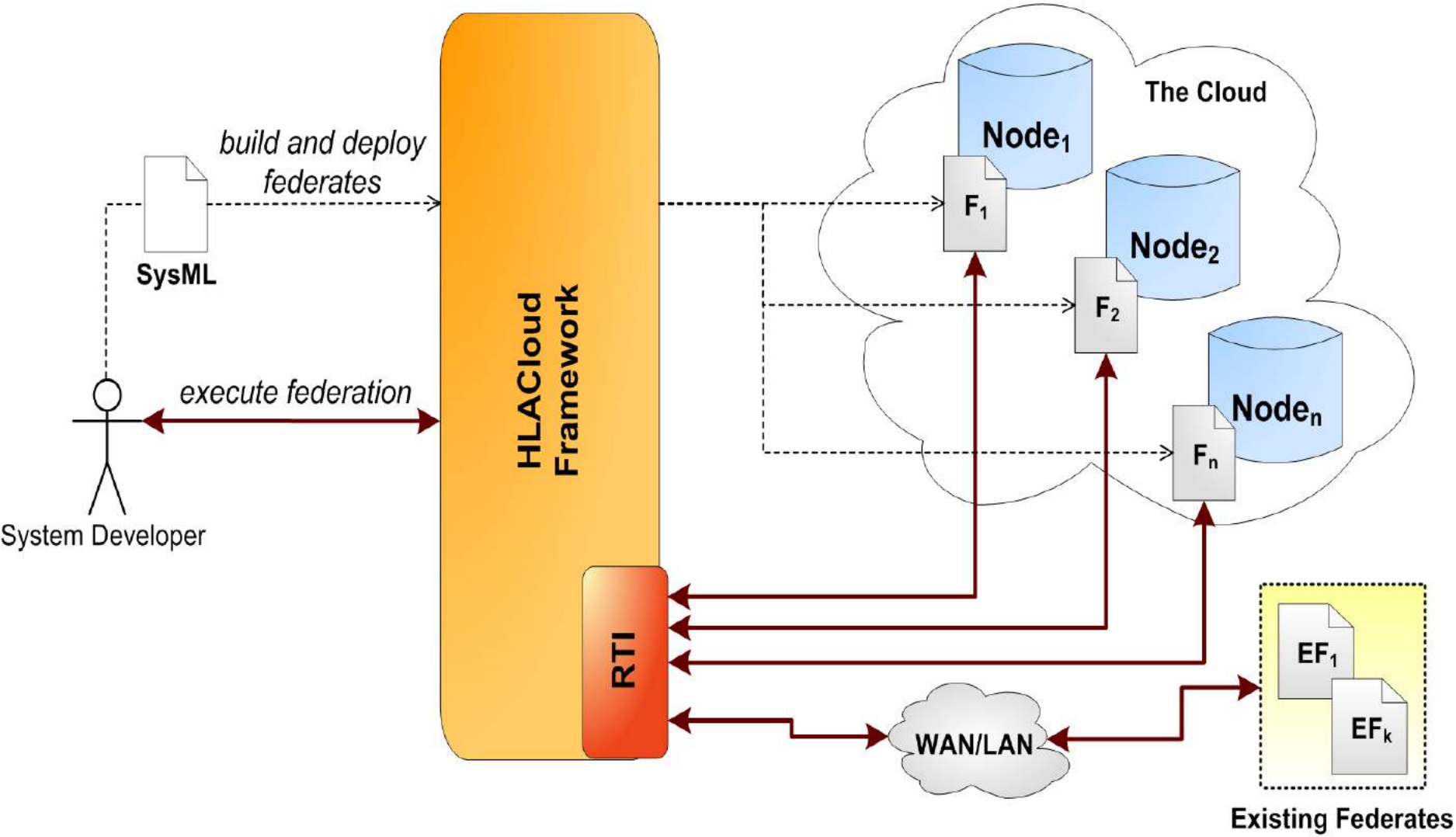
- System model
  - specified in *SysML*

- DS
  - implemented in *HLA*



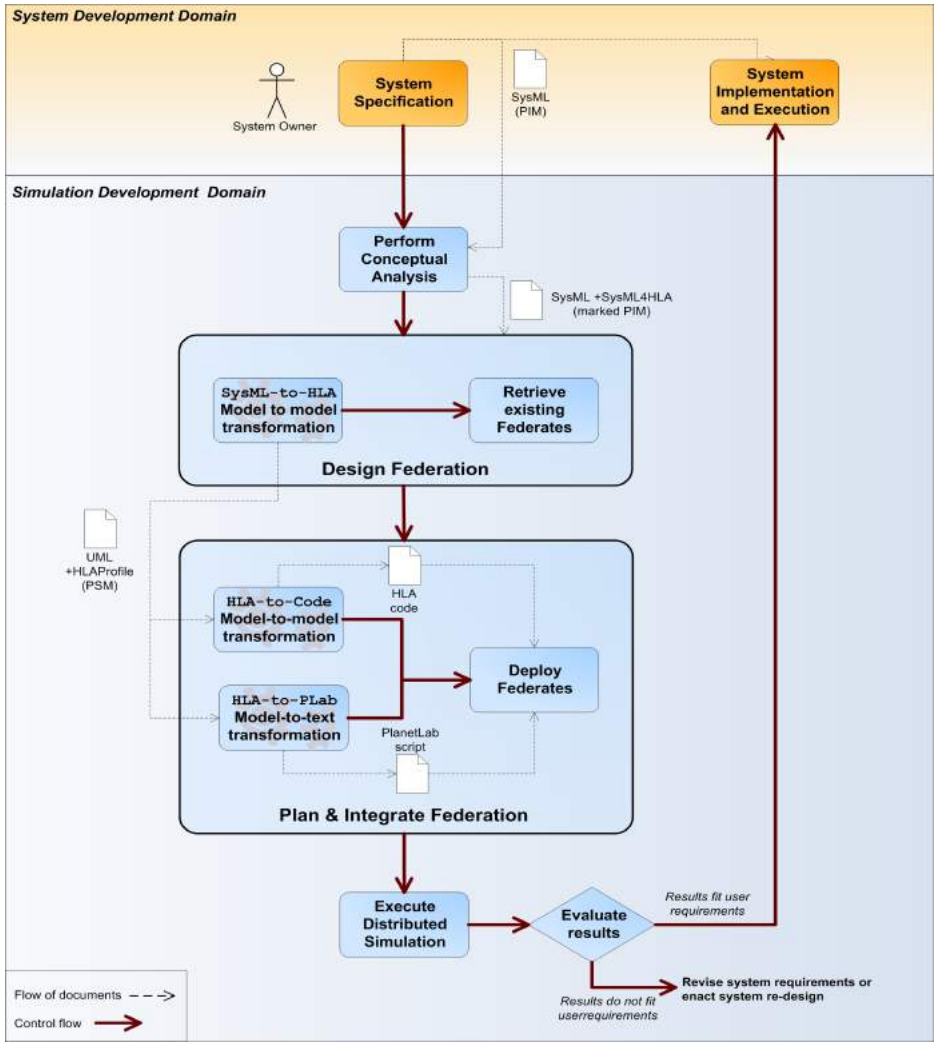
- DS execution
  - carried out on *PlanetLab*

