Assessing dependability for mobile and ubiquitous systems: Is there a role for Software Architectures?

Paola Inverardi

Software Engineering and Architecture Group
Dipartimento di Informatica
Università degli Studi dell'Aquila
I-67100 L'Aquila, Italy

QSIC 2010, Zhangjiajie, China

Setting the context

» Software architecture

- gives structure to the composition mechanism
- imposes constraints to the interaction mechanism
  - > roles, number, interaction mode, etc.

» Mobile & Ubiquitous scenario

- location-based
- resource-aware
- content-based
- user-need-aware

Context Awareness

- (Physical) Mobility allows a user to move out of his proper context, traveling across different contexts.
- How different? In terms of (Availability of) Resources (connectivity, energy, software, etc.) but not only …
- When building a *closed* system the context is determined and it is part of the (non-functional) requirements (operational, social, organizational constraints)
- If contexts change, requirements change → the system needs to change → evolution

When and How can the system change?

- When? Due to contexts changes → while it is operating → at run time
- How? Through (Self)adaptiveness/dynamicity/evolution
  Different kind of changes at different levels of granularity, from software architecture to code line
- Here we are interested in SA changes

The Challenge for Mobile & Ubiquitous scenario

» Context Awareness : Mobility and Ubiquity

↓

» (Self-)adaptiveness/dynamicity/evolution: defines the ability of a system to *change* in response of external changes

» Dependability: focuses on QoS attributes (performance and all ---abilities)

It impacts all the software life cycle but …

How does the SA contribute to dependability?

Dependability

» *the trustworthiness of a computing system which allows reliance to be justifiably placed on the service it delivers ...*

Dependability includes such attributes as **reliability, availability, safety, security**. (see IFIP WG 10.4 on DEPENDABLE COMPUTING AND FAULT TOLERANCE http://www.dependability.org/wg10.4/)

How do we achieve dependability? All along the software life cycle from *requirements* to *operation* to *maintenance.*

By analysing models, testing code, monitor execution

Dependability and QoS attributes

» *analysing models:* functional and non-functional, several abstraction levels, not a unique model

» *testing code:* various kind of testing e.g. functional-based, operational-based (still models behavioral and stochastic, respectively)

» *monitor execution:* implies monitoring (yet another … model of) the system at run time, it impacts the middleware

» **Focus is on models, from behavioral to stochastic**

Models for SA (examples)

- System dynamic model (LTS, MSC, etc)
- Queuing Network models (+-extended) derived from the dynamic models
- Models analysis, e.g. reacheability for deadlocks etc.
- Performance indices evaluation for QN

SOFTWARE ARCHITECTURES

- Abstractions of real systems: Design stage
- Computations => *Components*
- Abstraction over :
- Interactions => *Connectors*
- ++++ *Static & Dynamic Description* ++++

SOFTWARE ARCHITECTURES

- (***Closed***) ***Software Architectures***: components + connectors
- ***Architectural Styles***: family of similar systems. It provides a vocabulary of components and connector types, and a set of constraints on how they can be combined.
- ***Architectural Patterns***: well-established solutions to architectural problems. It gives description of the elements and relation type together with a set of constraints on how they may be used.

Analysing Evolving Systems

- Systems that *change* structure and/or behaviour
- Change *the four Ws*:
  - Why there is the need to change?
  - What does (not) change ? (only SA changes)
  - When does the change happen?
  - What/Who how is the change managed?

Four Examples

- Synthesis
- Performance
- Chamaleon
- Connect

EVOLUTION 1

SYNTHESIS

Tivoli, Autili, Inverardi

CBSE-Synthesis

Problem: The ability to establish properties on the assembly code by only assuming a relative knowledge of the single components properties.

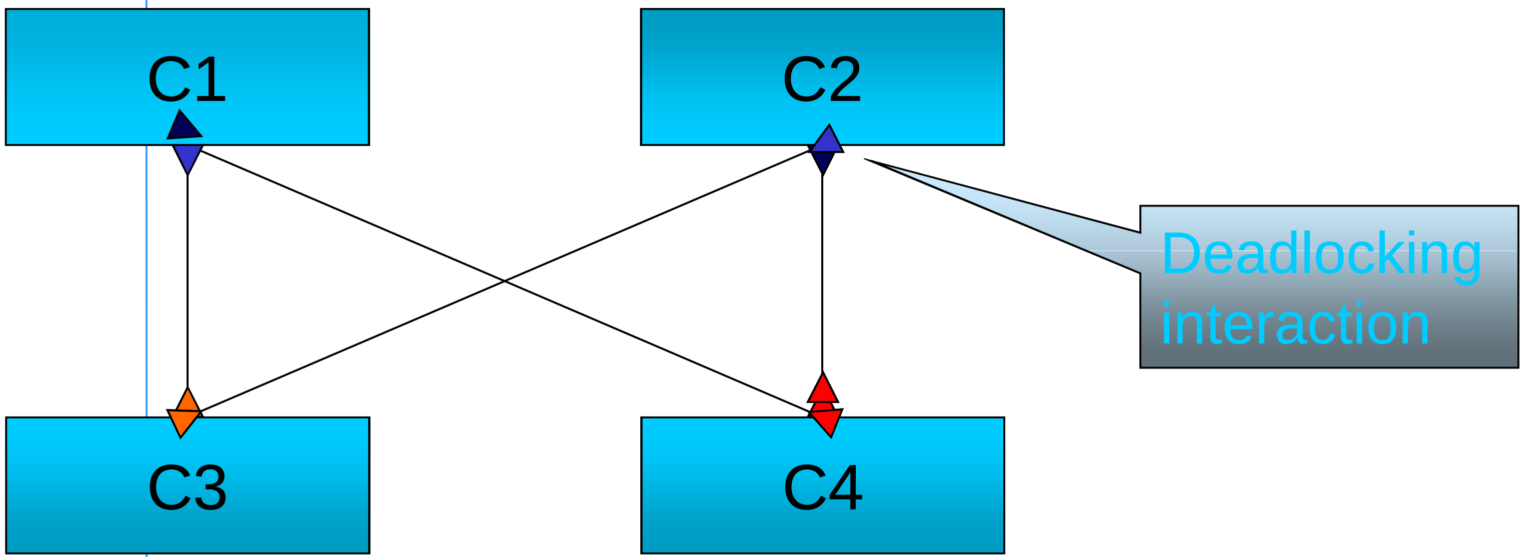
A software architecture represents the reference skeleton used to compose components and let them interact: interactions among components are represented by the notion of software connector.

Goals

» Provide a framework ***to support the development of distributed component-based systems*** out of a set of already implemented heterogeneous components by ensuring the correct functioning of the assembled system at components interaction protocol level.

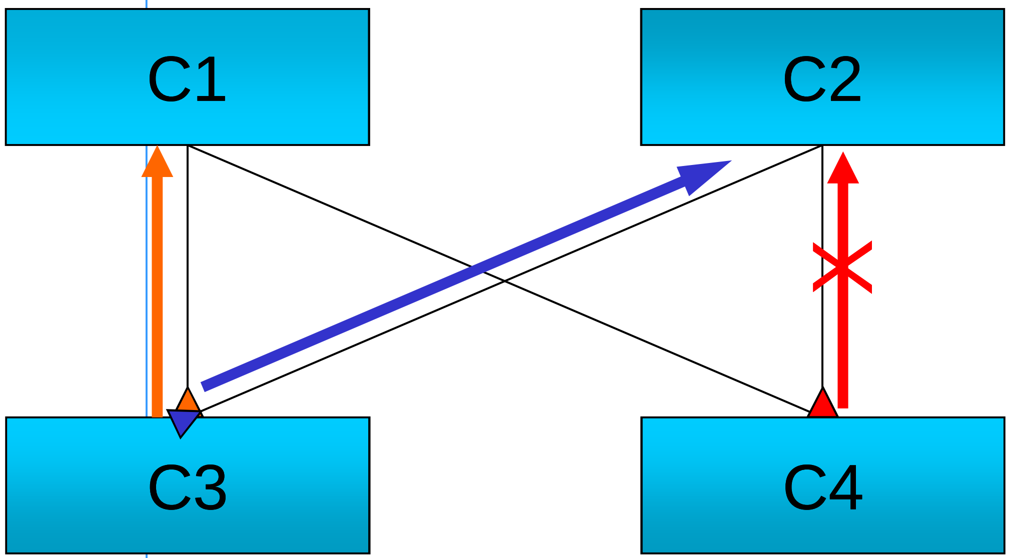
Problem description (1/2)

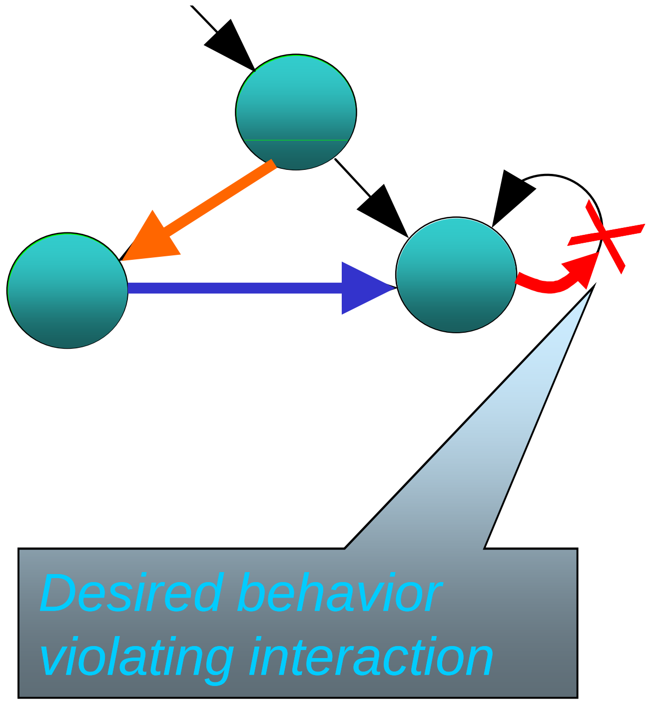
Adaptor-Free Architecture (AFA)



Problem description (2/2)

Adaptor-Free Architecture (AFA)

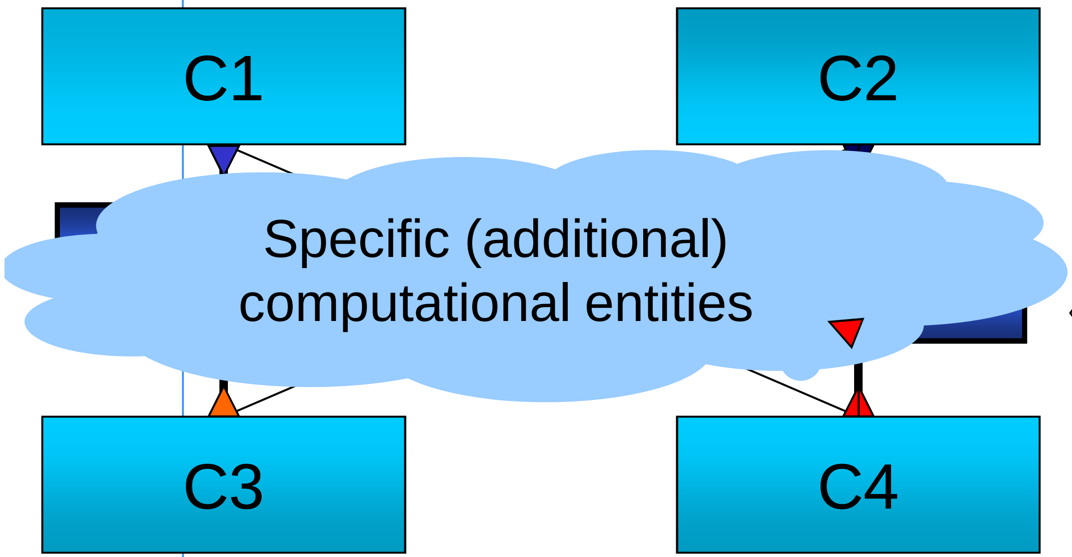




Distributed SYNTHESIS method: first step

Adaptor-Free Architecture (AFA)

Centralized Adaptor Based Architecture (CABA)



