Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica (DISIM)

Taming the uncertainty: variability as a means for predictable system evolution

Paola Inverardi

Software Engineering and Architecture Group

Information Engineering, Computer Science and Mathematics Department,

University of L'Aquila, Italy

25 Gennaio 2013, VAMOS, Pisa

Road Map

Objective: Support dependable software evolution

My perspective is a SE one

Variability to Tame uncertainty

Three approaches

Future directions

Where do I come from …

Enhancing configuration facilities in software development: A logic approach (ESEC 1987)

P. Asirelli, P. Inverardi IEI-CNR PISA

Abstract

The paper focuses on the suitability and advantages of a *Logic Data Base approach to manage configurational aspects* within Programming Environments. It describes part of a work which proposes Logic Data Bases as effective tools to be integrated with existing programming environments to increase their formalization and automation capabilities. In order to present the idea and its implications, we discuss, as a practical example,

the integration of a prototype Logic DBMS (EDBLOG) with a Unix-like environment for configuration management. In that framework, a possible *realization of the Make facility is shown*. The advantages of the proposed approach are mainly concerned with the easiness of *extension of the programming environment and of the configuration environment* to deal *with concepts which, in general, are very expensive to provide, e.g. histories and versions management*.

Where do I come from …

Experimenting with dynamic linking with ADA (Elsevier S&P 1993)

Paola Inverardi, Franco Mazzanti IEI_CNR PISA

Keywords: Ada; Dynamic reconfiguration; Dynamic linking

Abstract An *approach to achieving dynamic reconfiguration* within the framework of Ada[1] is described. A technique for introducing a kernel facility for dynamic reconfiguration in Ada is illustrated, and its implementation using the Verdix VADS 5.5 Ada compiling system on a Sun3–120 running the 4.3 BSD Unix operating system is discussed. *This experimental kernel allows an Ada program to change its own configuration dynamically, linking new pieces of code at run-time*. It is shown *how this dynamic facility can be integrated consistently at the Ada language level, without introducing severe inconsistencies* with respect to the Standard semantics.

Setting the context

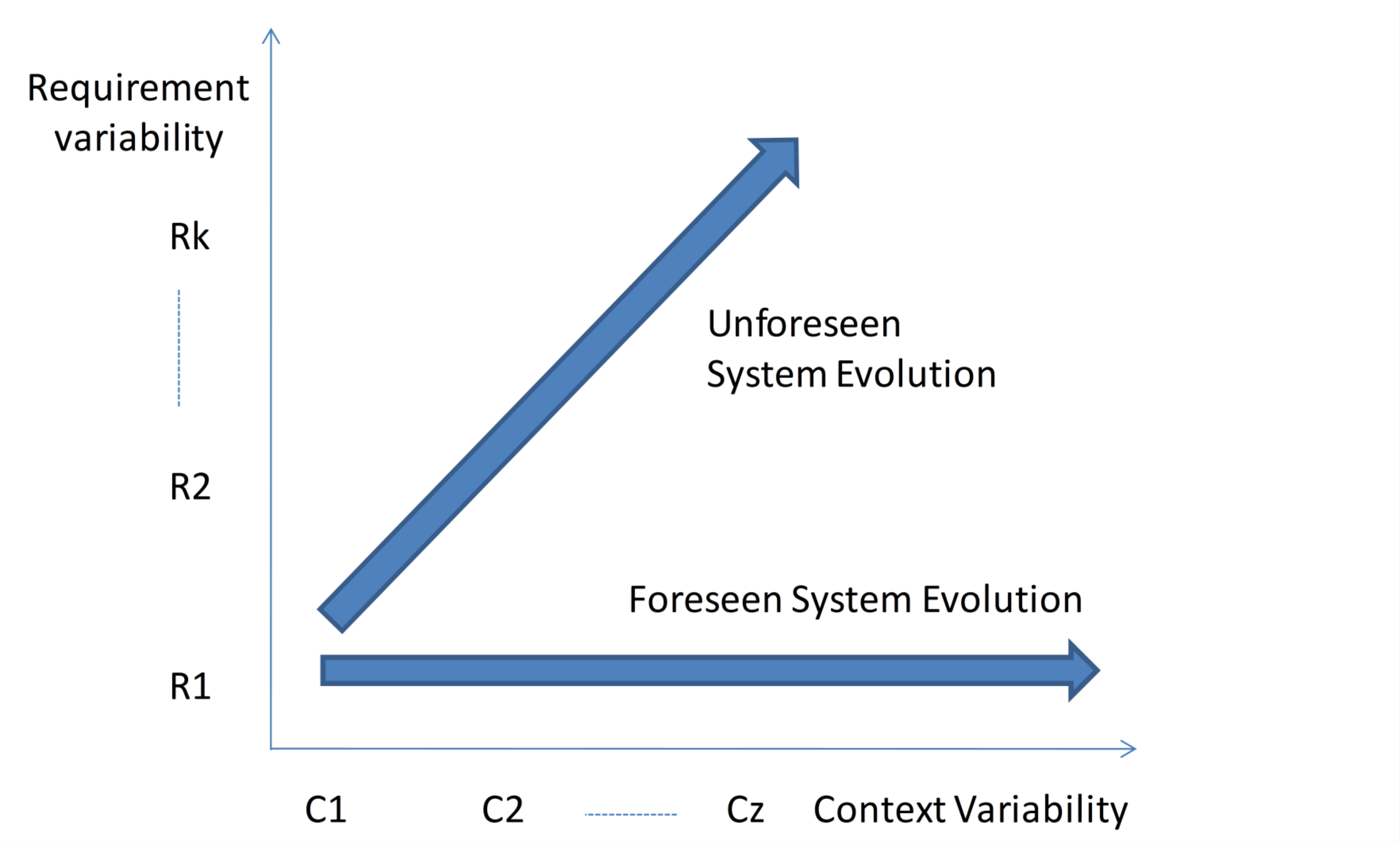
Ubiquitous software systems have to operate considering different (unpredictable) variability dimensions:

- Heterogeneity of the environment
- Changing user needs

(Self-)adaptive systems provide means to adjust their behavior in response to changes in the self and in the context:

- *Self* is the whole body of software, as represented in the whole set of artifacts that characterize the development and operation of the system (e.g. new requirements)
- *Context* is everything in the operating environment that affects the system properties and behavior

Evolution Taxonomy (1/2)



Evolution Taxonomy (2/2)

Foreseen Evolution:

foreseen context variations ⇒ selecting the most suitable variant[MoLi11] among the variants that are statically defined

Unforeseen Evolution:

unforeseen context variation ⇒ switching towards an un-anticipated system variant which satisfies a new requirement (@ run-time)

[MoLi11] M. Mori, F. Li, C. Dorn , P. Inverardi, S. Dustdar. "Leveraging State-based User Preferences in Context-aware Reconfigurations for Self-adaptive Systems". International Conference in Software Engineering and Formal Methods (SEFM). Montevideo, 2011

How to support variability?

Many ways at different system's abstractions and granularity (from requirements, to architecture, to code)

E.g. Software designer defines a set of software alternatives at design time for different **known** context

At **run-time** the system autonomously adopts the best variant based on the current context

- Context determines which variants are admissible and it helps to find the best reconfiguration possible

But…

- Contexts are not completely known at design time

Moreover…

At run-time, as a consequence of unforeseen environmental conditions new requirements may arise, thus:

- **the space of software alternatives must be augmented**

How to support consistent variability?

To prevent system incorrect behaviors, evolution has to be supported by validation mechanisms

- At design time: through validation of the known software alternatives
- At run-time: through validation of new software alternatives

(High-)assurance for adaptive systems:

- *“(high-)assurance provides evidence that the system satisfies continuously its functional or non-functional requirements thus maintaining the user’s expectations despite predictable and unpredictable context variations”*

Three approaches

- *A Framework to Support Consistent Design and Evolution of Adaptive Systems*
  - **Variability** at feature/component level foreseen and unforseen
  - **Consistency** of the configuration wrt requirements design and run time
- *Chamaleon for adaptable system*
  - **Variability** at programming level (adaptable classes) only foreseen
  - Consistency of the configuration wrt the context available resources deployment time
- *Service Choreography*
  - **Variability** at the service level
  - **Consistency** wrt the role required for the service behavior

A Framework to Support Consistent Design and Evolution of Adaptive Systems

System variability is expressed following the Software Product Line Engineering perspective (SPLE)