Rationale

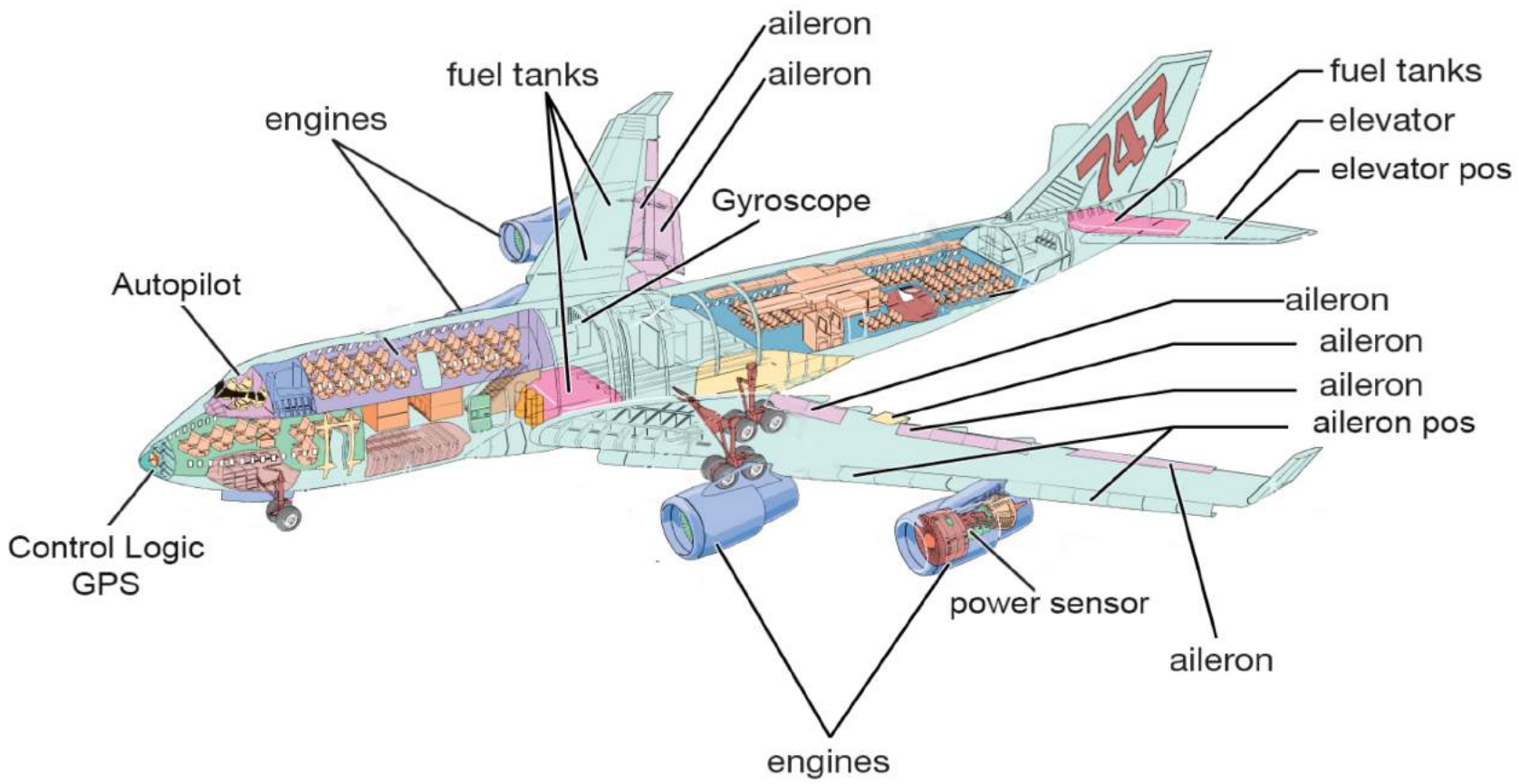

build and deploy
federates
SysML
execute federation
System Developer
HLACloud
Framework
RTI
The Cloud
Node1
Node2
Noden
F1
F2
Fn
WAN/LAN
EF1
EFk
Existing Federates