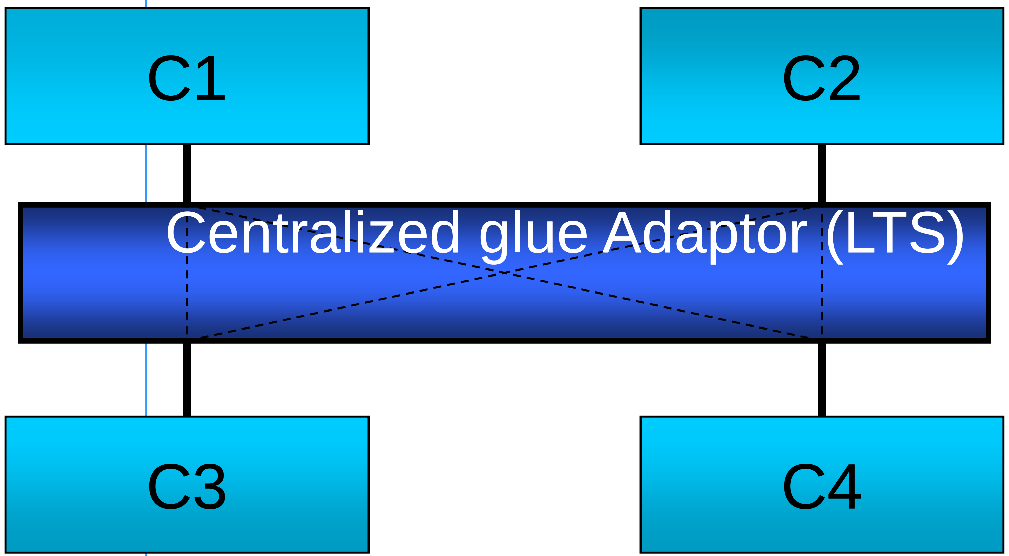
Centralized glue Adaptor models-LTS

»It models a *dummy* router for the components interaction

» Automatically generated by Synthesis

Distributed SYNTHESIS method: second step

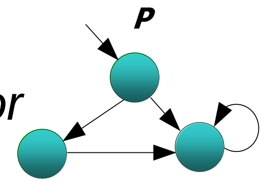
Centralized Adaptor-Based Architecture (CABA)



1) Deadlock-freedom analysis

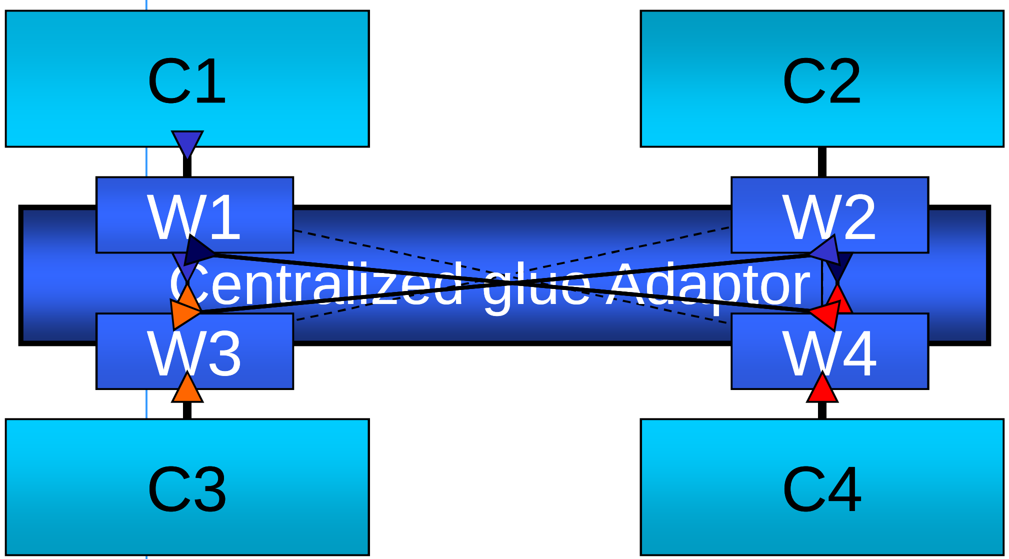
2) Desired behavior analysis

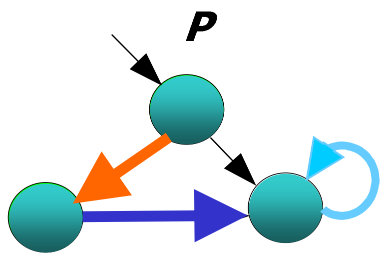
Desired behavior specification: LTS-based notation



SYNTHESIS method overview: second step

Distributed Adaptor-Based Architecture (DABA)





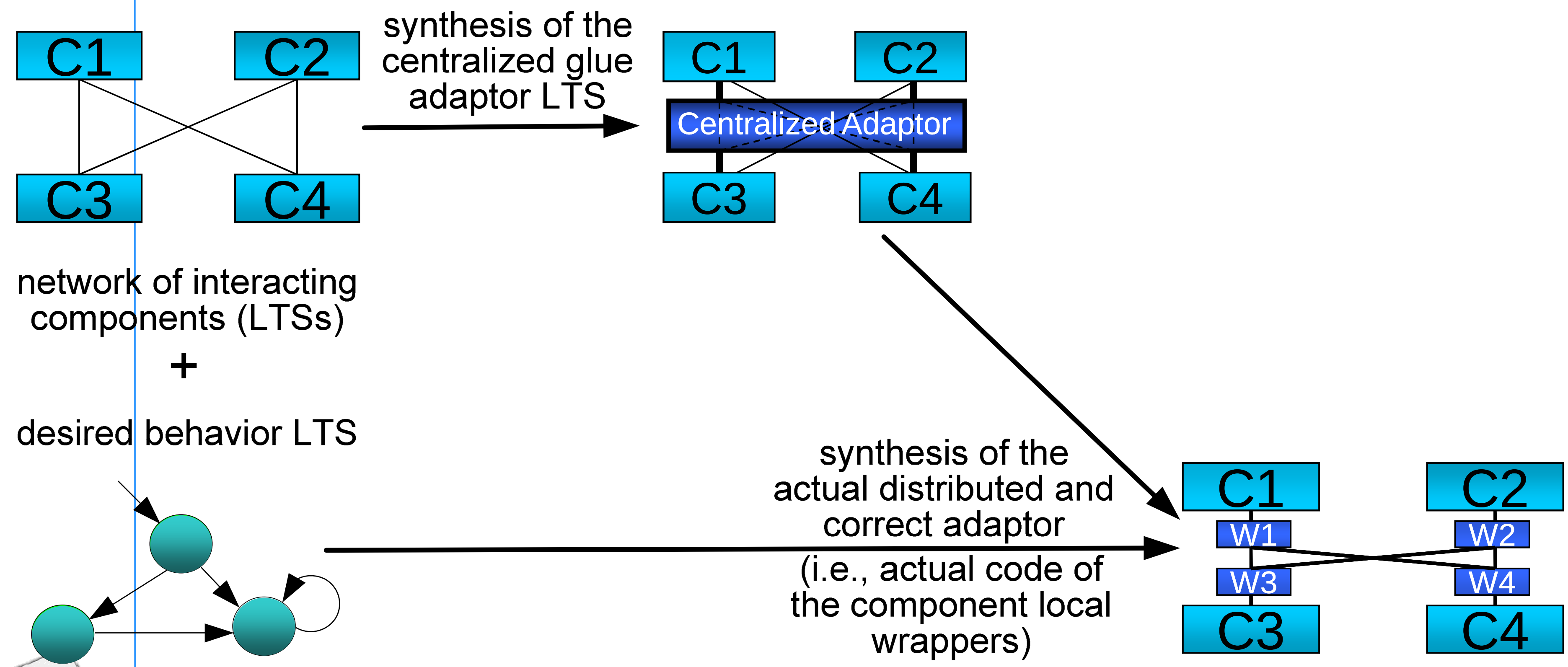
Distributed Adaptor
(*i.e.*, set of wrappers)

» Deadlock free

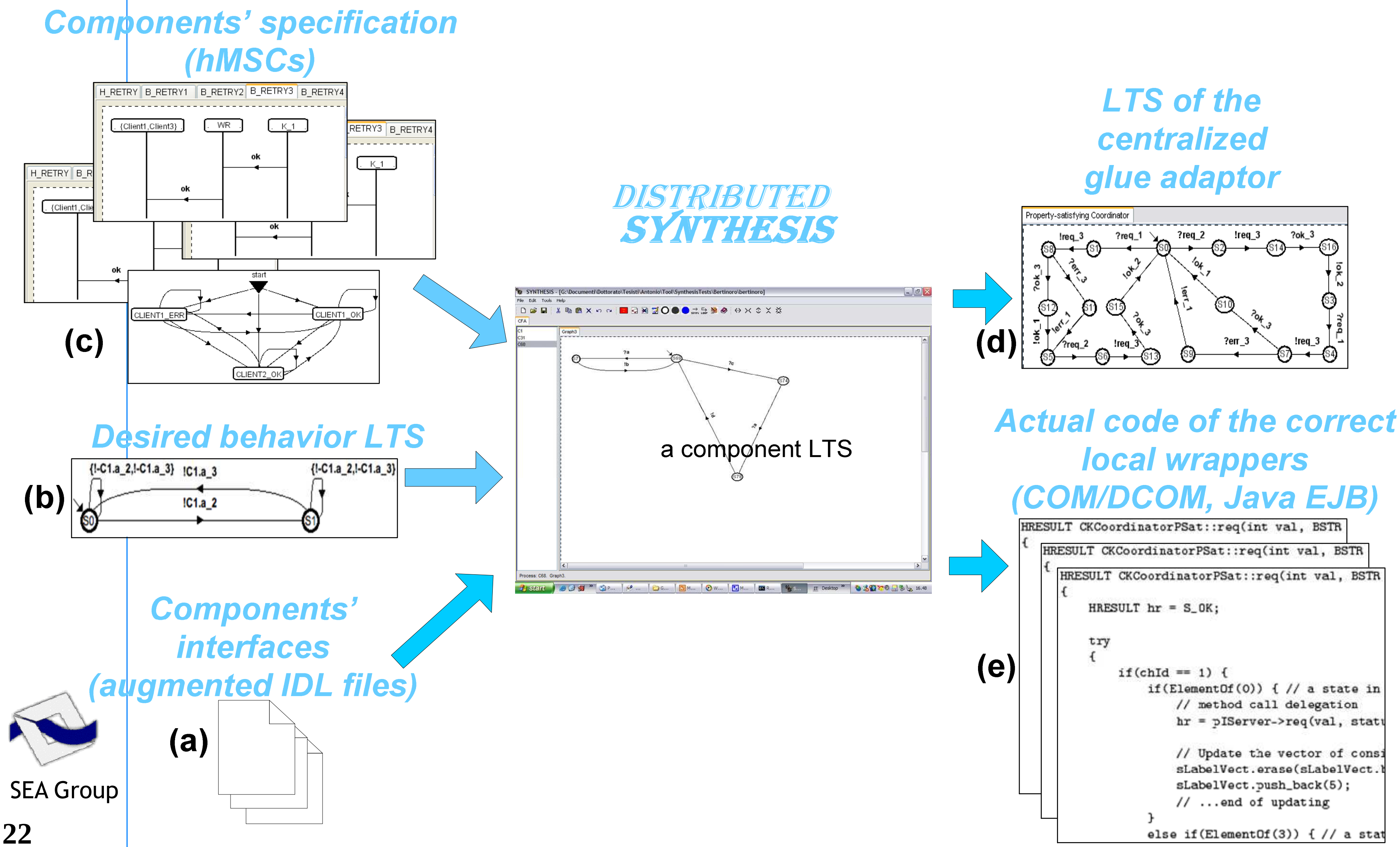
» Desired behavior satisfying

(automatically distributed by Synthesis)