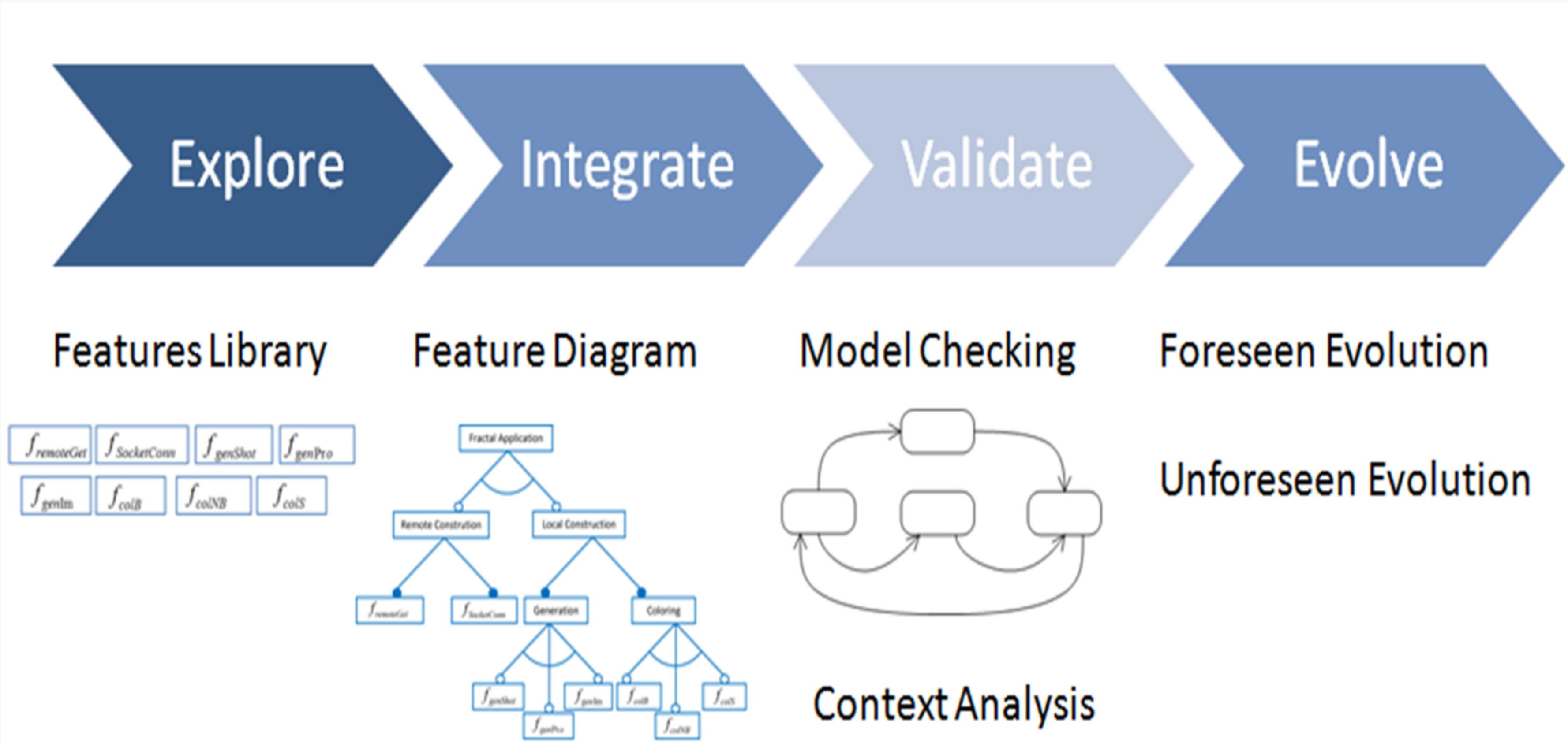
The single unit, the so called feature, represents the smaller part of a service that can be perceived by a user

Features are combined into configurations in order to produce the space of system alternatives

Inspired by SPLE we adopt the notion of *feature interaction phenomenon* as notion of high-assurance

A system configuration shows a *feature interaction phenomena* if its features run correctly in isolation but they give rise to undesired behavior when jointly executed

Software Lifecycle Process



[InMorCycle12] P. Inverardi and M. Mori. A software lifecycle process to support consistent evolutions. ", 2nd book on Software Engineering for Self-Adaptive Systems, 2012.

[AuDiIn11] M. Autili, D. di Ruscio, P. Inverardi, P. Pelliccione, M. Tivoli, and V. Cortellessa.EAGLE: Engineering softwAre in the ubiquitous Globe by Leveraging uncErtainty. new ideas track esec, 2011

Case Study: E-Health

- **E-Health distributed application to monitor vital parameters belonging to elderly people**
- **Probes sense patient information whereas the home gateway transmit them to the hospital**
- **Doctors visualize the trends of pulse oximetry and heart rate through PDA and desktop devices**
- **Adaptive behavior:**
  - Set of system alternatives to visualize the vital parameters at the doctor’s device as textual or graphical representation (possibly real-time)
  - Each alternative
    - has a different requirements specification
    - consumes a certain amount of resources to be provided by the environment (e.g. memory, CPU, etc…)

CASE STUDY: E-HEALTH

Context Model

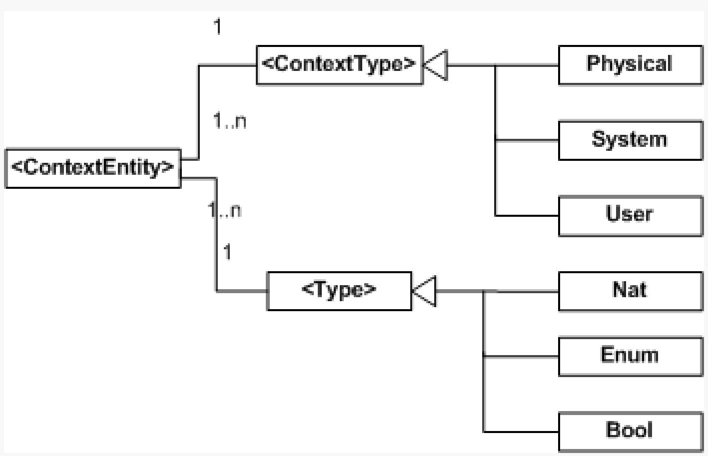
•Portion of the environment that is beyond the control of the system but may affect its behavior

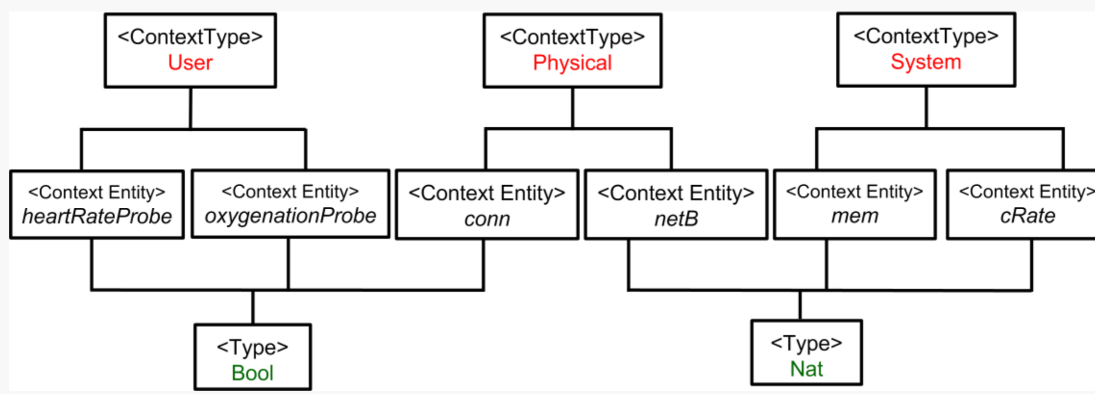
•Entails the set of entities (key-value pairs)

•Two perspective:

- Context structure: set of entities with context type (System, User, Physical) and type (Bool, Enum, Nat)
- Context space: set of valid assignments for the entities

  Context state: $\vec{c} \in$ $S =$ $m(ContextEntity)$





Adaptive Application

- **Space of software alternatives**
- **Each alternative is a different combination of features (configuration)**
- **We define a feature as a triples (R,I,C)** [ClHe08] [GL07] **where:**
  - R is a functional, performance or quality requirement (context independent)
  - I is the code implementation (e.g., Java)
  - C: constraint requirement (context dependent)
- **A configuration** $G_F = \langle R_F, I_F, C_F \rangle$ **is obtained by combining a subset of features F**
- **We assume to have an abstract union operator to combine features, which is expressed in terms of union operator for R, I and C**
  - Given two features $f_1 = \langle R_1, I_1, C_1 \rangle$ and $f_2 = \langle R_2, I_2, C_2 \rangle$ their union is defined as: $f_1 \cup_f f_2 = \langle R_1 \cup_R R_1, I_1 \cup_I I_2, C_1 \cup_C C_2 \rangle$

Context Requirements

Predicates over context entities

Syntax:

<C>::= <ContextEntity><rel-op><value>|<C><log-op><C>
<rel-op>::=≥|≤|>|<|=
<log-op>::=AND|OR
<value>::=<natural>

Each expression may be related to a single feature or to a system variant

EXAMPLE: FEATURE

$R_{\mathrm{graphOx}} =$ **If Oxigenation data are available, Receive Oxygenation rate and View it on the graphic widget - If "*OxygenationProbe*" then (Each 10 times "*getOximetryData*" follows a "*displayGraph*")**

$I_{\mathrm{graphOx}} =$

```
public class graphIOxygenationViewer{
  XYDataset oximetryDataset = new XySeriesCollection();
  …
  public void viewGraphicalOximetry (Graph g){
        ….
        for (i=0; i<10; i++){
          XYDataItem dataOx = OximetryRetrieving.getOximetryData();
          dataVectOx.add(dataOx);
        }
        g,displayGraph(dataVectOx);
        }…  }
```

$$C_{\mathrm{graphOx}} = mem \geq 50 \wedge cRate \geq \;000 \wedge \;\mathrm{xygenationProbe} = \mathrm{true}$$

EXAMPLE: SYSTEM VARIANT (1/2)

$R_{EHealth}: R_{graphOx} \cup_R R_{textOx} \cup_R R_{getOxData}$

$C_{EHealth}: mem \geq 70 \wedge cRate \geq 1100 \wedge oxygenationProbe = true \wedge conn = 1 \wedge b \geq 20$

$I_{EHealth}$:

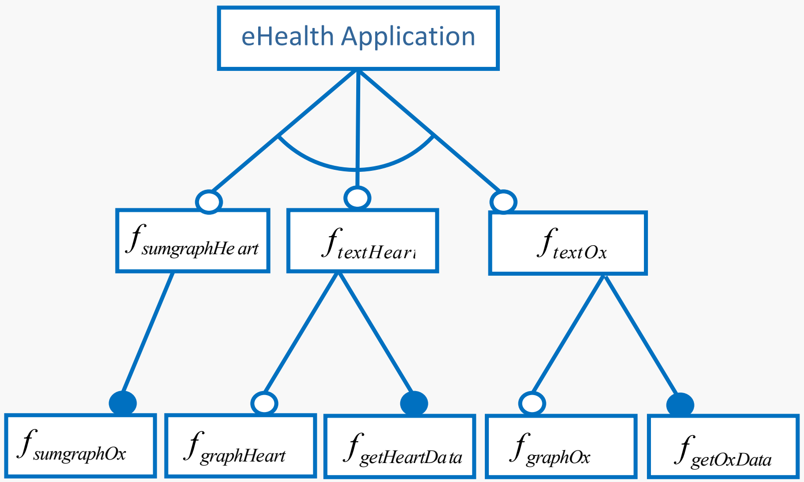
```
public class VariantEHealth{
 static Graph myGraphViewer;
 static Text myTextViewer;
 public static void execute(){
  myGraphViewer =   new Graph();
  myTextViewer = new Text();
  GraphOximetryViewer graphOx = new GraphOximetryViewer();
  TextOximetryViewer textOx = new TextOximetryViewer();
  graphOx.viewGraphicalOximetry(myGraphViewer);
  textOx.viewTextualOximetry(myTextViewer);
 }
 ...
}
public class GraphOximetryViewer{
 XYDataset oximetryDataset = new XYSeriesCollection();
 ...
 public void viewGraphicalOximetry(Graph g){
  ...
  for(int i = 0;i<10;i++){
   XYDataItem dataOx = OximetryRetrieving.getOximetryData();
   dataVectOx.add(dataOx);
  }
  g.displayGraph(dataVectOx);
 }
 ...
}
```

EXAMPLE: SYSTEM VARIANT (2/2)

```
public class TextOximetryViewer {
 ...
 public void viewTextualOximetry(Text myTextViewer) {
  XYDataItem dataOx = OximetryRetrieving.getOximetryData();
  myTextViewer.displayText(dataOx.getYValue());
 }
 ...
}

public class OximetryRetrieving{
 ...
 public static XYDataItem getOximetryData(){
  try {
   socket = (StreamConnection) Connector.open(connectionURL,
   Connector.READ_WRITE);
  }catch (Exception ex){
   System.out.println("Err.Open.Conn.To": +connectionURL);
   System.out.println(ex);
   }
  ...
  \* Get Oxygenation Data oxData*\
  ...
  DataOxymetryMeM.add(OxData);
  return oxData;
 }
 ...
}
```

Case Study: Feature Diagram



[CE00] Krzysztof Czarnecki and UlrichW. Eisenecker. Generative programming: Methods, Tools and Applications. Addison-Wesley, 2000

Evolution Consistency

- We adopt the feature interaction phenomenon as our notion of *consistency*
- Given a certain variant $G_F = (R_F, I_F, C_F)$ we define the *consistency* as: $I_F, C_F \vdash R_F$
  - (i) $C_F[\vec{c}/\vec{x}]$: context requirement satisfiability (**context analysis**) [InMoRe11]
  - (ii) R_F : context independent requirement satisfiability
  - (iii) $I_F \vdash R_F$: validates implementation w.r.t the context independent requirement (**model checking** or testing) [InMoCh11]

- □ Consistency check at design time → Foreseen Evolution
- □ Consistency check at run-time → Unforeseen Evolution

[InMo11] P. Inverardi and M. Mori. Requirements Models at Run-time to Support Consistent System Evolutions. In Requirements@Run-time. 2011

[InMoCh11] P. Inverardi and M. Mori. Model checking Requirements at run-time in Adaptive Systems. ASAS, 2011

CHAMELEON

A framework for the development and deployment of adaptable Java applications

Marco Autili, Paolo Di Benedetto and Paola Inverardi *Hybrid Approach for Resource-based Comparison of Adaptable Java Applications*. Science of Computer Programming (SCP) - 2012, DOI: http://dx.doi.org/10.1016/j.scico.2012.01.005

Marco Autili, Paolo Di Benedetto, Paola Inverardi: *Context-Aware Adaptive Services: The PLASTIC Approach*. FASE 2009: 124-139

Marco Autili, Paolo Di Benedetto, Paola Inverardi, Fabio Mancinelli: *A Resource-Oriented Static Analysis Approach to Adaptable Java Applications.* COMPSAC 2008: 1329-1334

Fabio Mancinelli, Paola Inverardi: *Quantitative resource-oriented analysis of Java (Adaptable) applications*. WOSP 2007: 15-25

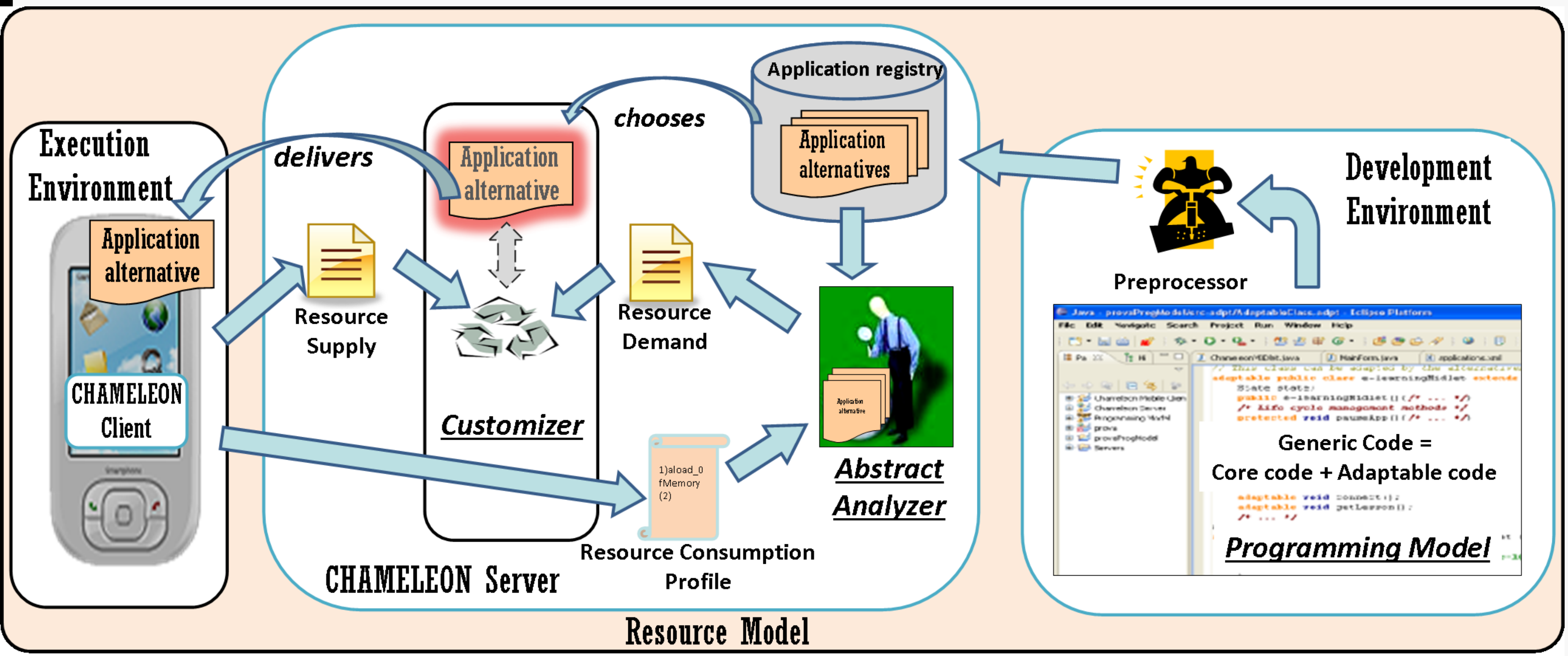
SUMMARY

- A programming model to develop adaptable applications reducing redundancy and promoting maintenance
- Models to represent and reason on resources
- An abstract analyzer that is able to estimate applications resource consumptions
- **An integrated framework that enables the development, discovery and deployment of adaptable applications and services.**