Model-driven Process

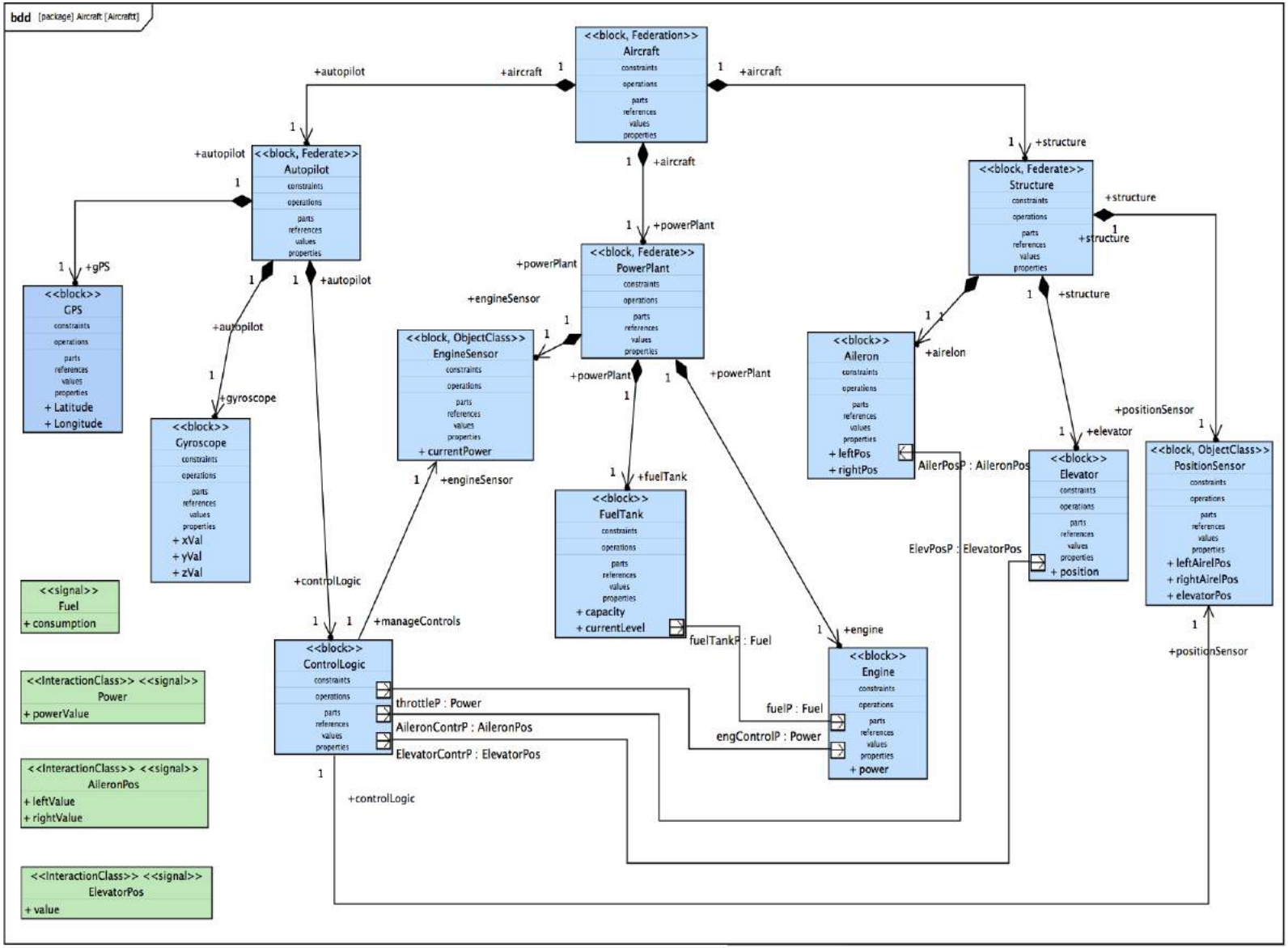
- based on DSEEP
  - IEEE Recommended Practice for Distributed Simulation Engineering and Execution Process
- M2M transformation
  - `SysML-to-HLA`
    - from *SysML* to *HLA-based UML*
- M2T transformations
  - `HLA-to-Code`
    - *HLA-based UML* to *HLA code*
  - `HLA-to-Plab`
    - from *HLA-based UML* to *PLab* configuration
- Modeling Extensions
  - *SysML4HLA* Profile
    - for *SysML* annotation
  - *HLA* Profile
    - *for HLA-based UML* annotation



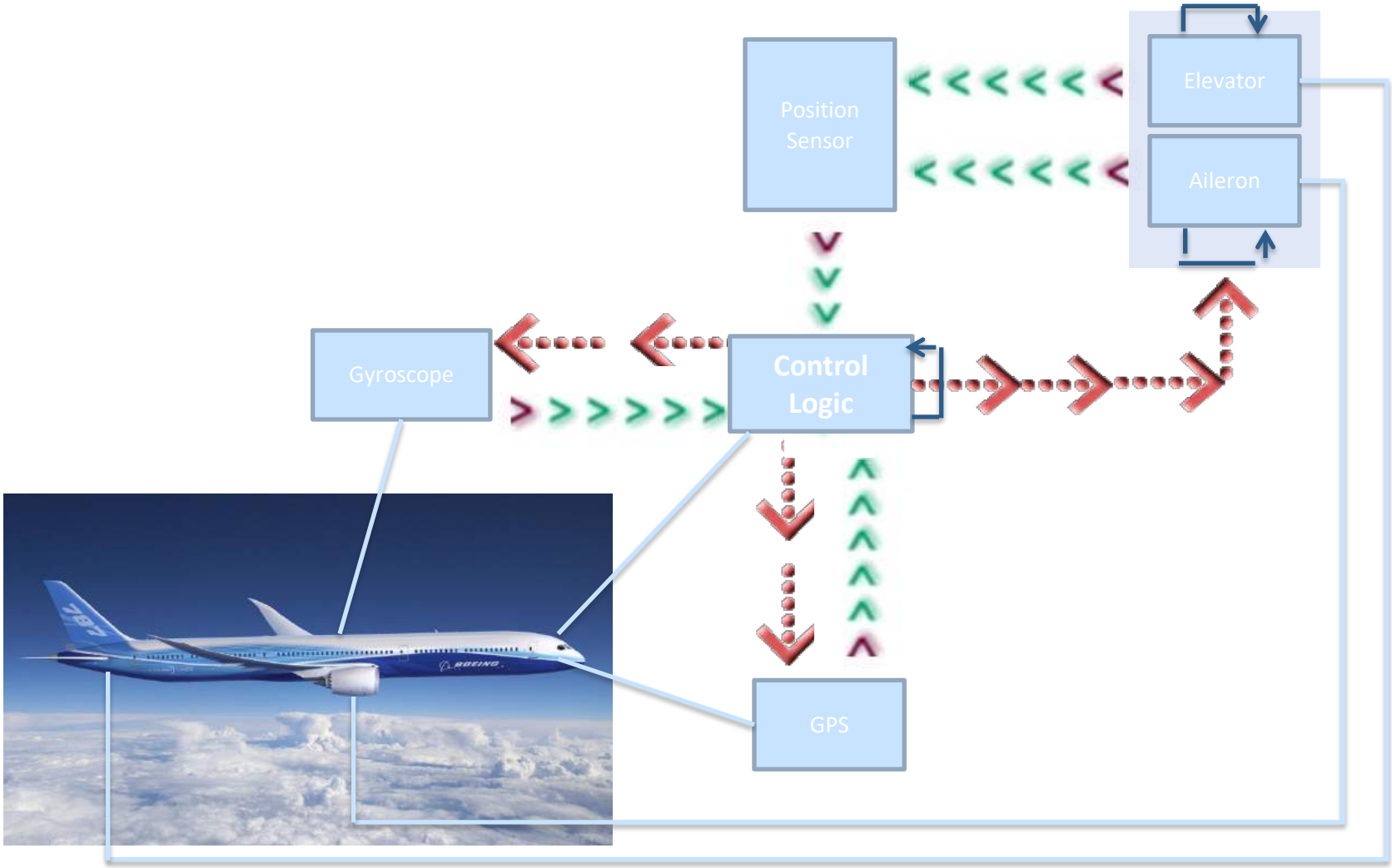
Example Application (*structure*)