To summarize …



The *Distributed SYNTHESIS* tool



The four Ws: Synthesis

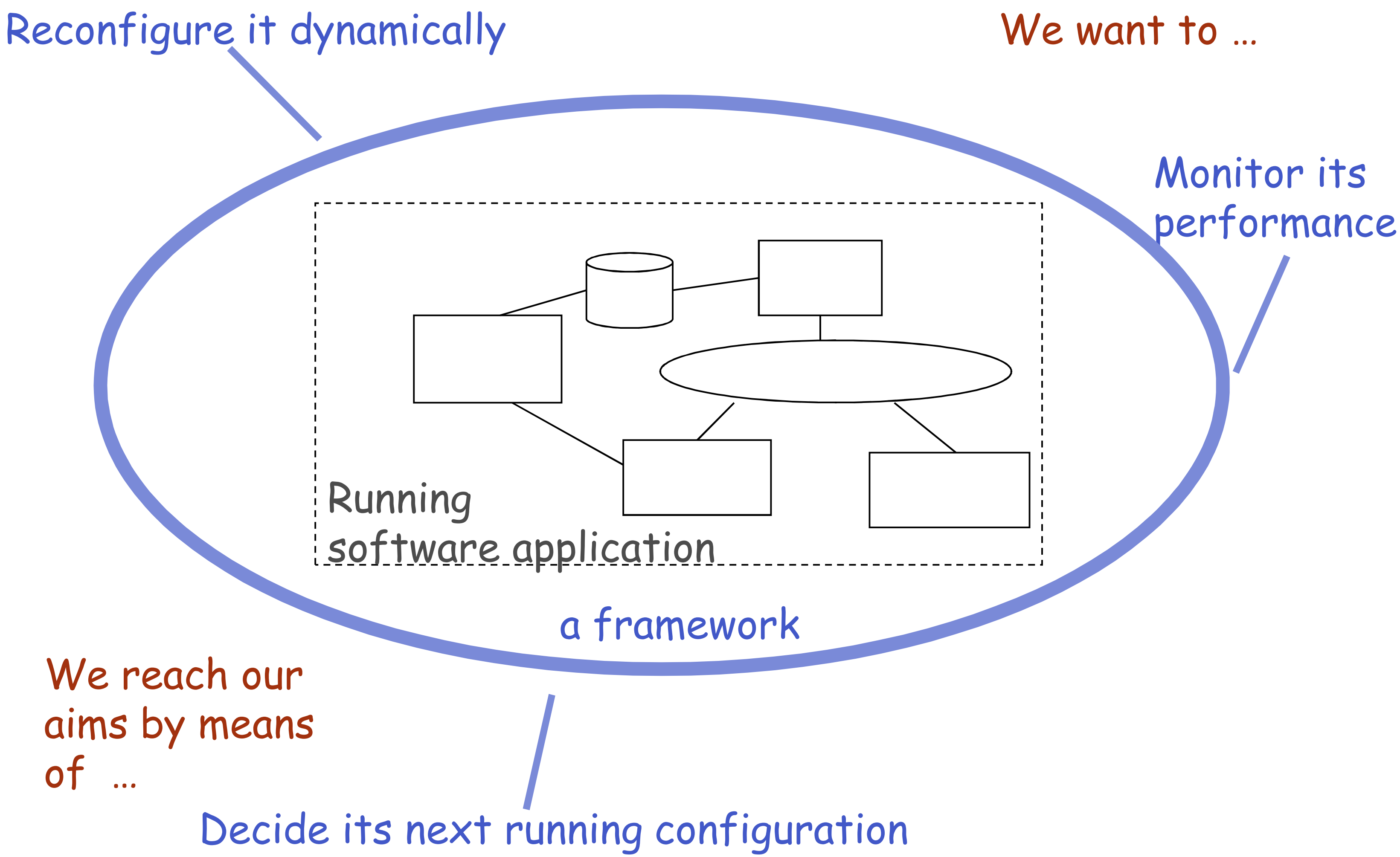
* *the four Ws*:
  - Why there is the need to change?
    > To correct functional behavior. E.G. Avoid deadlock
  - What does (not) change ?
    > The topological structure and the interaction behavior
  - When does the change happen?
    > At Assembly time but also …
  - What/Who how is the change managed?
    > An external entity: Synthesis

EVOLUTION EXAMPLES: 2

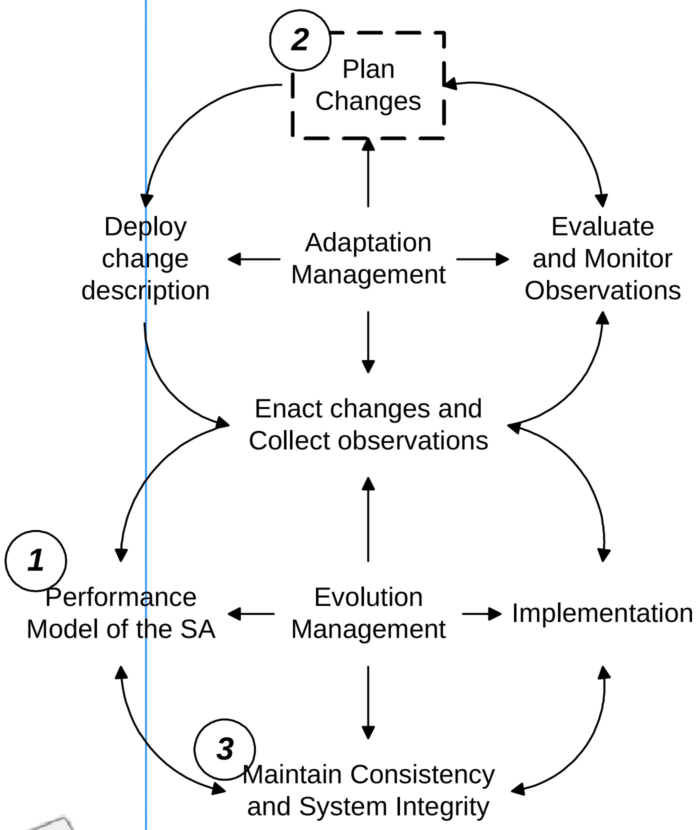
PERFORMANCE

Caporuscio-Di Marco-Inverardi

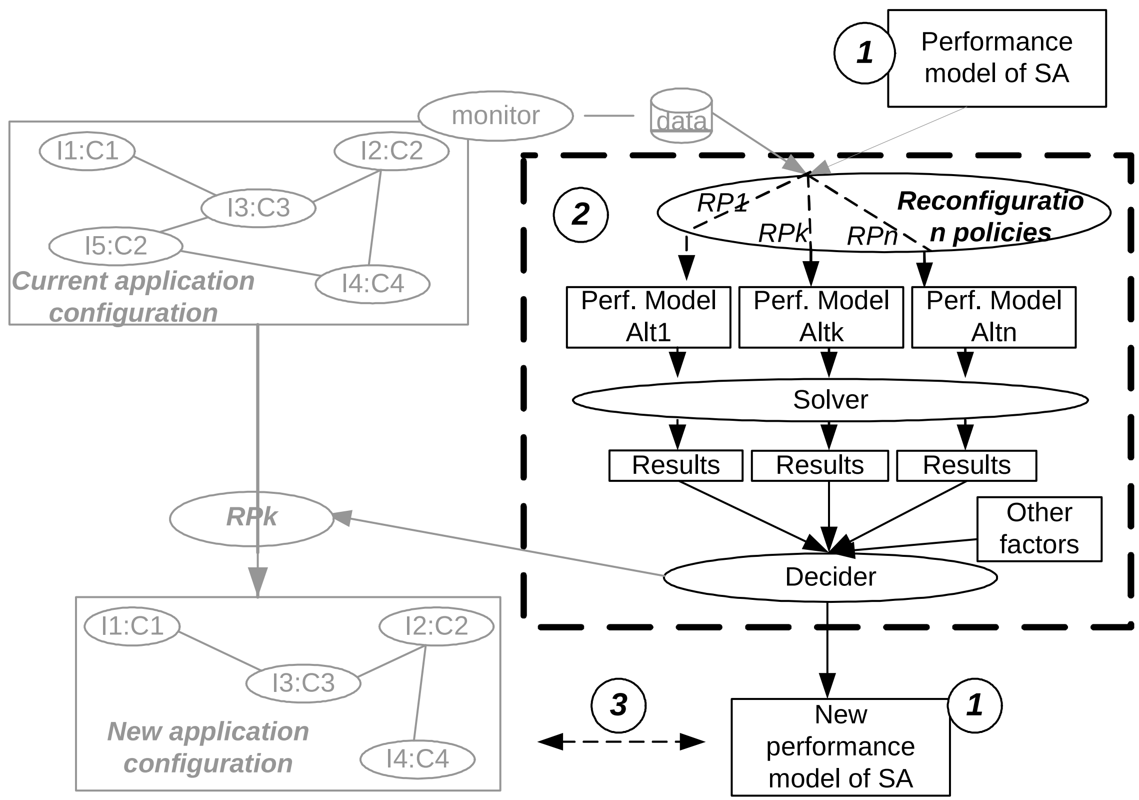
PERFORMANCE : system reconfiguration



The Adaptation process



(a) The Processes



(b) Flow of the activities in an adaptation step

SEA Group

Issues to address

» What is the relevant data to collect? And how to use it?

- Data collected is more fine-grained than the performance model parameters.

» When should we reconfigure the application? Which are the reconfiguration alternatives?

- It depends on the application.

» Models have to be modified and evaluated online (fast solution techniques).

- Which performance model should we use?
- How do we take the decision on the next configuration?
- Different aspects should be considered (security, resources availability,…)

The four Ws: Performance

* *the four Ws*:
  - Why there is the need to change?
    > To correct non- functional behavior. i.e. Adjust Performance
  - What does (not) change ?
    > The topological structure
  - When does the change happen?
    > At run time …
  - What/Who how the change is managed?
    > An external entity: the configuration framework