❑ ***Resource-aware adaptation***

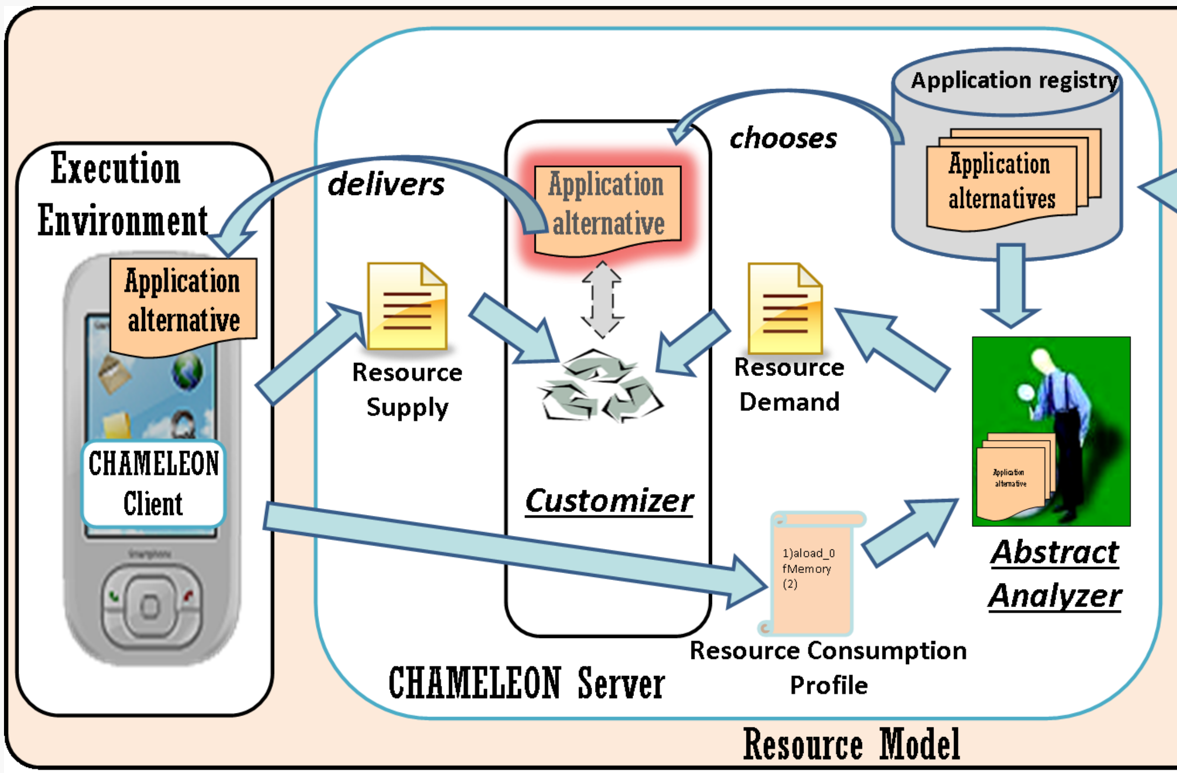
The applications used to provide and/or consume services are implemented as "*generic*" code that, at discovery time, can be customized (i.e., tailored) to run correctly on the actual execution context.

CHAMELEON Framework



- Programming Model: permits to implement applications in terms of ***generic code*** (extension to the Java language)
  - ***core code + adaptable code***
- Preprocessor: derives from the generic code a set of ***application alternatives***, i.e., different standard Java components that represent different ways of implementing the same service.

Development Environment



Programming Model

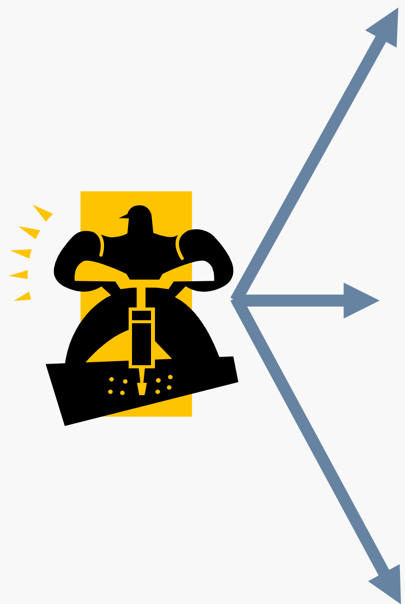
```
adaptable class C {
    adaptable void m1 ( ) ;
    adaptable void m2 ( ) ;
}

alternative class A1 adapts C {
    void m1( ) { . . . }
    void s1 ( ) { . . . }
}

alternative class A2 adapts C {
    void m1( ) { . . . }
}

alternative class A3 adapts C {
    void m2( ) { . . . }
}

alternative class A4 adapts C {
    void m1( ) { . . . }
    void m2( ) { . . . }
}
```

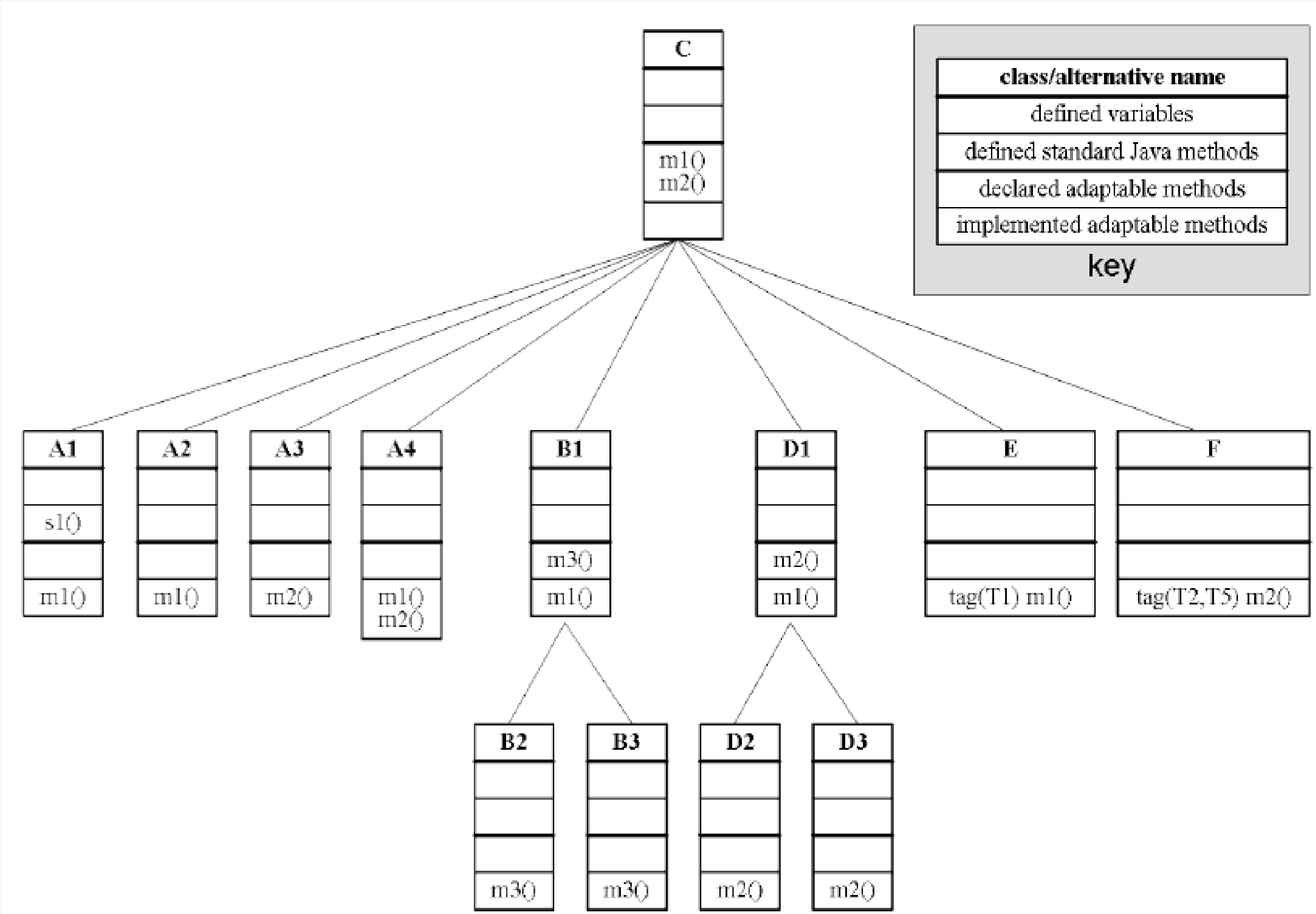


```
class C {
    void m1 ( ) { . . . } // from A2
    void m2 ( ) { . . . } // from  A3
}

class C {
    void m1 ( ) { . . . } // from  A1
    void s1() { . . . } // from  A1
    void m2 ( ) { . . . } // from  A3
}

class C {
    void m1 ( ) { . . . } // from A4
    void m2 ( ) { . . . } // from A4
}
```