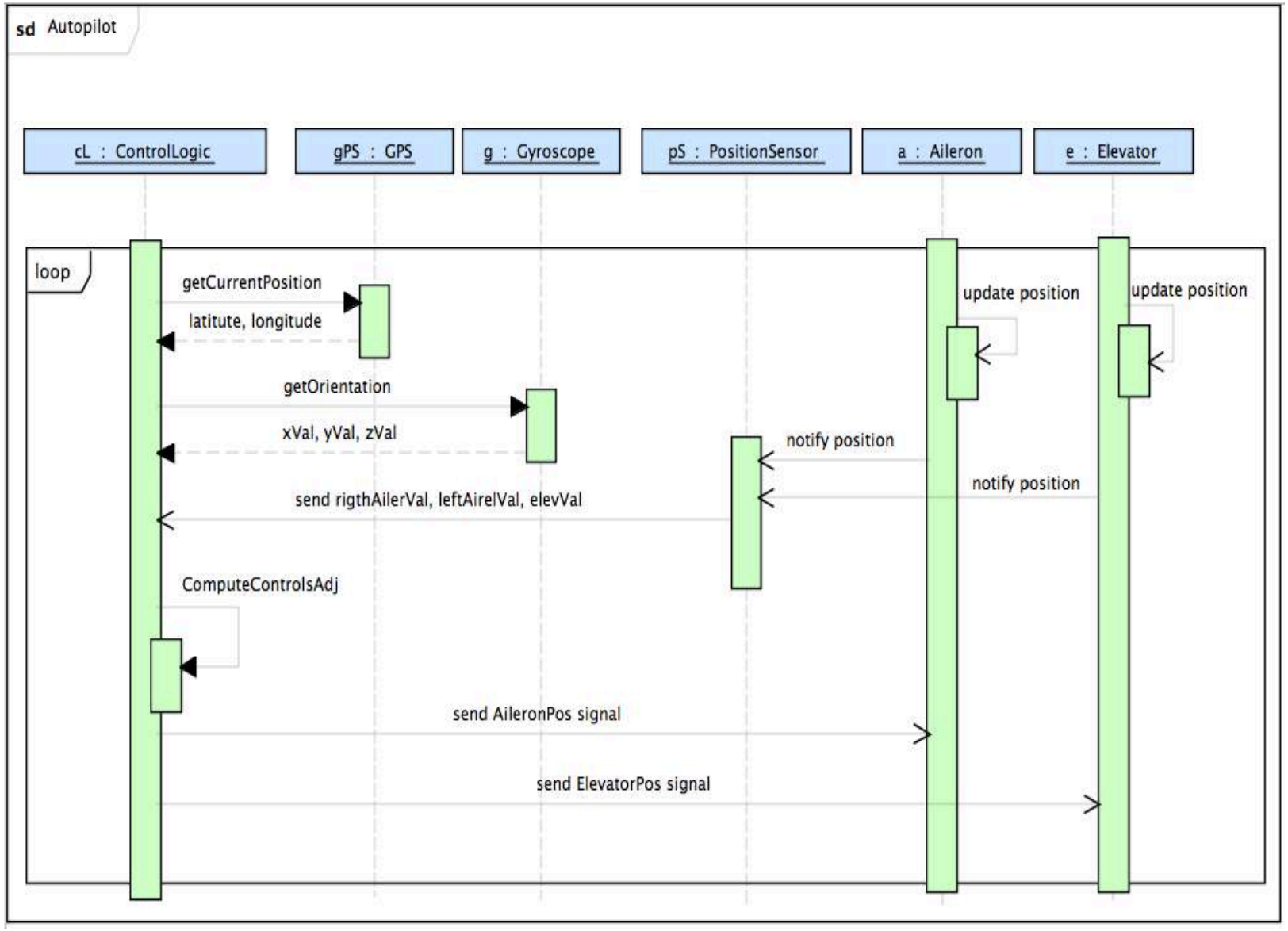
SysML Model (*BDD*)