EVOLUTION EXAMPLES: 3

CHAMELEON

A framework for the development and deployment of adaptable Java applications

Di Benedetto, Mancinelli, Autili, Inverardi

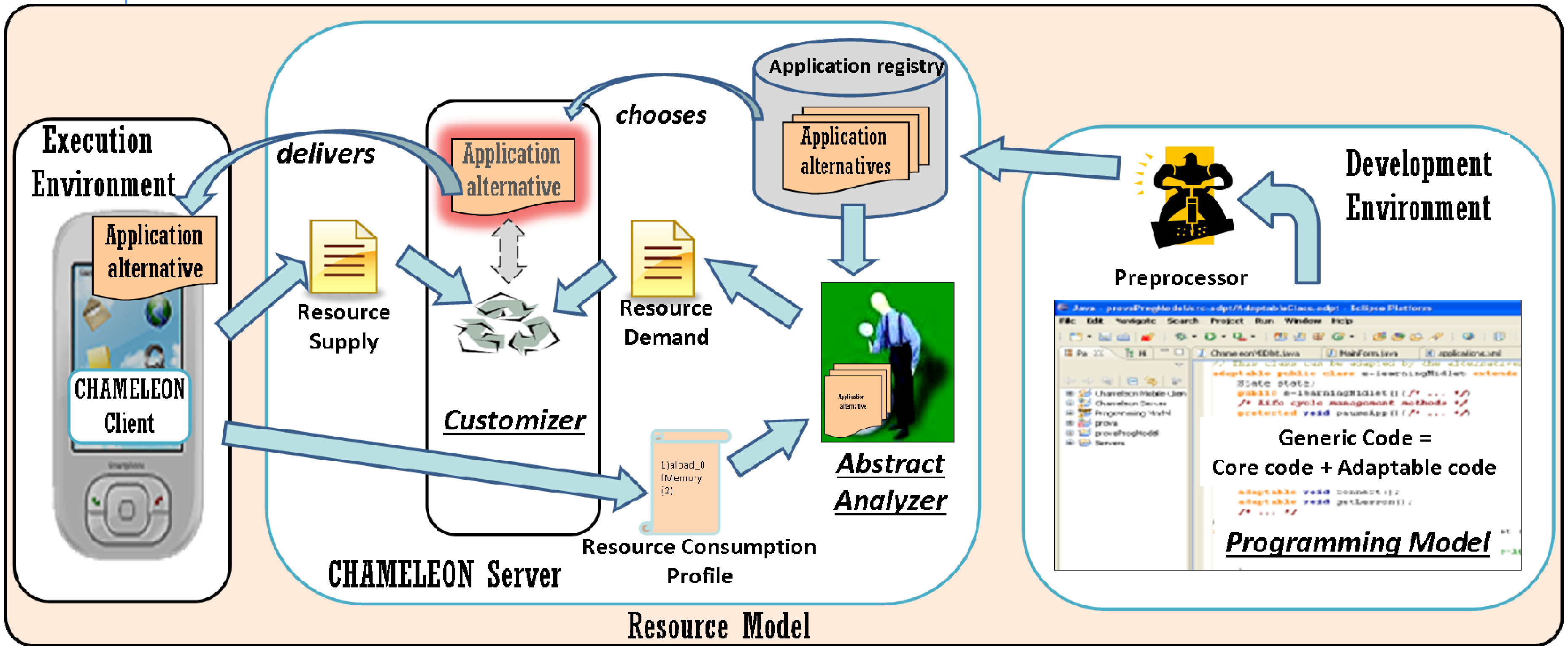
Summary

- 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.**

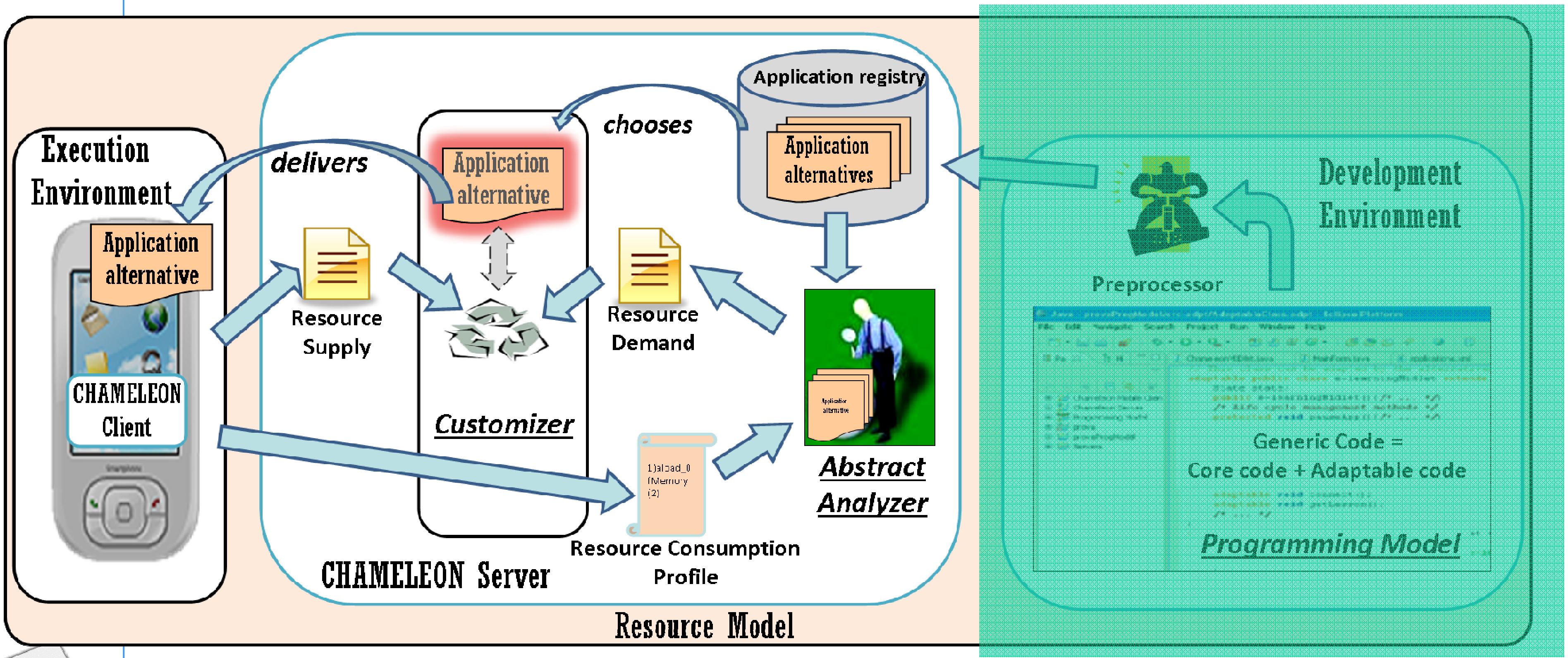
❑ *Resource-aware adaptation*

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



Development Environment



Programming Model

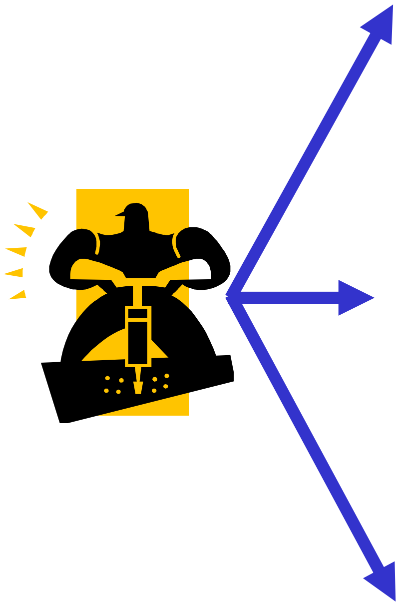
```
adaptable class C {
    adaptable void m1 ( ) ;
    adaptable void m2 ( ) ;
}

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( ) { . . . }
}
```