Alternatives Tree

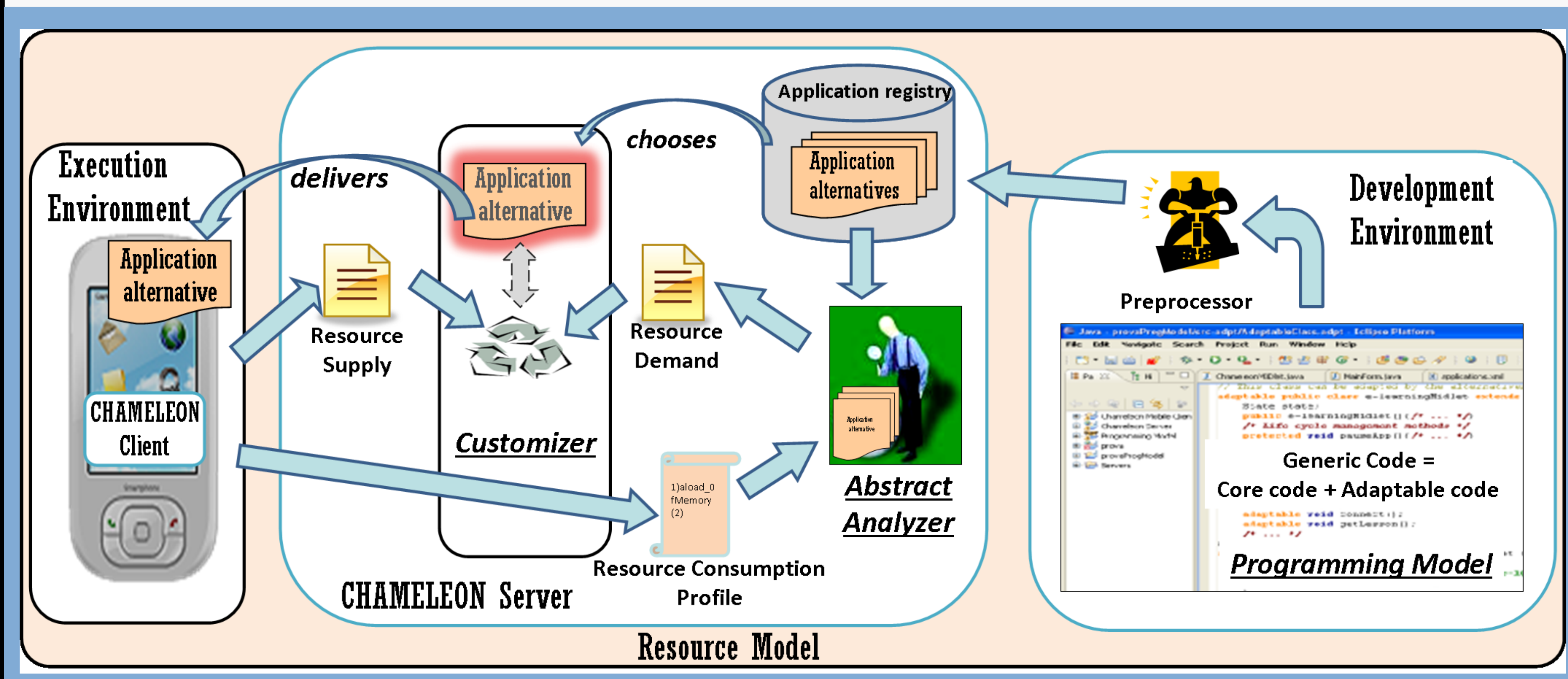


ADAPTABLE APPLICATION PREPROCESSING

C

C.1	{ A1.m1(); A1.s1(); A3.m2() }
C.2	{ A2.m1(); A3.m2() }
C.3	{ A4.m1(); A4.m2() }
C.4	{ B1.m1(); B2.m3(); A3.m2() }
C.5	{ B1.m1(); B3.m3(); A3.m2() }
C.6	{ D1.m1(); D2.m2() }
C.7	{ D1.m1(); D3.m2() }
C.8	{ tag(T1)E.m1(); A3.m2() }
C.9	{ A1.m1(); A1.s1(); tag(T2; T5)F.m2() }
C.10	{ A2.m1(); tag(T2; T5)F.m2() }
C.11	{ B1.m1(); B2.m3(); tag(T2; T5)F.m2() }
C.12	{ B1.m1(); B3.m3(); tag(T2; T5)F.m2() }

Resource Model



Resource Model

Resource Model: formal model for resources

Resource: **entity required to accomplish an activity/task.**

CHAMELEON Resources as typed identifiers:

- *Natural* for consumable resources (Battery, CPU,...)
- *Boolean* for non consumable resources that can be present or not (API, network radio interface, ...)
- *Enumerated* for non consumable resources that admits a limited set of values (screen resolution, …)

Resource Instances and sets

Resource Instance

- Association resource(value)
  - e.g. Bluetooth(true)

Resource Set

- a set of resource instances, with no resource occurring more than once

Resource Sets are used to specify

- Resource Supply: {Bluetooth(true), Resolution(low), Energy(30)}
- Resource Demand: {Bluetooth(true), Resolution(high)}

Compatibility

- Used to determine if an application can run safely on the execution environment

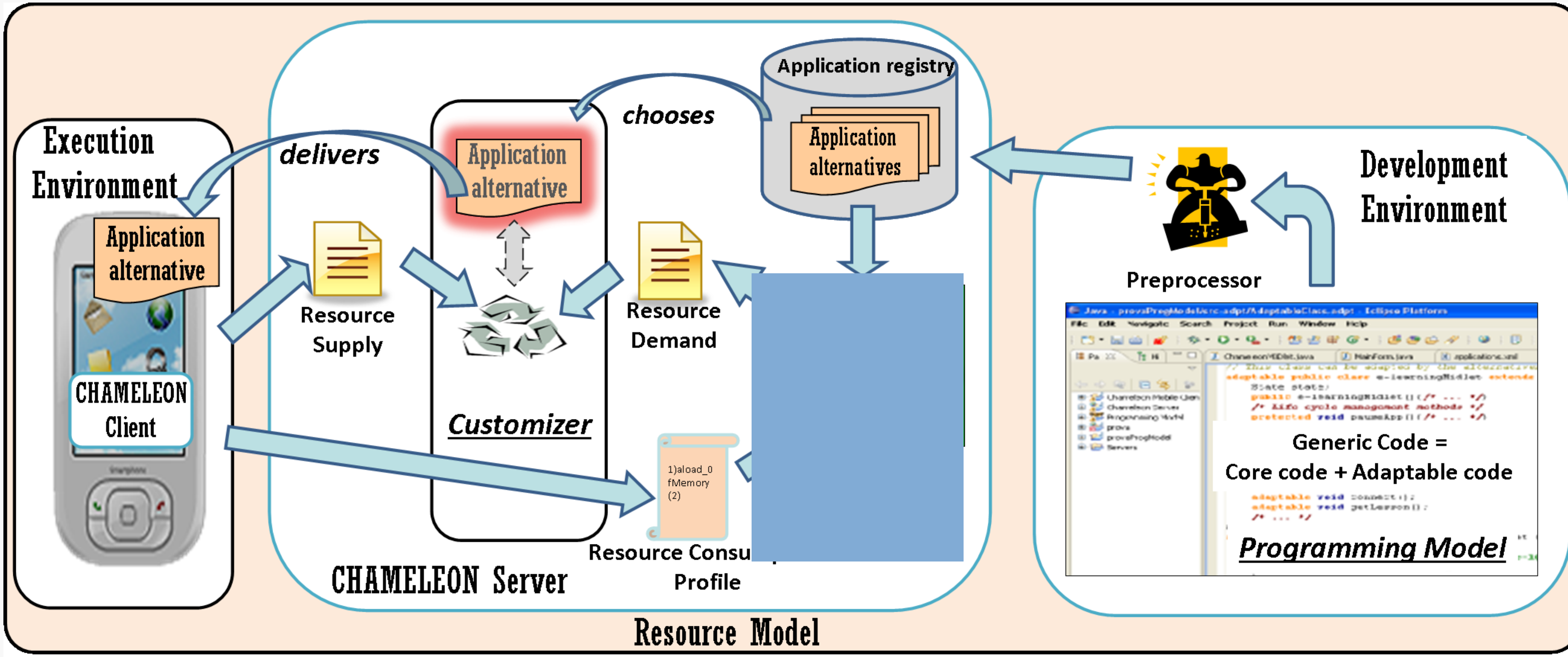
- A resource set *(demand)* P is compatible with a resource set *(supply)* Q ($P \lhd Q$) if:
  1. **(Availability)** For every *resource instance* r(x) $\in$ P there exists a *resource instance r(y)* $\in$ Q.
  2. **(Wealth)** For every pair of *resource instances* r(x) $\in$ P and r(y) $\in$ Q, $p(x) \leq p(y)$.

- A resource sets family (demand) P is compatible with a resource set (supply) Q, if $P_i \lhd Q$, $\forall P_i \in P$.

Goodness

- used to choose the best compatible application alternative w.r.t. a given execution context
- based on a notion of priority (P) among resources that expresses the "importance" given to a particular resource consumption
- P:Resources→Integer.
  - P(r) < 0 → the less r is consumed the better is (e.g., Energy).
  - P(r) = 0 → the consumption of resource r is ininfluent (Bluetooth)
  - P(r) > 0 → the more r is consumed the better it is (e.g., Thread)

Abstract Analyzer



- Statically analyzes each application alternative
  - impact that bytecode instructions have on resources
- Abstracts a standard Java Virtual Machine
- Derives the Resource Demand (and the Code-embedded SLS)
- Worst case analysis based on the resource consumption profile

Resource Consumption Profiles

Provides the description of the characteristics of a specific execution environment

Specifies the impact that Java bytecode instructions (patterns) have on resources

1) istore_1 → {CPU(2)}
2) invoke.* → {CPU(4)}
3) .* → {CPU(1), Energy(1)}
4) invokestatic LocalDevice.getLocalDevice() → {Bluetooth(true), Energy(20)}

Can be created on the basis of:

- experimental results based on benchmarking tools
- Information provided by device manufacturers, network sensors …

Always exists a default Resource Consumption Profile

The more are accurate, the more the analysis is precise

ANALYZER : STANDARD CONTROL FLOW RULE

$$\frac{\begin{array}{c} m = \langle \texttt{cid}, n_m, t_m \rangle \quad t_m(pc) = \texttt{instruction} \\ r_1 = b(\texttt{instruction}) \\ r' = r \oplus r_1 \oplus ResourceAnnotation(pc) \end{array}}{\langle e, b, m, pc, r \rangle \rightarrow_{ARA} \langle e, b, m, pc + 1, r' \rangle}$$

0: aload_0
werOnZ
3: invokestatic LocalDevice.getLocalDevice( )
4: astore_1
5: goto −> 10
6: aload_0
7: get field screenLScreen ;
8: ldc " Please swi tch on Bluetooth "
9: invokevirtual display.out (Ljava/lang/St r i n g)
10: return

$\langle$ e, b, m, 0, $\varnothing$ $\rangle$

$\langle$ e, b, m, 1, { , $\rangle$

$\langle$ e, b, m, 2, { } $\rangle$

1)
2) invokestatic LocalDevice.getLocalDevice() → {Bluetooth(true), Energy(20)}
3)

ANALYZER: CONDITIONAL FORWARD JUMP RULE

$$\frac{\begin{array}{c} m = \langle \mathtt{cid}, n_m, t_m \rangle \quad t_m(pc) = \mathtt{if\_TcmpOP} \ \mathit{addr} \quad \mathit{addr} > pc \\ r' = r \oplus b(\mathtt{if\_TcmpOP} \ \mathit{addr}) \\ \langle e, b, m, pc+1, r' \rangle \xrightarrow{*}_{ARA} R_{b1} \\ \langle e, b, m, \mathit{addr}, r' \rangle \xrightarrow{*}_{ARA} R_{b2} \\ R = R_{b1} \bigcup R_{b2} \end{array}}{\langle e, b, m, pc, r \rangle \xrightarrow{*}_{ARA} R}$$

0: **aload_0**

3: **invokestatic** LocalDevice.getLocal ... ;

4: **astore_1**

5: **goto** –> 10

6: **aload_0**

7: **get field** screenLScreen ;

8: **ldc** " Please swi tch on Bluetooth "

9: **invokevirtual** display.out ....