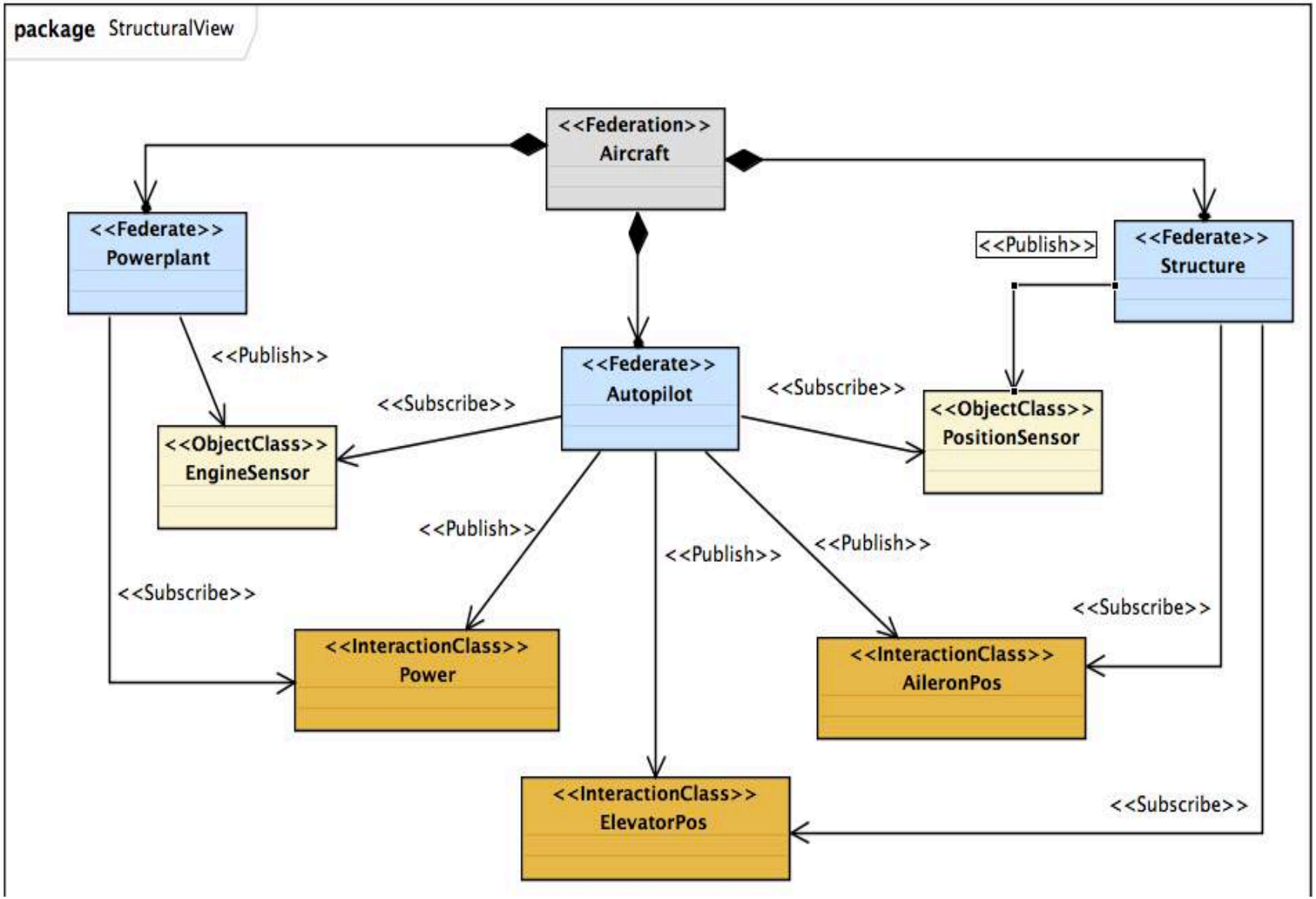
Example Application (*behavior*)



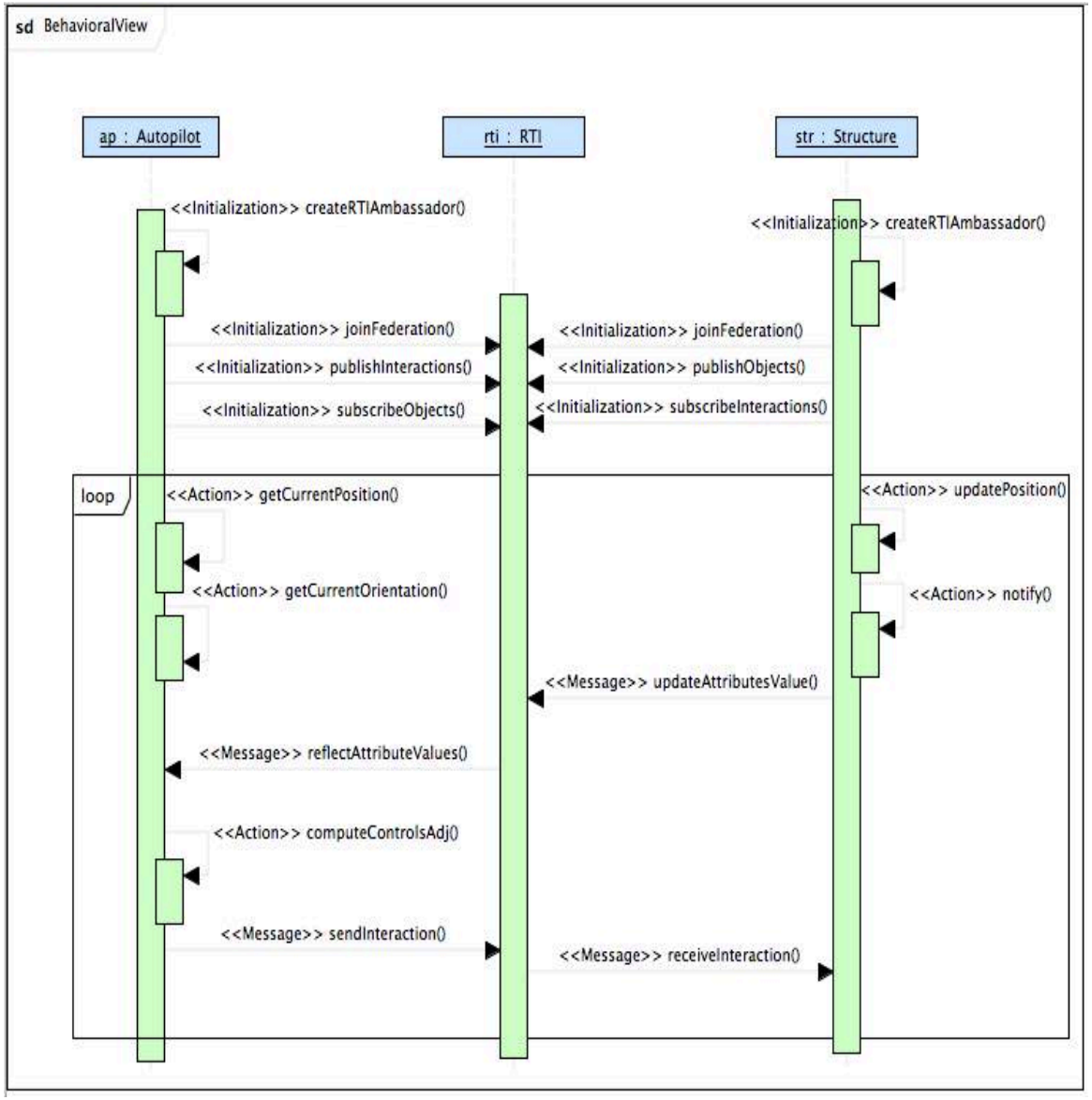
SysML Model (*SDs*)



HLA-based UML Model (*CD*)



HLA-based UML Model (*SDs*)