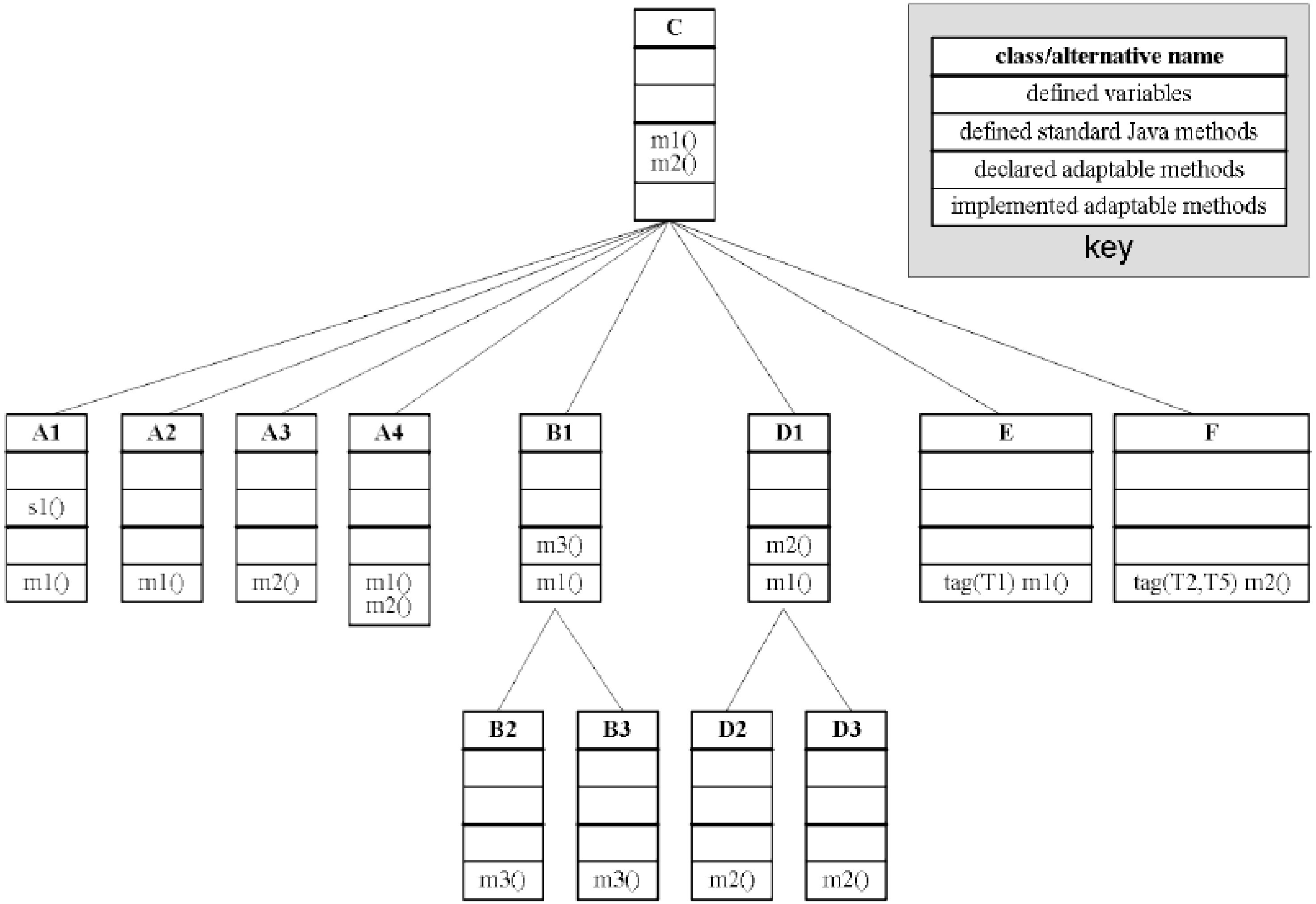


```
class C {
    void m1 ( ) { . . . } // from A2
    void m2 ( ) { . . . } // from A3
}

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

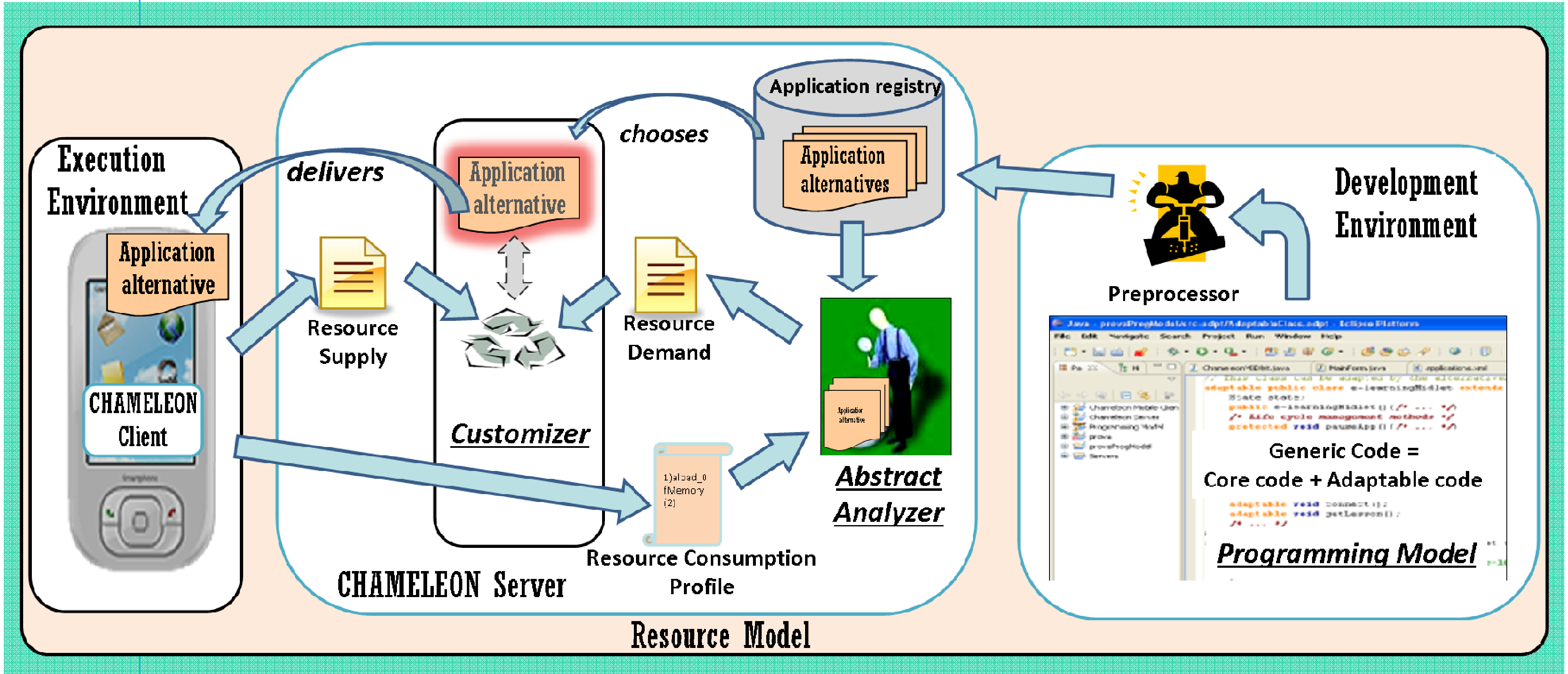


Adaptable Application Preprocessing

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

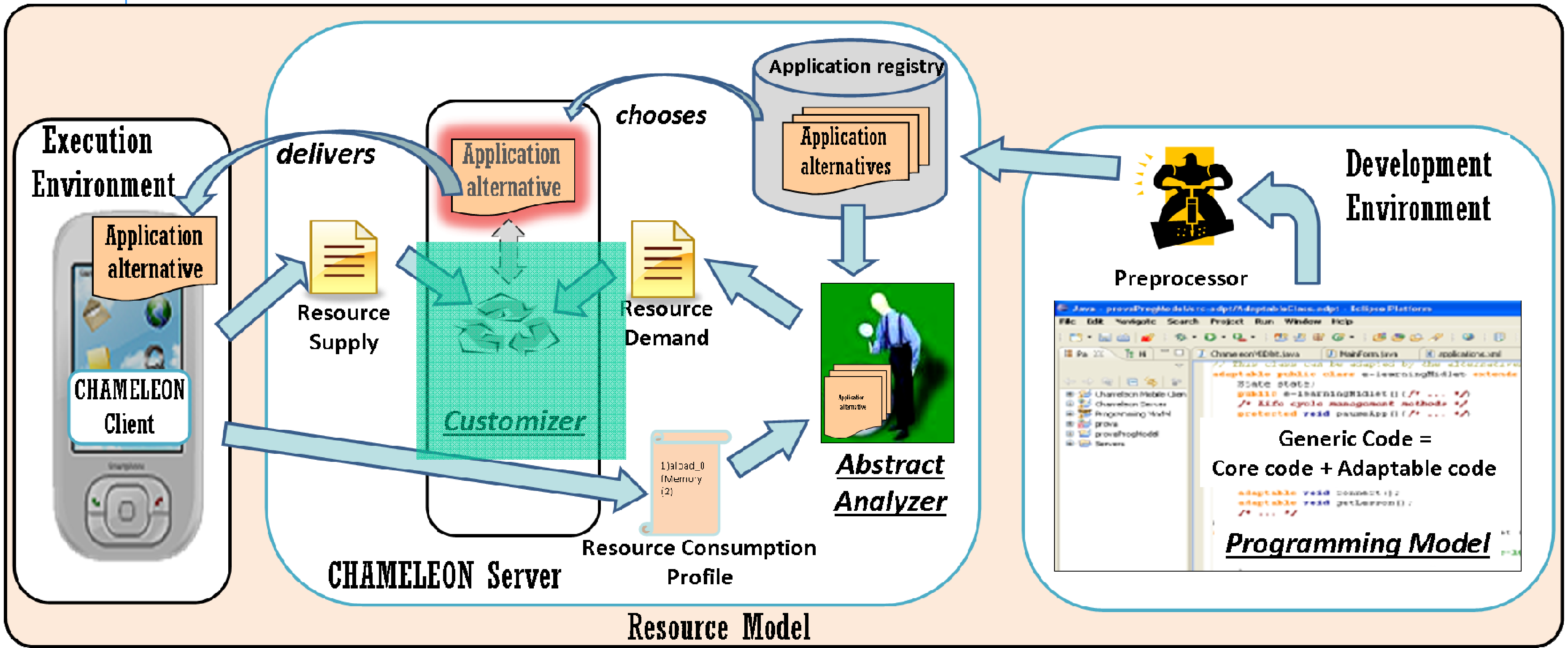
- A resource set *(demand)* P is compatible with a resource set *(supply)* Q ($P \lhd Q$) if:
  1. **(Availability)** For every *resource instance* r(x) ∈ P there exists a *resource instance r(y)* ∈ Q.
  2. **(Wealth)** For every pair of *resource instances* r(x) ∈ P and r(y) ∈ Q, p(x) ≤ p(y).

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

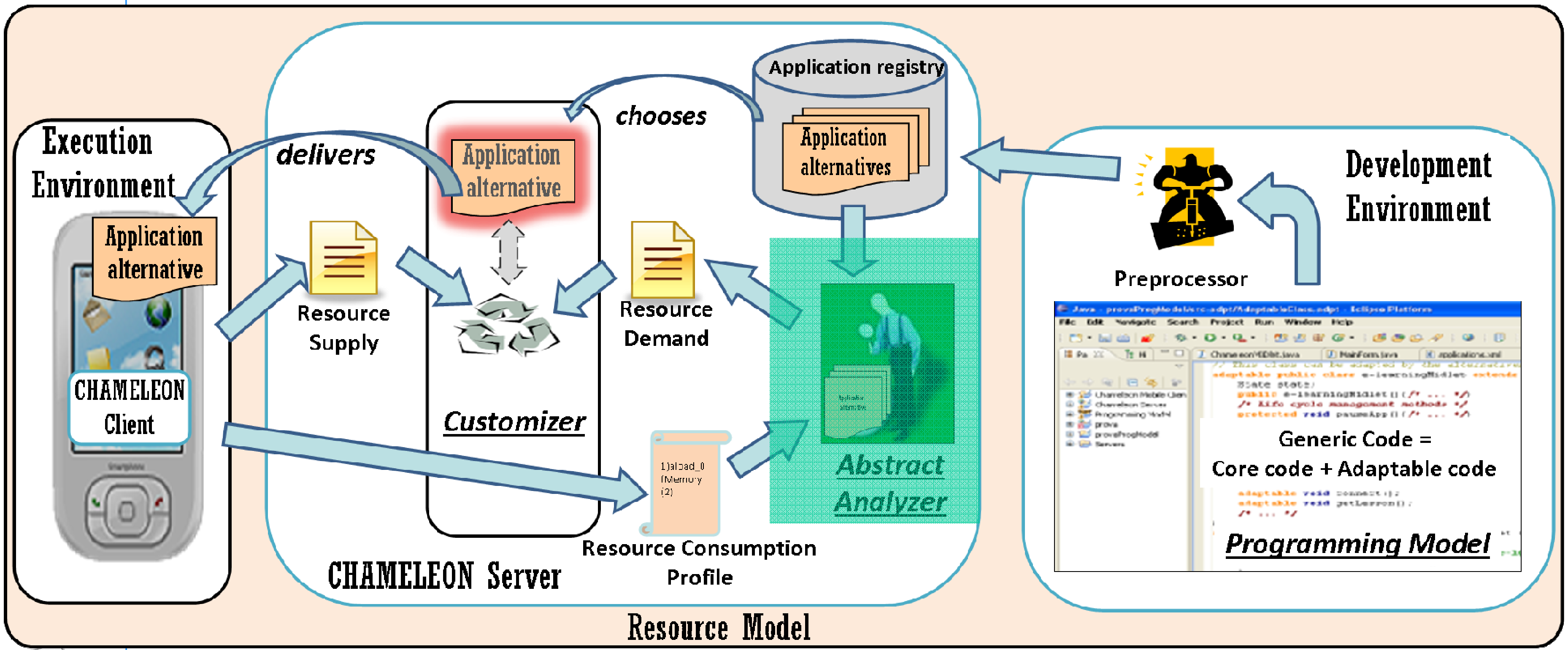
Goodness

- 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 is (e.g., Thread)

Customizer


Execution Environment
Application alternative
CHAMELEON Client
delivers
Application alternative
chooses
Application registry
Application alternatives
Resource Supply
Resource Demand
Customizer
Abstract Analyzer
Resource Consumption Profile
CHAMELEON Server
Preprocessor
Development Environment
Generic Code =
Core code + Adaptable code
Programming Model
Resource Model

Abstract Analyzer



Abstract Analyzer

- Interpreter that abstracts a standard JVM
- Statically analyzes an application inspecting all the possible computation paths and determines its Resource Demand (resources required to correctly execute the application)
- 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)}

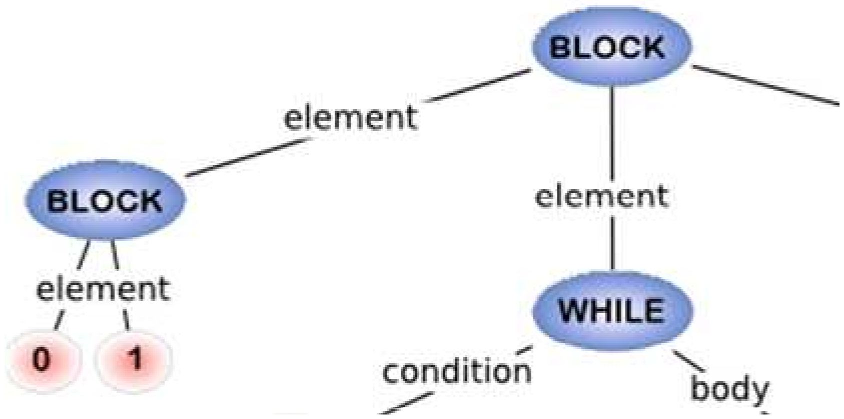
- Can be created on the basis of:
  - experimental results based on benchmarking tools
  - Information provided by device manufacturers, network sensors …

- Always exists a default Resource Consumption Profile

- The more are accurate, the more the analysis is precise

Fall-Back Leaf Rule

$$\frac{IsLeaf(n) \quad Label(n) = instr \quad instr\ !Like("invoke*") \quad !IsAnnotation(n) \quad r = b(instr) \quad C = \{r\}}{\langle e, b, M, n \rangle \rightarrow_{AA} C}$$



0: iconst_0
1: istore_1
:

0: iconst_0	→ C={ {CPU(1), Energy(1)} }
1: istore_1	→ C={ {CPU(2), Energy(1)} }
:	

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

IF_ELSE rule

$$
\frac{
\begin{array}{c}
Label(n) = \texttt{IF\_ELSE} \\
Children(n, \texttt{condition}) = \{n_{cond}\} \\
Children(n, \texttt{trueBranch}) = \{n_{true}\} \\
Children(n, \texttt{falseBranch}) = \{n_{false}\} \\
\langle e, b, M, n_{cond} \rangle \xrightarrow{*}_{AA} C_{cond} \\
\langle e, b, M, n_{true} \rangle \xrightarrow{*}_{AA} C_{true} \\