10: **return**

⟨ e, b, m, 2, {CPU(3), Energy (2)} ⟩

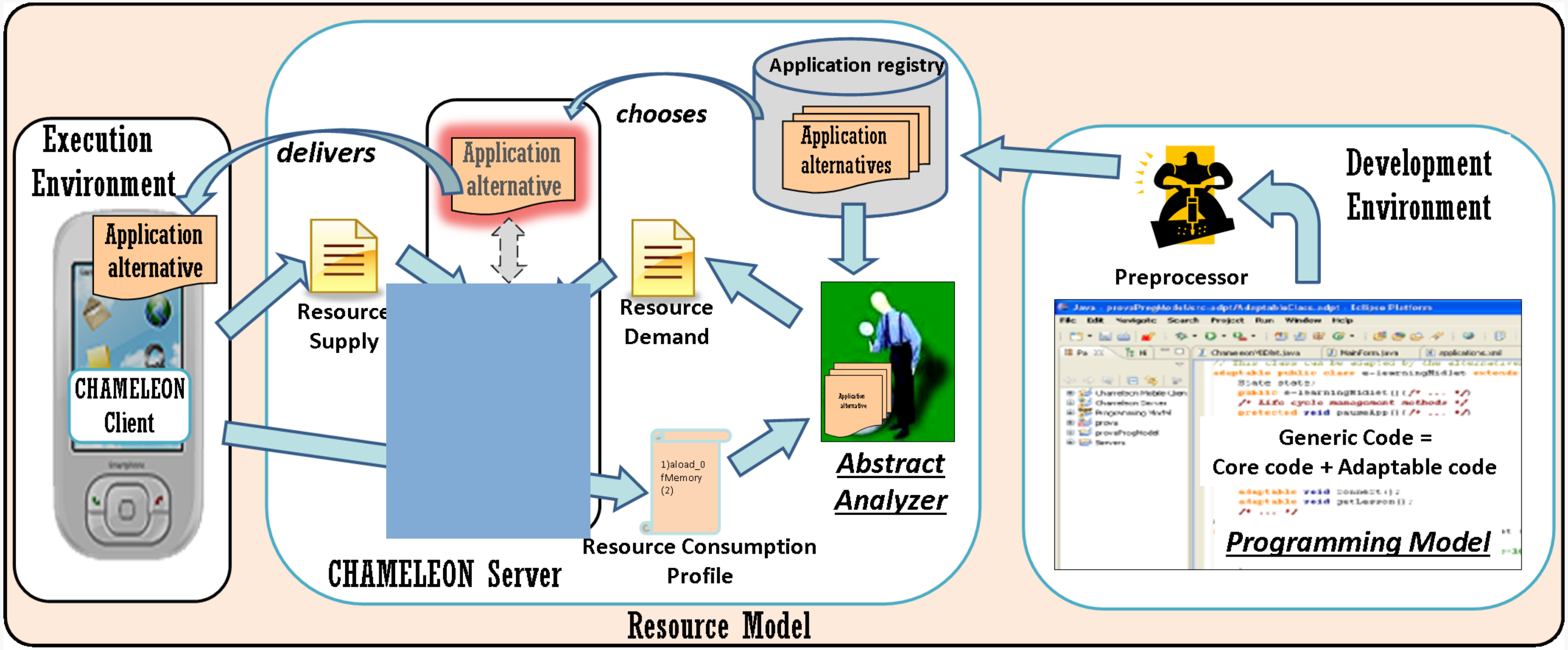
$r' = \{CPU(4), Energy\ (3)\}$

$R_{b1} = \{CPU(8), Energy\ (26), Bluetooth(true)\}$

$R_{b2} = \{CPU(10), Energy\ (8)\}$

$R = R_{b1} \cup R_{b2} = \{ \ \{CPU(8), Energy\ (26), Bluetooth(true)\}, \ \{CPU(10), Energy\ (8)\} \ \}$

CUSTOMIZER



- Compares the *resource demand* of the alternatives with the *resources supplied* by the execution context
- Determines the application alternatives that can run safely in the execution context (i.e., *compatibility*)
- The selected application alternative is then deployed (via OTA)

Discussion

Adaptation is restricted at discovery time, that is at the moment in which the service execution context and the user QoS preferences are known

- cost effective and suitable also for limited devices
- unpredictable context changes might invalidate the SLA
  - a re-negotiation of the SLA is necessary
  - services need to be adapted at run-time

Note it is aimed at selecting alternatives not at measuring absolute consume of resources!

What about providing self-evolving services?

CHOREOS: LARGE SCALE CHOREOGRAPHIES FOR THE FUTURE INTERNET

FP7-ICT-2009-5 COLLABORATIVE PROJECT

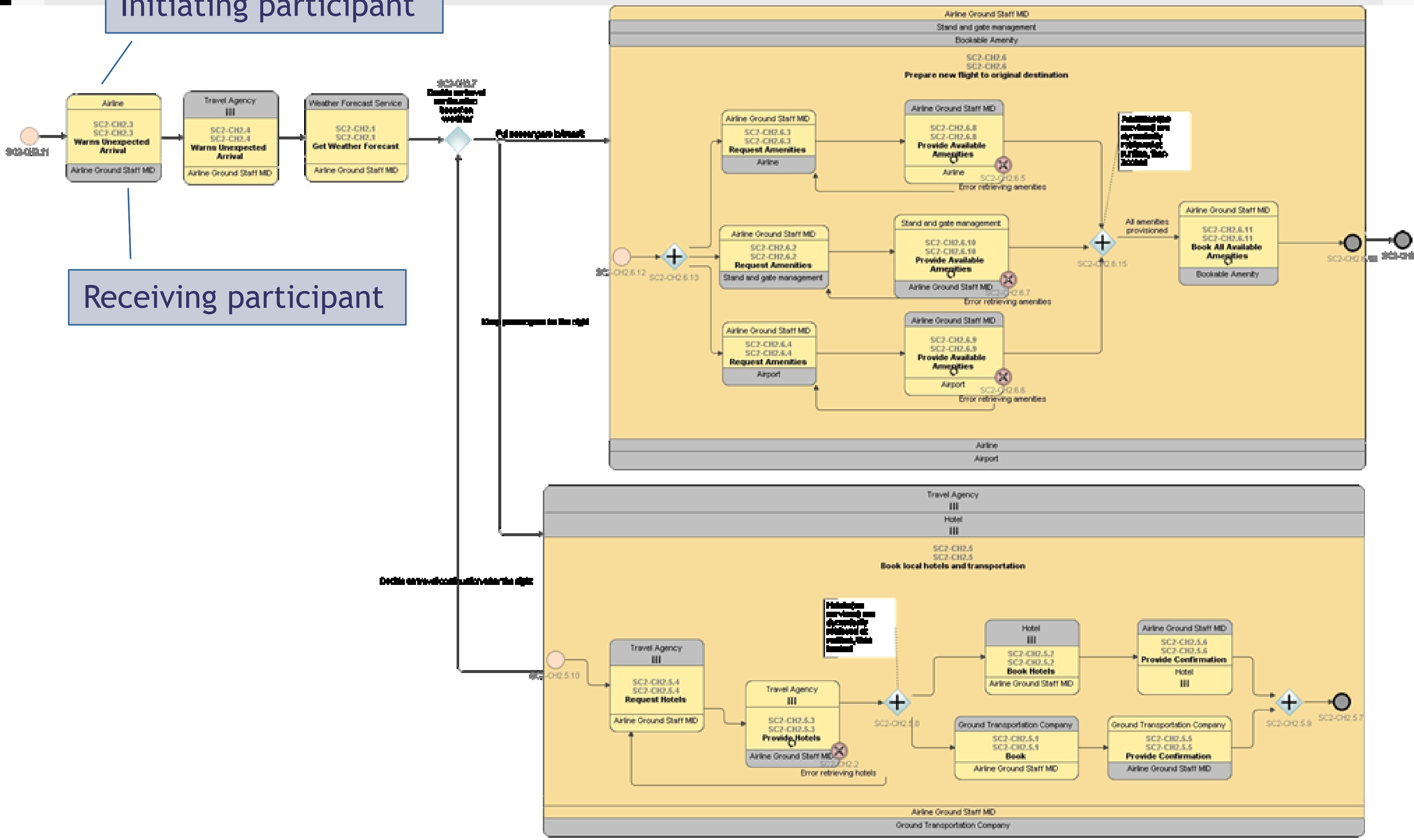
- CHOReOS introduces a dynamic development process, and associated methods, tools and middleware sustaining the ever-adaptable composition of services by domain experts being the users of business choreographies in the Future Internet

- M. Autili, D. Di Ruscio, A. Di Salle, P. Inverardi, M. Tivoli *A Model-Based Synthesis Process for Choreography Realizability Enforcement*, FASE 2013, LNCS 7793 to appear
- All references on Synthesis project