Java/HLA code (portion)

```
private AutopilotAmbassador fedamb;
private GPS gPS;
private Gyroscope gyroscope;
private ControlLogic manageControls;
private ControlLogic positionControl;
private ControlLogic controlLogic;

//Constructor
public Autopilot( GPS gPS,  Gyroscope gyroscope,  ControlLogic manageControls,  ControlLogic positionControl,  ControlLogic controlLogic, ){

this.gPS=gPS;
this.gyroscope=gyroscope;
this.manageControls=manageControls;
this.positionControl=positionControl;
this.controlLogic=controlLogic;

public static void main(String() args) {

        Autopilot a = new Autopilot();
        a.start(args (0));
    }
}
private void start(String crcHost){
        System.out.println("Starting HLA Infrastructure...");
        fedamb= new AutopilotAmbassador();
        try {
            fedamb.start(crcHost);

        System.out.println("HLA Infrastructure Started");
        }  catch (RTIexception e) {
            System.out.println("AutopilotAmbassador Error");
            e.printStackTrace();
            return;
        }

        while(true){
```

2. The `SimArch` approach

- Hides the local/distributed nature of the simulation system
- Hides the details of the specific DS infrastructure (e.g., HLA)
- Eases the switch between LS/DS systems
- Bridges the gap between the simulation model and its implementation (i.e., the DS/LS simulation system)
- Only requires mapping the PIM of the SUS to the simulation model

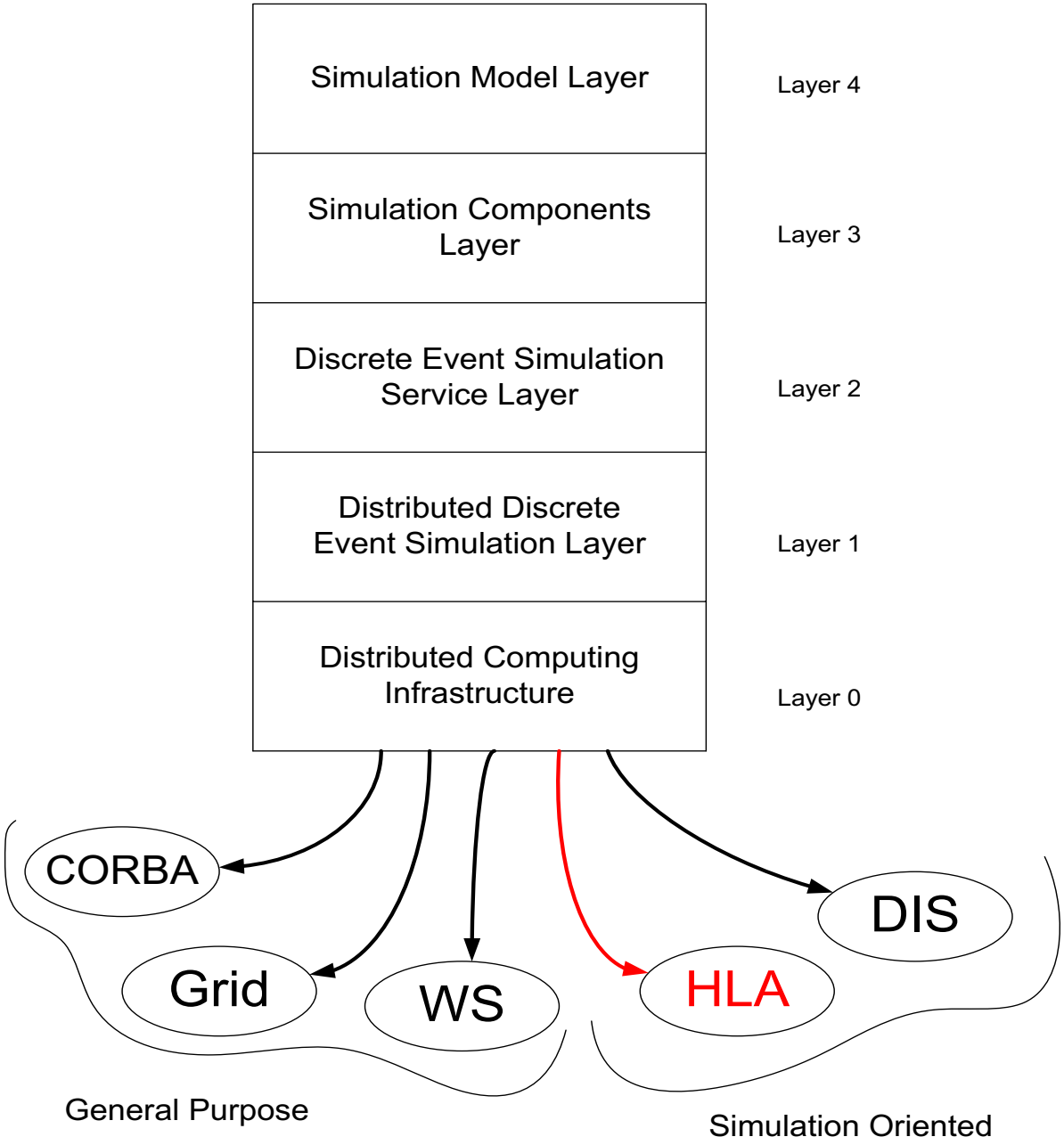
SimArch

- A layered architecture to enable the model-driven development of DS systems:
  - transparent deployment of simulation components in either a local or a distributed environment
  - transparent introduction/modification of the layers' implementation to meet additional/specific requirements
  - definition of custom (domain-specific) simulation languages on top of the layered architecture
  - support for *simulation-in-the-loop* approaches

SimArch Main Features

- Multiparadigm simulation environment
  - process interaction DES paradigm
  - agent-based modelling paradigm
- Composed of four layers
- Provides the definition of:
  - Service interfaces
  - Data interfaces
  - Factory interfaces for component instantiation

SimArch Layers



`SimArch` implementation status

- Layer 1
  - ***DDESoverHLA*** library: provides a DES abstraction on top of HLA
- Layer 2
  - ***SimJ*** library: provides generic simulation components
  - ***SimJ*** can be seen as a *metamodel* for defining domain specific simulation languages
- Layer 3
  - ***jEQN*** library: provides the primitives for defining EQN simulation models

Domain Specific Languages (DSLs)

- Programming language
- Domain-specificity
- Increased expressiveness (decreased generality)
- Ease-of-use
- Reduced domain and programming expertise
- Verificability and transformability
- Declarative
- Enabler of reuse