\langle e, b, M, n_{false} \rangle \xrightarrow{*}_{AA} C_{false} \\
C_1 = C_{cond} \oplus C_{true} \\
C_2 = C_{cond} \oplus C_{false} \\
C = C_1 \cup C_2
\end{array}
}{
\langle e, b, M, n \rangle \xrightarrow{*}_{AA} C
}
$$

The four Ws: Chamaleon

* *the four Ws*:
  - Why there is the need to change?
    > To match resource supply of the execution context
  - What does (not) change ?
    > The component non functional behavior
  - When does the change happen?
    > At deployment time but also …
  - What/Who how is the change managed?
    > An external entity: Chamaleon

A completely open scenario: CONNECT

- Ubiquitous systems: components travel around willing to communicate with only their own knowledge
- Exploit the process: discover-learn-mediate-communicate
- No global SA assumed
- The SA in terms of components and connectors results from the completion of the process
- and dependability … ? It is built in the composition e.g. embedded in the connectors.

CONNECT

Emergent Connectors for Eternal Software Intensive Networked Systems

7FP-Call 3 - ICT-2007

http://connect-forever.eu/

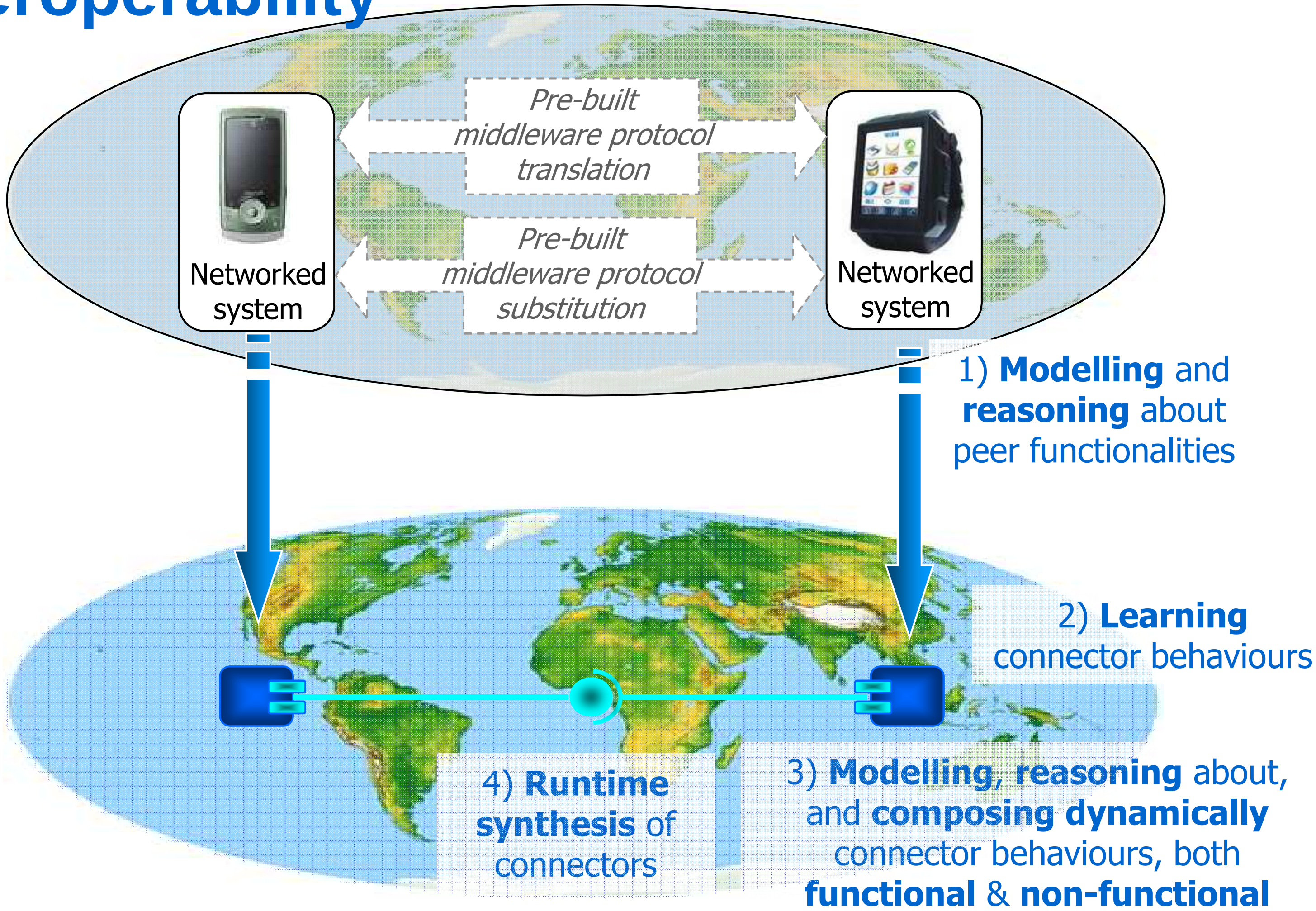
A run-time model-centric approach to eternal interoperability

From Non-CONNECTed

Pre-built connectors at syntactic level

To CONNECTed

Emergent connectors at semantic level for eternal connectivity



Synthesis of application-layer conversation protocols

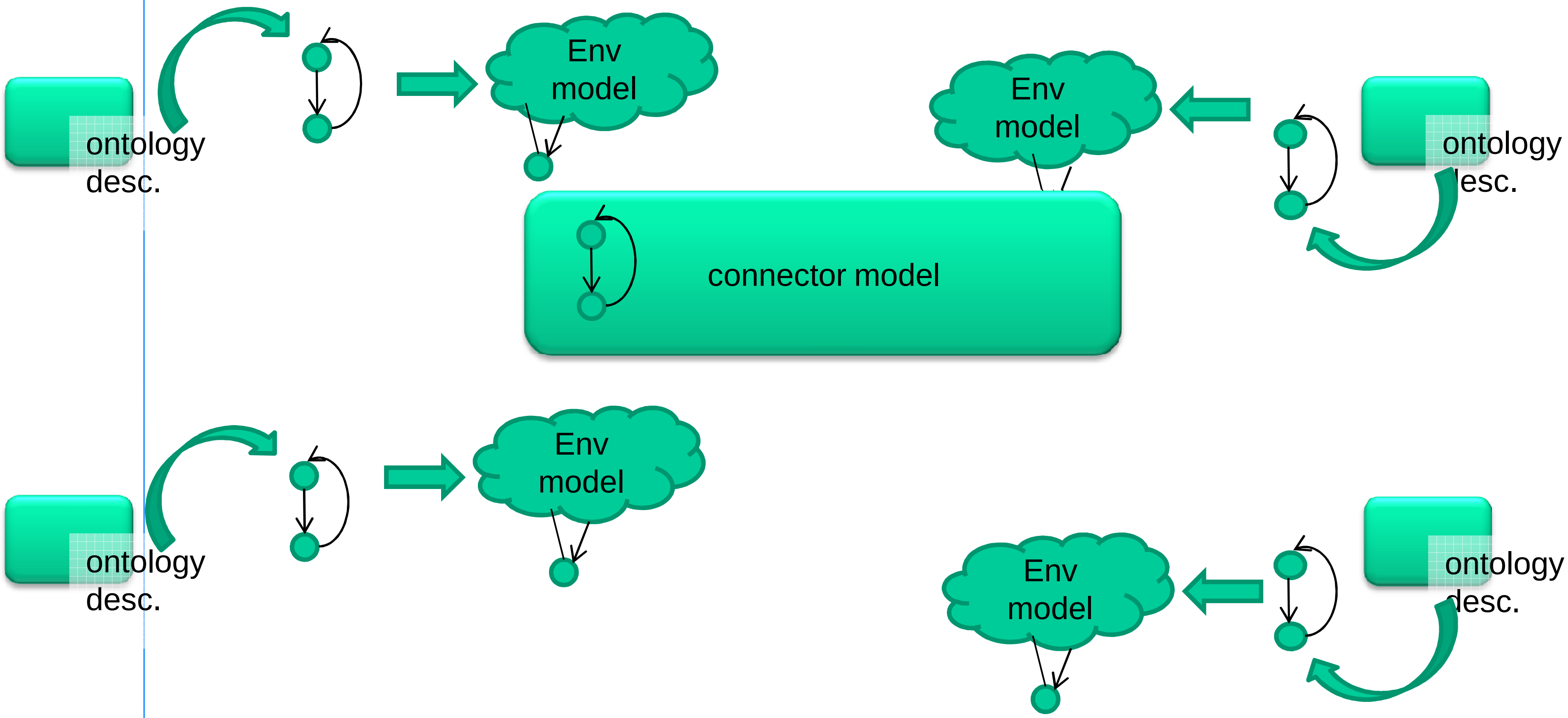
» To support the automated construction of application-layer connector models

- 1: identifying the conditions on the networked applications interaction and composition that enable run-time connector synthesis
  - > SA and connector patterns
- 2: the synthesis process is seen as a behavioral model unification process
  - > ontologies
  - > modeling notations
  - > unifying know and unknown information

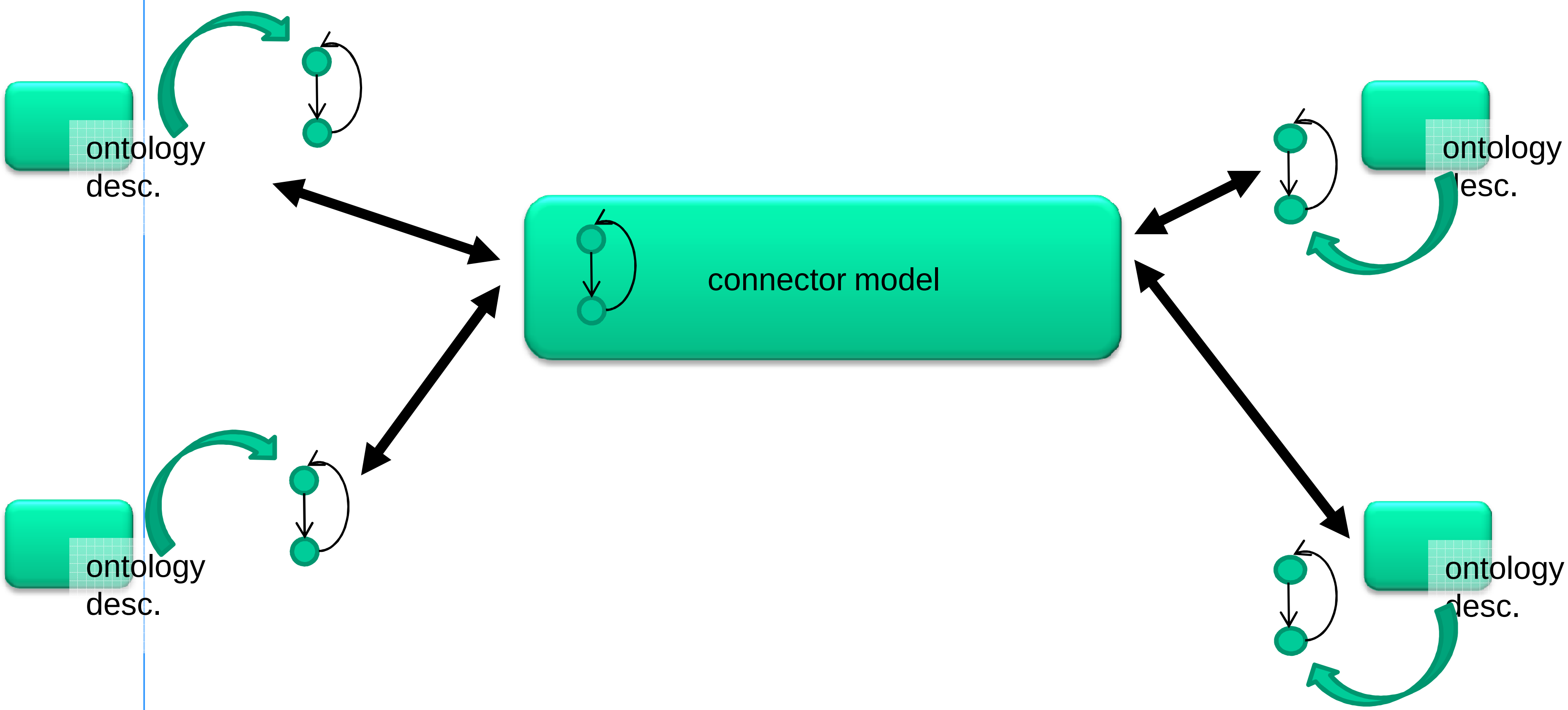
» The challenge

- compositionality and evolution

synthesis process steps



synthesis process steps

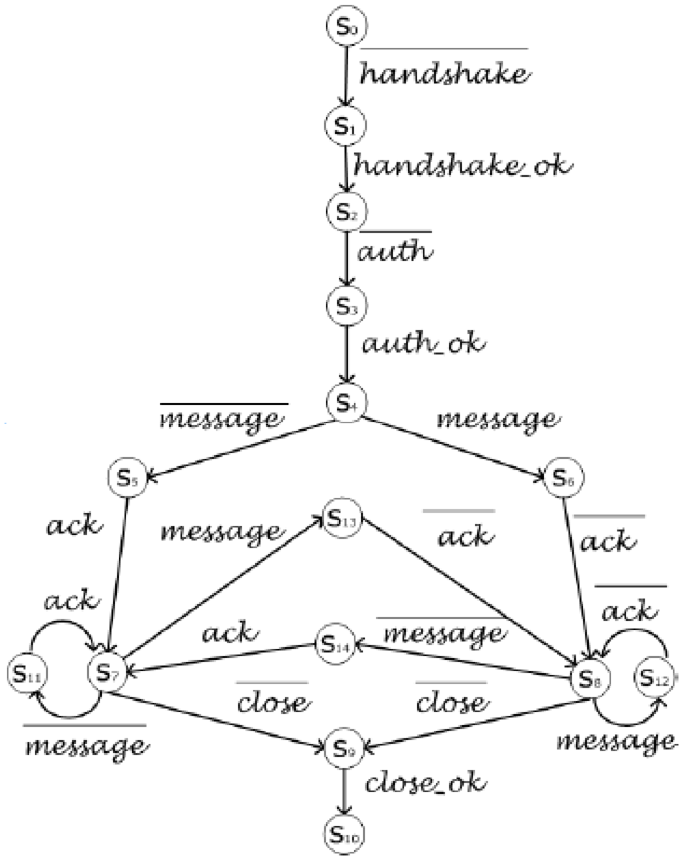


Foundations for the automated mediation of heterogeneous protocols

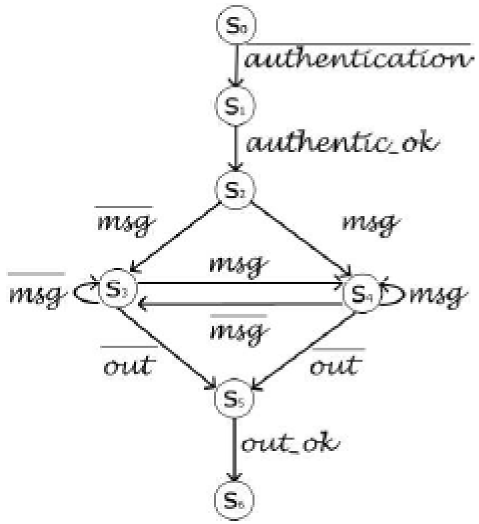
- Modeling notation used to abstract the behavior of the protocols to be bridged
  - finite state machines
- Matching relationship between the protocol models
  - necessary (but non-sufficient) conditions for protocol interoperability
    - > e.g., "*sharing the same intent*"
  - data and functional mediations are assumed to be provided
- Mapping algorithm for the matching protocol models
  - sufficient (and "most permissive") conditions for protocol interoperability