BPMN2 CHOREOGRAPHY EXAMPLE

Initiating participant

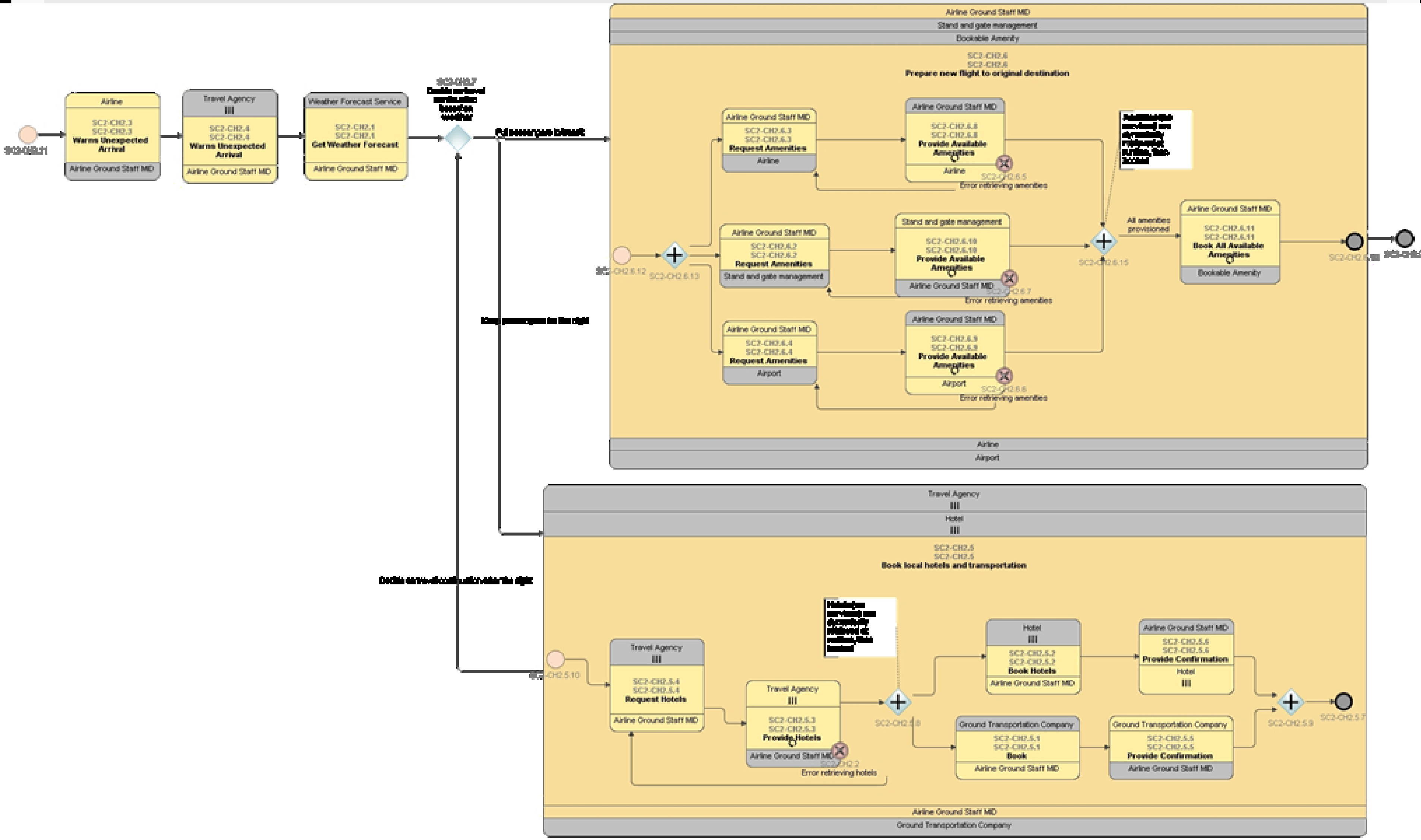
Receiving participant



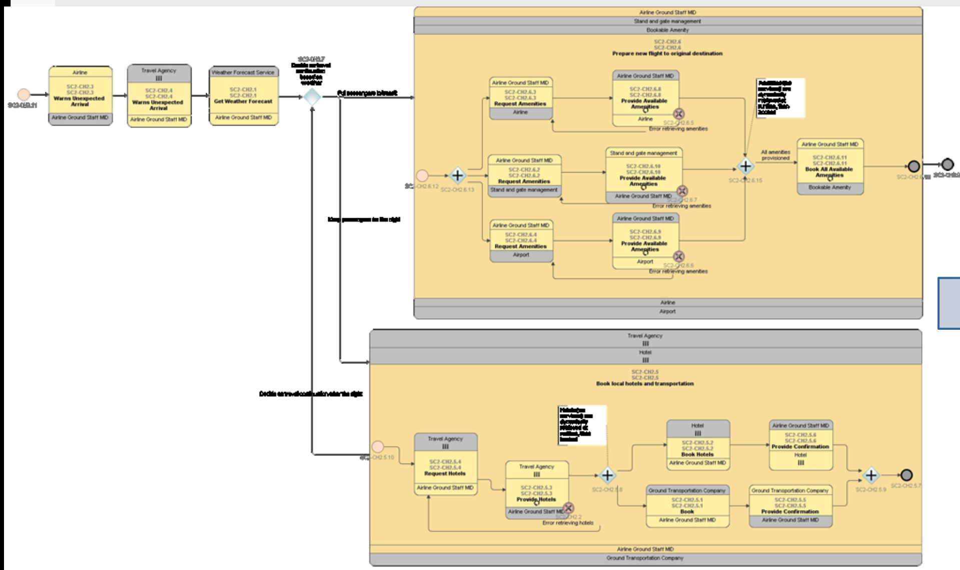
Choreography Variability points

- A choreography specification has variability points related to the notion of participant roles
- For each participant, a role specifies the interaction behavior that a service has to support in order to be able to play the role of the participant in the choreography
- For a given participant, its role can be obtained through projection

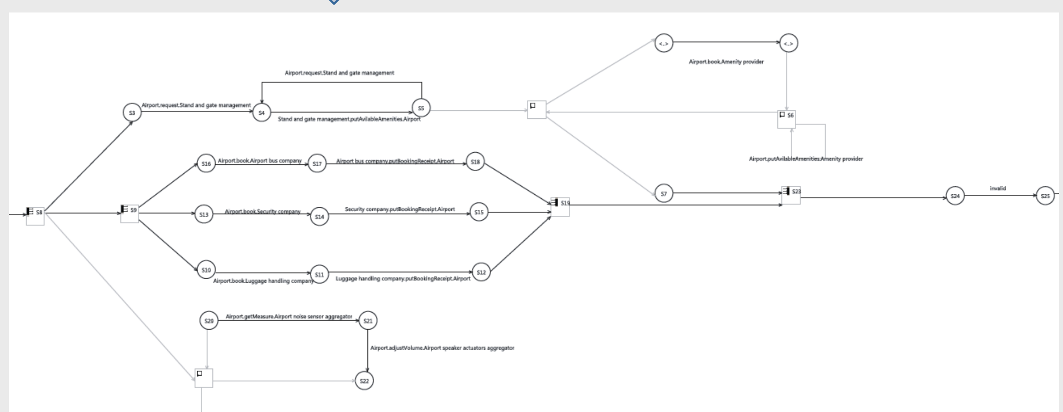
From BPMN2 to CLTS



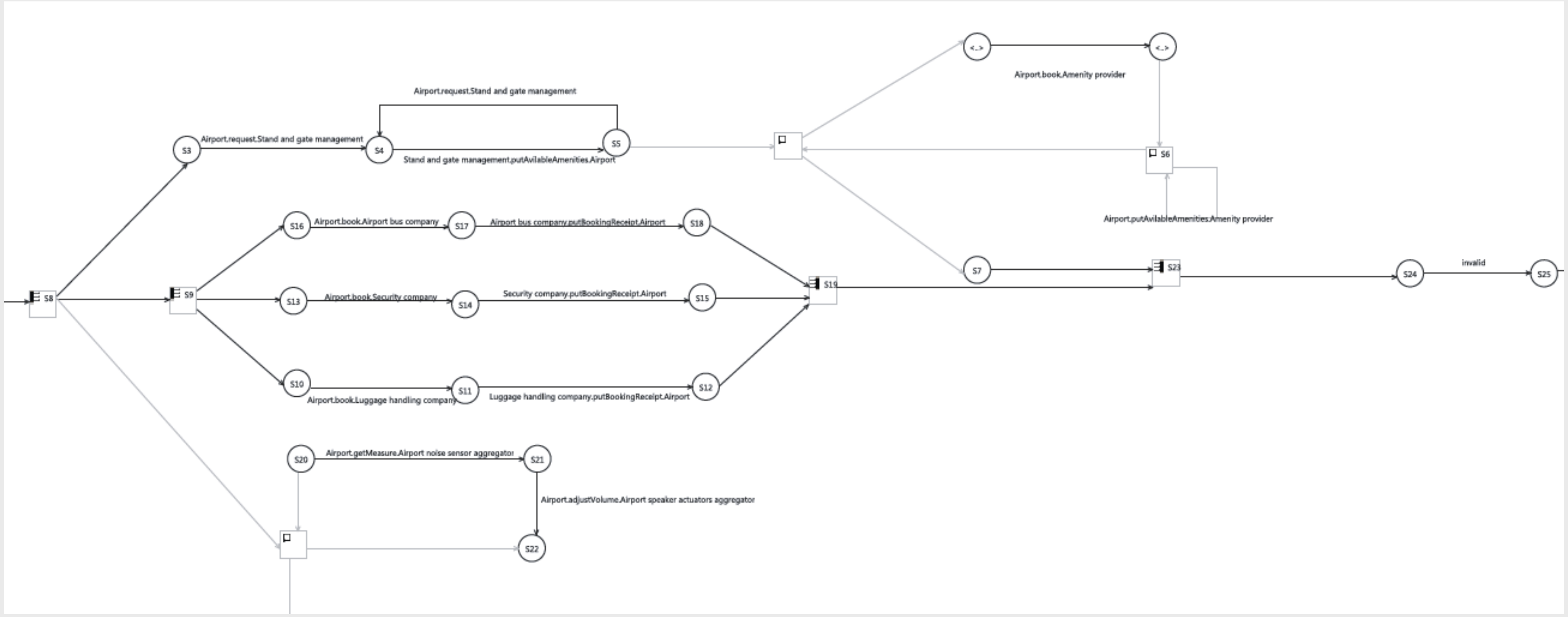
From BPMN2 to CLTS



For coordination purposes, the BPMN2 specification is transformed to an extended LTS, called Choreography LTS (CLTS)