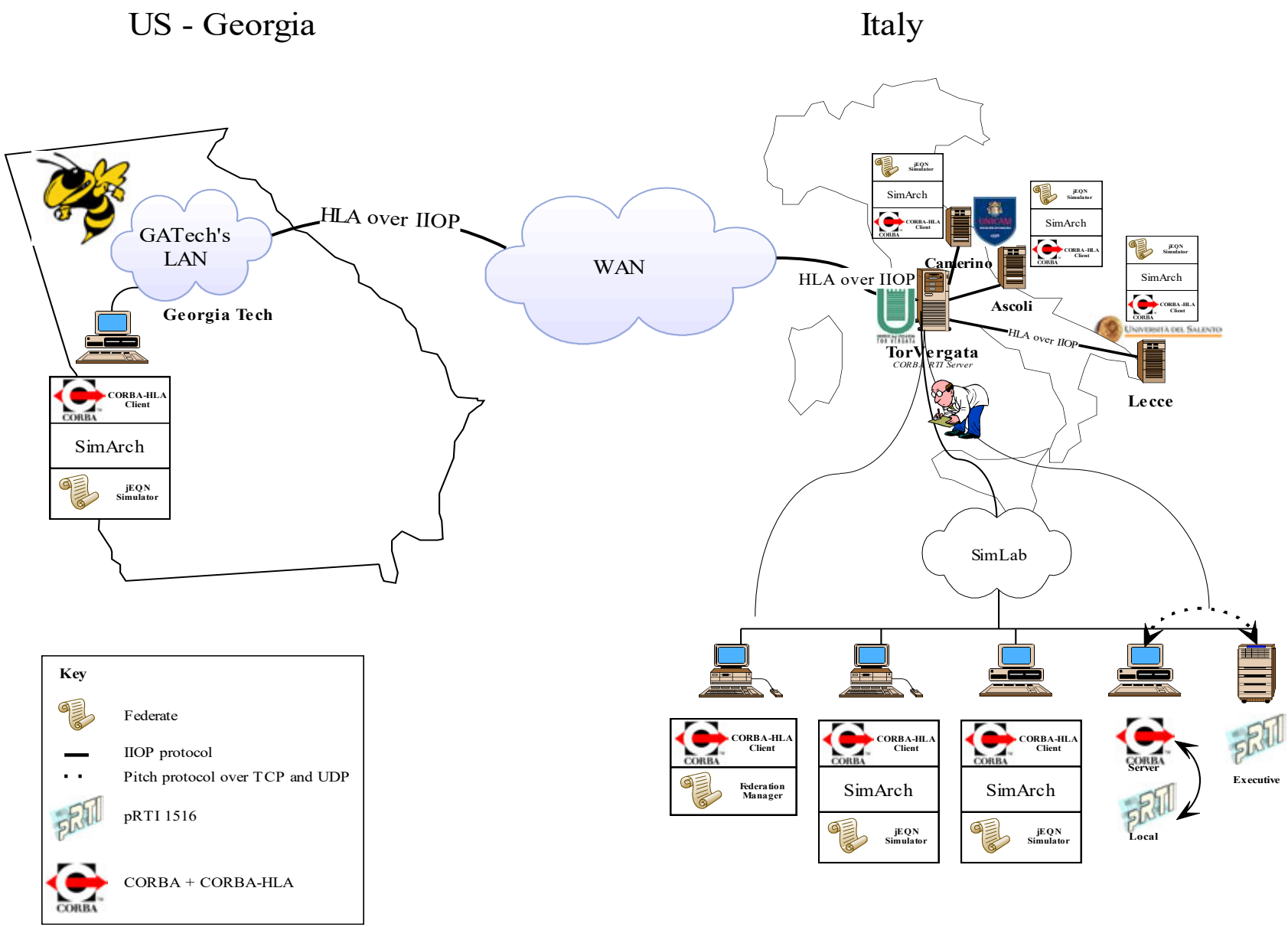
jEQN: a DSL for EQNs

- Based on:
  - Domain analysis of EQN models
  - Declarative approach (specify *what* to simulate rather then *how* to simulate)
- Used to:
  - Reduce the semantic gap between the model specification and the corresponding LS/DS system
- Composed of:
  - Simulation services defined by SimArch
  - Set of EQN simulation components
  - Set of parameters for the components

Example Application Domains

- Computer networks domain
  - *Distributed Computer systems* simulation
  - *Wireless systems* simulation
- Space systems domain
  - *Ground Segment* simulation
- Emergency management domain
  - *Building Evacuation* simulation

Example federates deployment



Conventional vs. SimArch

Features	*Conventional Approach*	*SimArch Approach*
Choice of DS Platform	Required	Not required
Choice of DS Implementation	Required	Not required
DSL-based	No	Yes
PIM (SUS) to PSM	Required (multiple)	Required (single)
PSM to Code	Required	Not required
Effort Savings	High	Very High
Maintainability	High	Very High
Reusability	High	Very High
Adaptability	Low	Very High
DS expertise	Low	Very Low