    - > e.g., *"talking, at least partly, a common language"*
  - a concrete mediator as final output

The instant messaging example

do they "*share the same intent*"?



(a) Windows Messenger protocol



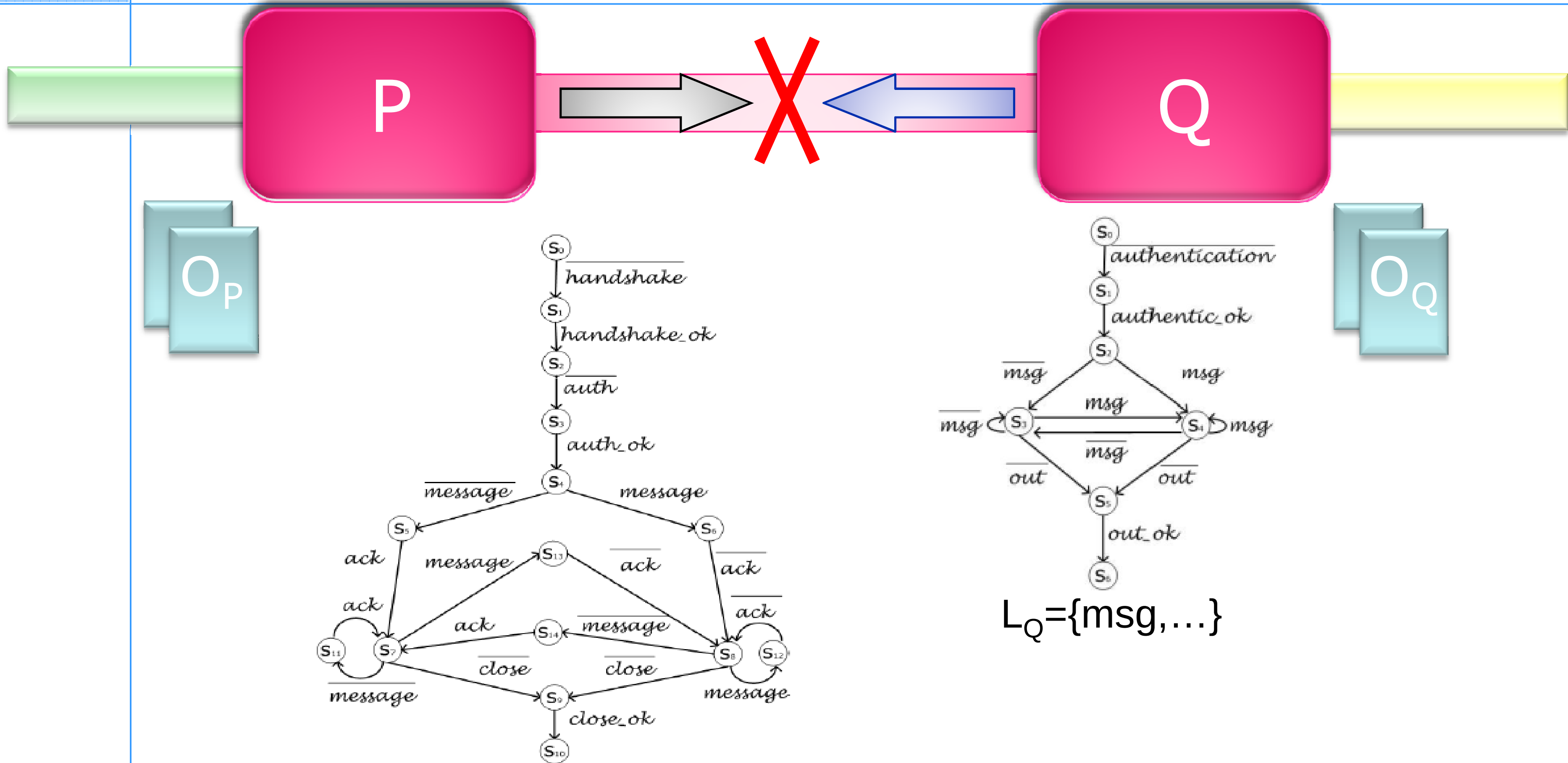
(b) Jabber protocol

Application Level (AL) Interoperability

Assumptions:

- » Two applications with known interaction protocols, i.e. visible behavior
- » Two known ontologies + ontology mapping
- » A specification of what is the purpose of the conversation (initial and final states)
  - Notion of *coordination policies*
- » Protocol compatibility expressed via *equivalence on the coordination policies*

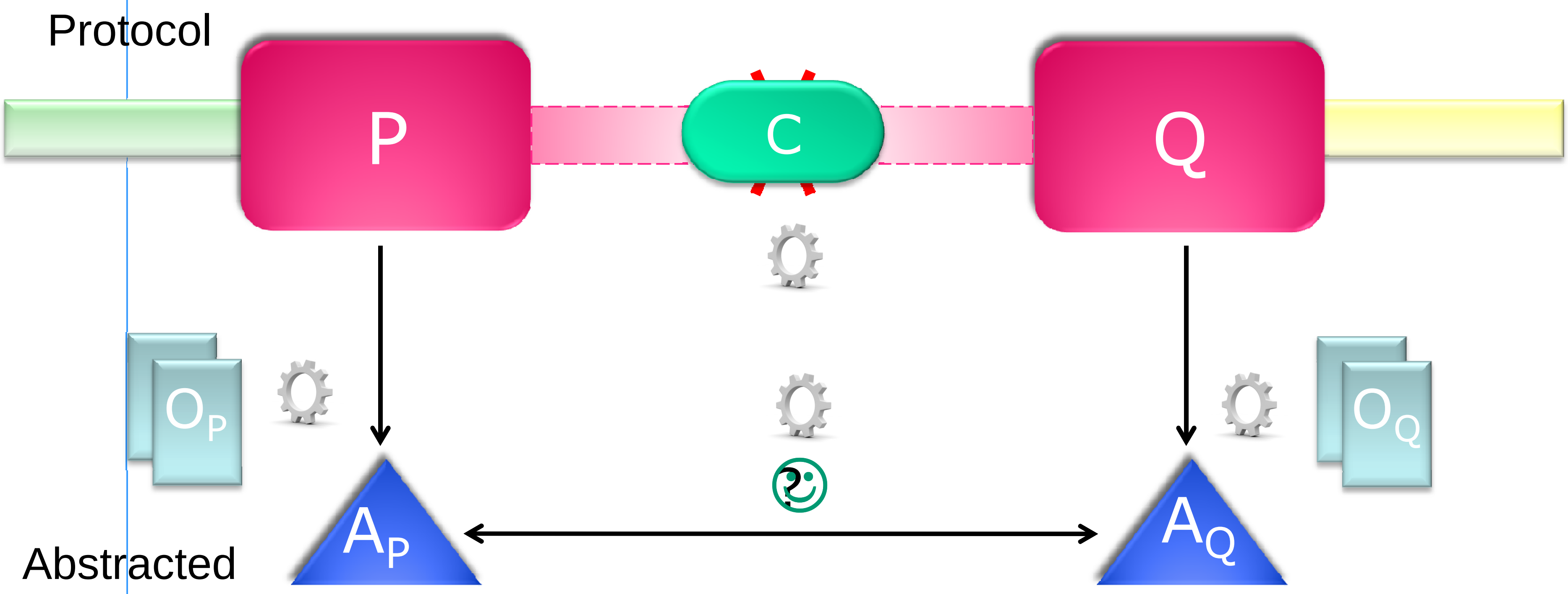
Interoperability problem: An example



L_P={message,ack,…}

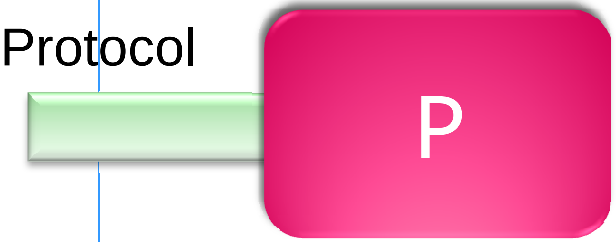
SEA Group

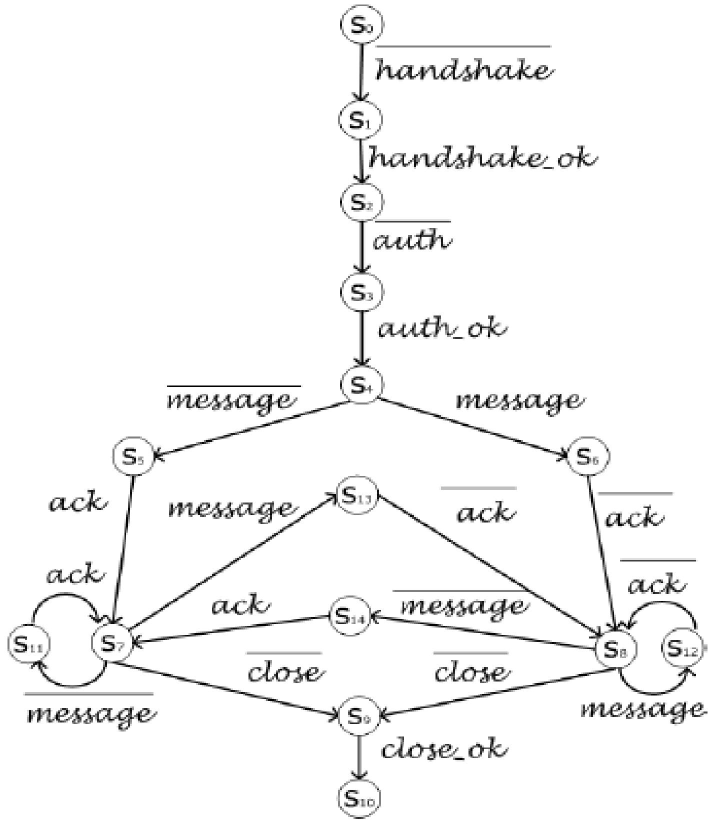
Interoperability problem: The proposed solution



Formalization of the solution (1/4)

Protocol



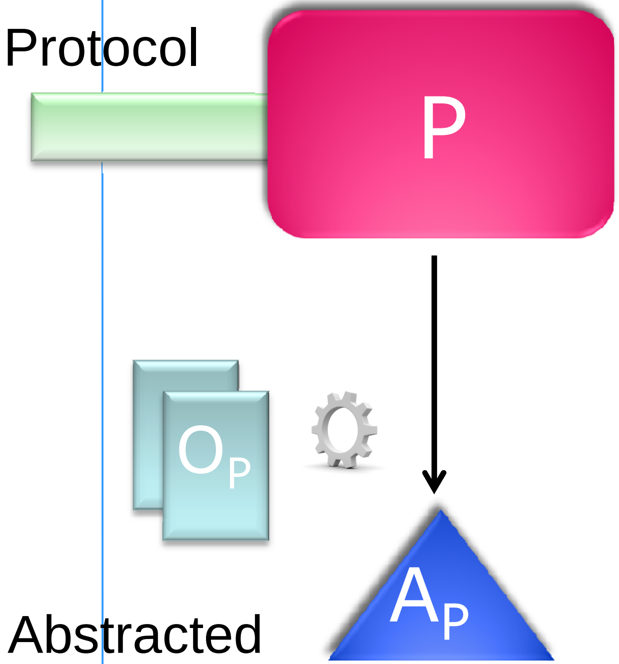


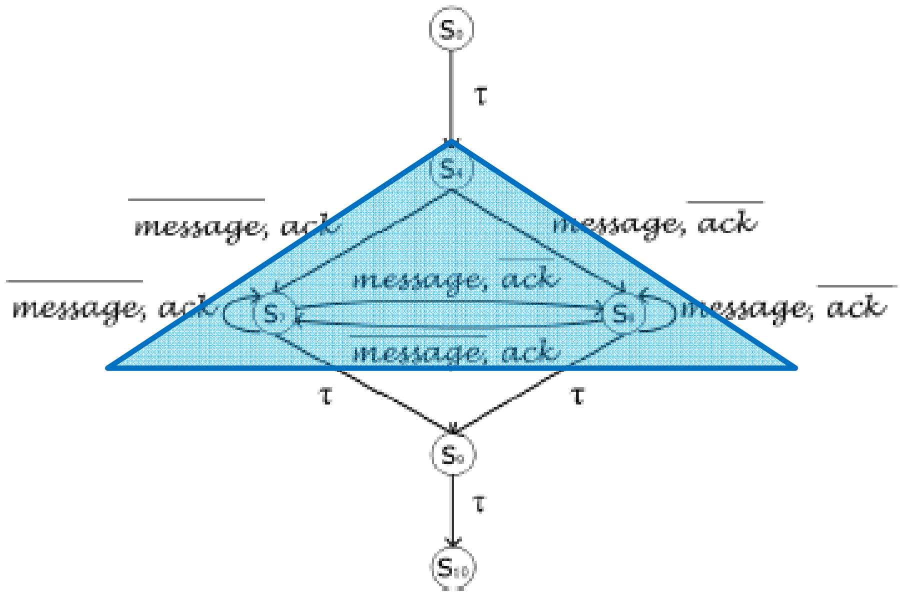
(a) Windows Messenger protocol

Formalization of the solution (2/4)

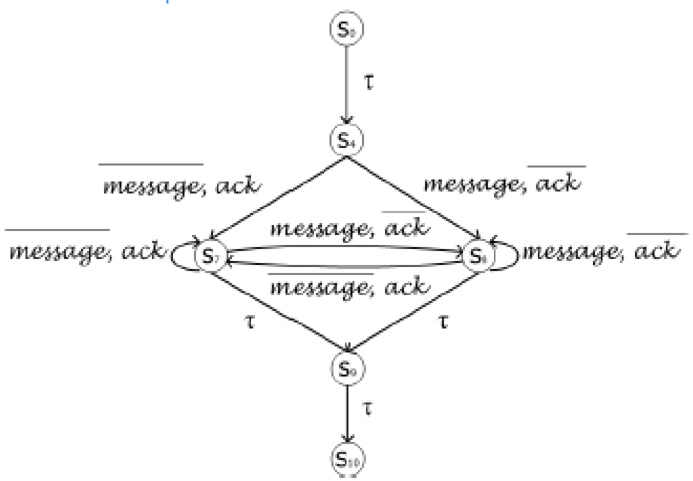
To build the abstracted…

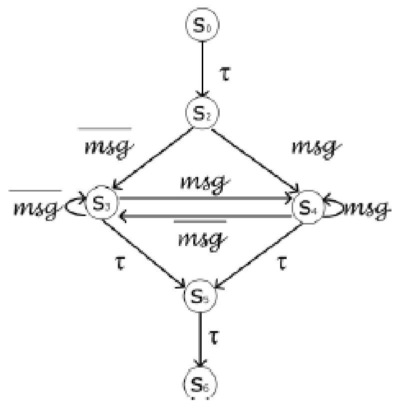
"message, ack" <-> "msg"



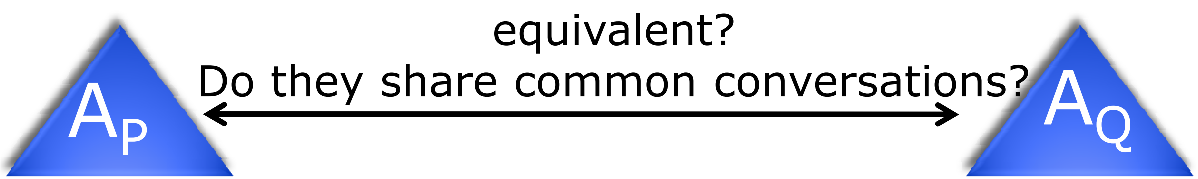


Formalization of the solution (3/4)



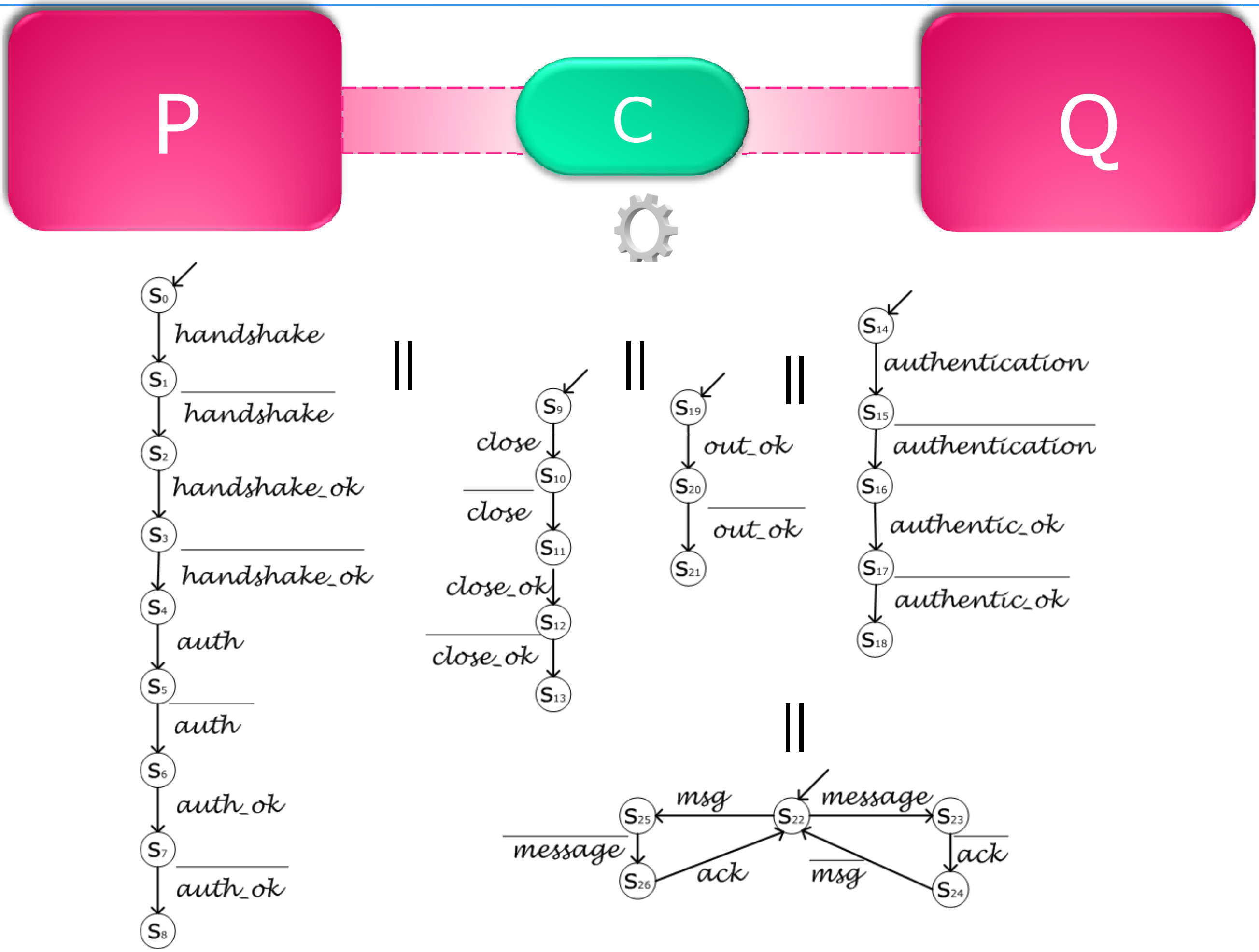


Abstracted



Formalization of the solution (4/4)

Protocol



Discussion on the mismatches coverage

- Extra send mismatch
- Extra receive mismatch
- One send - many receive mismatches
- Many send - one receive mismatch
- Signature mismatch
- Ordering mismatch

✓ Mismatch coverage: all the 6 + combinations (e.g., mismatch 5 combined with the remaining mismatches)

The four Ws: Connect

* *the four Ws*:

- Why there is the need to change?
  - > To allow communication between incompatible protocols
- What does (not) change ?
  - > The overall interaction behavior and the architecture
- When does the change happen?
  - > At run time
- What/Who how is the change managed?
  - > An external entity: Connect enablers

Summarizing

- Synthesis: fixed SA with connector(s) allows the *correct* assembly of component-based systems
- PFM: fixed SA structure – preplanned re-configurations- choice of the *right* one at run time
- Chamaleon: fixed SA structure – arbitrary re-configurations depending on the adaptation alternatives – choice of the *right* one at deployment time
- Connect: Fixed SA pattern, a.k.a. Mediator, *correctly* synthesized on the fly at run time.

Software Architecture and dependability