Choreography LTS (CLTS)

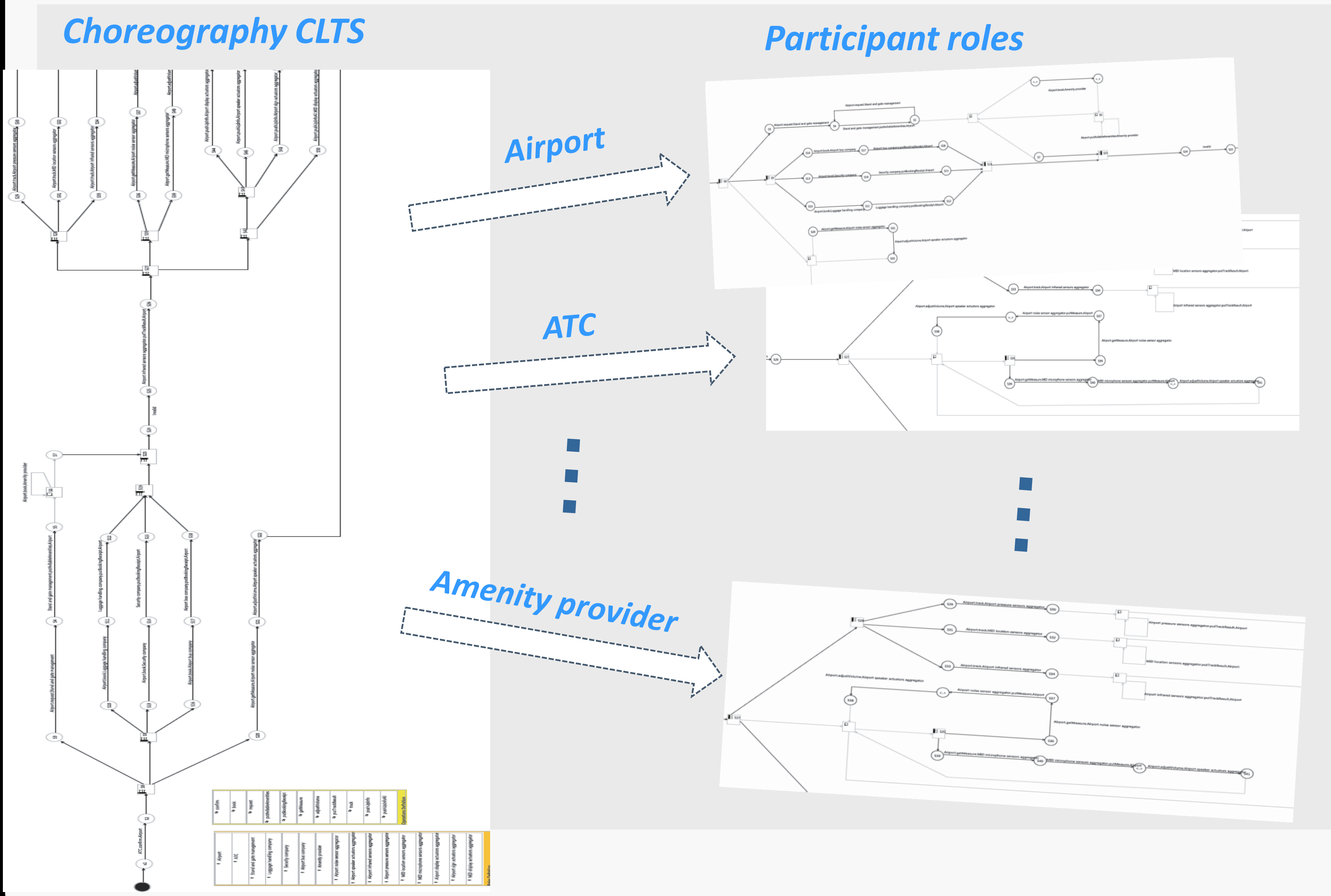


A CLTS is an LTS that, for coordination purposes, is suitably extended with fork and join constructs, conditional branching and loops.

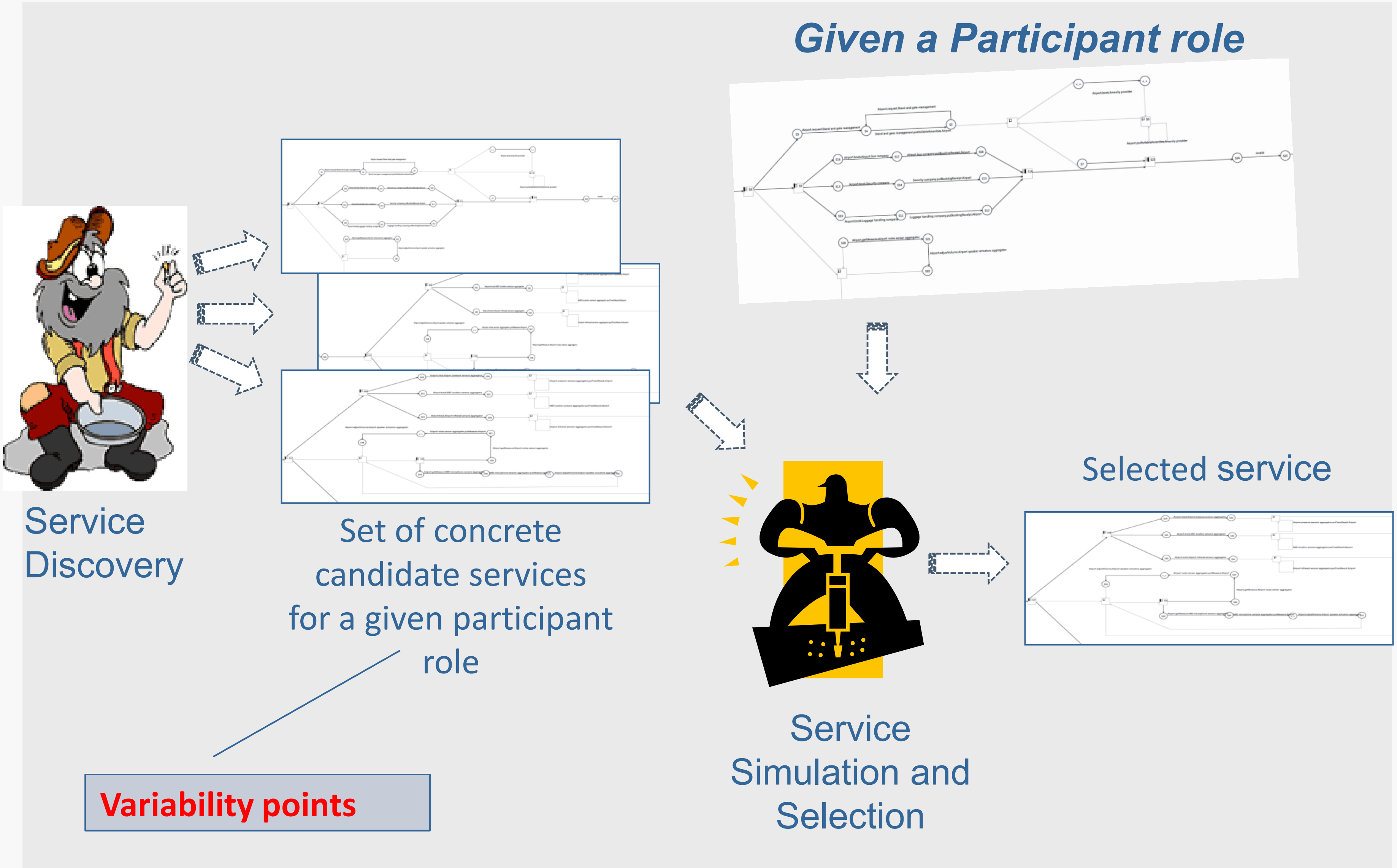
CHOReOS distinguishes:

- Generative approaches
    - services are aptly developed for the specific choreography
- Non generative approaches
    - services are discovered from a service registry
    - the discovery phase retrieves those services whose behaviors (specified as CLTSs) is **compatible with the roles** as extracted from the choreography through **projection**
    - to check **compatibility**, a suitable notion of **simulation** is applied to extended LTSs

Choreography projection



Discovery through simulation



CONCLUSION AND FUTURE WORKS

- **Evolving systems in the Ubiquitous world**
  - unpredictable evolutions
  - assurance/dependability

Explicit variability confines evolution in precise boundaries and helps controlling unpredictability by making analysis possible

Explicit variability classicaly describes what is going to change

A complementary approach is to establish variability implicitely by determining what is **NOT** going to change

Purely constrained approaches make analysis easier but less control on the variants

Trade off in between generality and precision