Model-based Interface Specification for Systems Integration

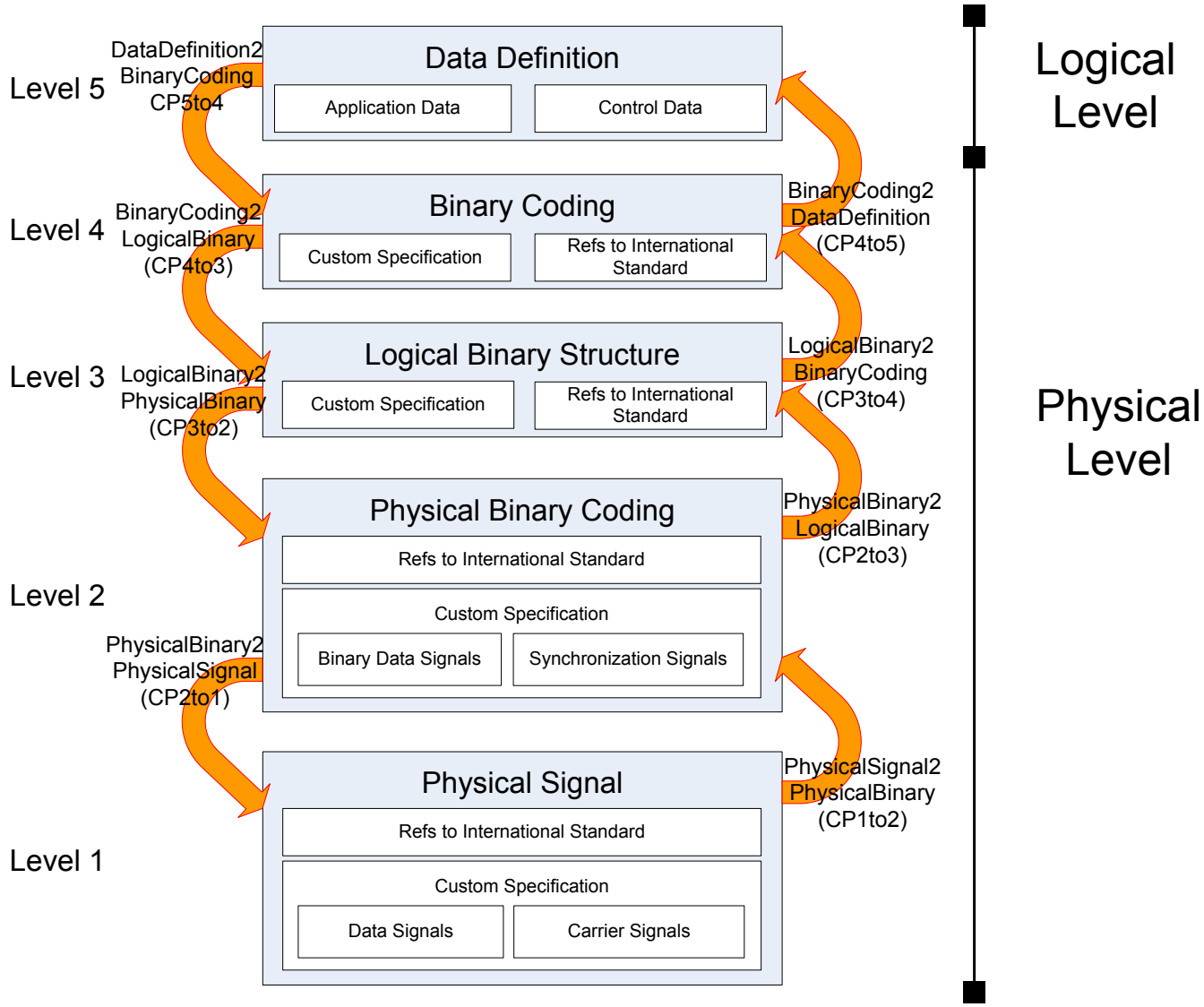
- The engineering of complex systems requires a careful consideration of the **interactions** among sub-systems and components
- Such interactions may reveal *significant anomalies* at *system integration* time
- Interface problems are even exacerbated in ***net-centric complex systems***, or systems of systems, due to the *heterogeneity* and *dynamicity* of constituent sub-systems and systems

ICML
(Interface Communication Modeling Language)

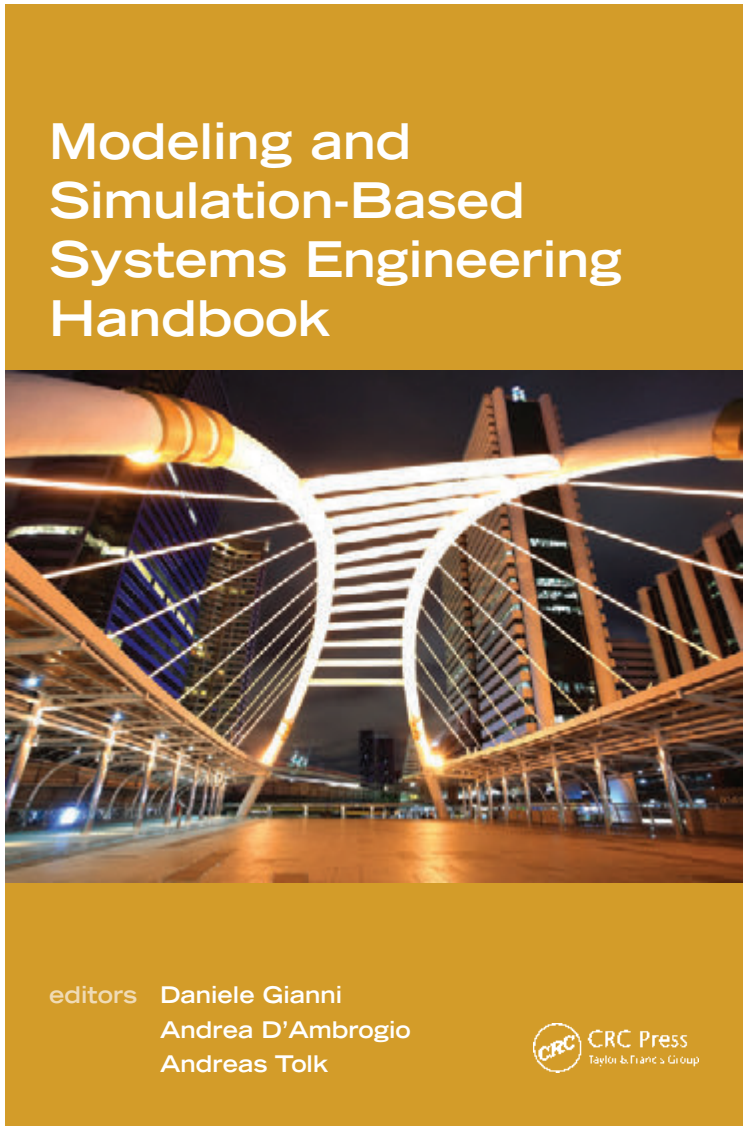
- A model-based language that can be used to graphically and unambiguously specify data interfaces, thus contributing to support the design of interoperable data communication systems
- Designed on a preliminary domain analysis on radio signal specifications (Time-Division Multiplexed signals)
- Developed within the *ESA SOCIS (Summer of Code in Space)* program - 2012 and 2013 editions
- Applied to:
  - Galileo receivers engineering, for supporting the reuse of existing HW and SW resources;
  - Service Systems Engineering for the *Galileo Open Service signal-in-space* interface specification
- Further info on:

  sites.google.com/site/icmlmodellinglanguage/

ICML Specification



M&S-based Systems Engineering Book



Modeling and Simulation-based Systems Engineering Handbook

by

Daniele Gianni
Andrea D'Ambrogio
Andreas Tolk

Pages: 464
Publisher: CRC Press, 2014
ISBN-10: 1466571454
ISBN-13: 978-1466571457