- For closed systems allows for predictive analysis: from the SA dependability properties are *deduced*
- For open systems the SA may represent the *invariant* with respect to the applications changes or it my be *induced* by the actual system components
- Depending on the architectural change different level of dependability can be assured by pre-preparing the models and the verification strategies
- SA allows for implementing *reusable* verification strategies.

References

P. Pelliccione, P. Inverardi, H. Muccini: **CHARMY: A Framework for Designing and Verifying Architectural Specifications.** IEEE Trans. Software Eng. 35(3): 325-346 (2009)

P. Inverardi, Massimo Tivoli: **The Future of Software: Adaptation and Dependability**. ISSSE 2008: 1-31

Massimo Tivoli, Paola Inverardi: **Failure-free coordinators synthesis for component-based architectures.** Sci. Comput. Program. 71(3): 181-212 (2008)

Marco Autili, Paola Inverardi, Alfredo Navarra, Massimo Tivoli: **SYNTHESIS: A Tool for Automatically Assembling Correct and Distributed Component-Based Systems**. ICSE 2007: 784-787

Marco Autili, Leonardo Mostarda, Alfredo Navarra, Massimo Tivoli: **Synthesis of decentralized and concurrent adaptors for correctly assembling distributed component-based systems.** Journal of Systems and Software 81(12): 2210-2236 (2008)

Mauro Caporuscio, Antinisca Di Marco, Paola Inverardi **Model-Based System Reconfiguration for Dynamic Performance Management**, Elsevier Journal of Systems and Software JSS, 80(4): 455-473 (2007).

M. Autili, P. D. Benedetto, and P. Inverardi. **Context-aware adaptive services: The plastic approach.** In Proc. of Fundamental Approaches to Software Enginneering (FASE'09), volume 5503 of LNCS, pages 124-139. Springer, 2009.

M. Autili, P. Di Benedetto, P. Inverardi, D. A. Tamburri. **Towards self-evolving context-aware services. In Proc. of Context-aware Adaptation Mechanisms for Pervasive and Ubiquitous Services** DisCoTec'08, volume 11, 2008. http://eceasst.cs.tu berlin.de/index.php/eceasst/issue/view/18.

M. Autili, P. di Benedetto,, P. Inverardi **A Programming Model for Adaptable Java Applications**, to appear in PPPJ 2010: 8th International Conference on the Principles and Practice of Programming in Java

R. Spalazzese, P. Inverardi, and V. Issarny. **Towards a formalization of mediating connectors for on the fly interoperability**. In Proceedings WICSA/ECSA 2009, pages 345-348, 2009.

R. Spalazzese, P. Inverardi**, Mediating Connector Patterns for Components Interoperability.** To appear Proc. ECSA 2010, LNCS**.**