

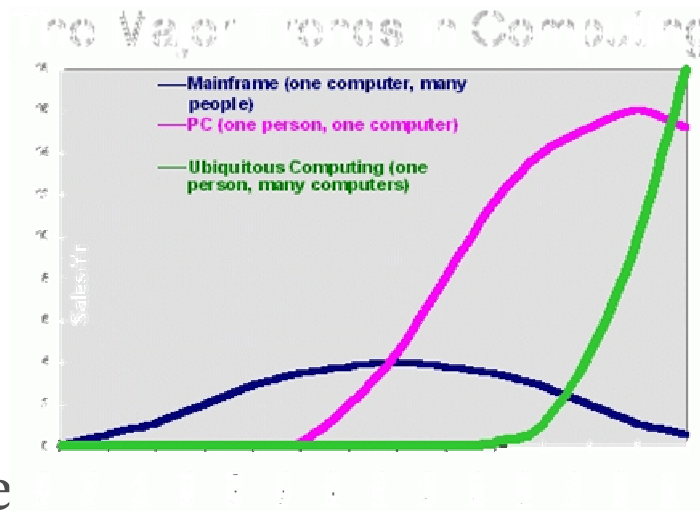
Run time models in adaptive service infrastructure

Paola Inverardi

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

Setting the context

- » **Software Services:** Application software → the software an end-user interacts with → also middleware
- » **Future:** Ubiquitous computing → *one person many computers* ratio (<http://sandbox.xerox.com/ubicomp/>)



- » An instance



The Problem/Question/Issue: *Softure*

- » **Softure**: *Software* in the near ubiquitous future
- » What are the challenges for software in this new era?
 - Copying with variability of contexts: Context Awareness
 - Self-adaptiveness/dynamicity/evolution
 - Dependability
 -
- » Are these challenges **new** in the Software domain?



A Software Engineering Perspective

The *process view*: the focus is on the set of activities that characterize the production and the operation of a software service/system

Growing **Complexity** of Software has exacerbated the *dichotomy* *development/static* time vs *execution/dynamic* time



At present the focus is still on **development time** although the SOA paradigm has undermined it.

Does SOA bring something new in this picture?



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** → **open** systems



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*
- » Our bias is on SA models (not only ...)



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

How do we build **dependable adaptive** systems?



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



Mobile and ubiquitous systems

- » Open systems accounting for
 - changes in the **context**
 - user needs
- » **Context**
 - network context conditions
 - execution environment characteristics
- » **User needs as dependability requirements**
 - availability, reliability, safety, and security
 - e.g., availability as performance indexes
 - > responsiveness, throughput, service utilization



Models at run time: Three experiences

- » System Re-configuration : Siena reconfiguration
- » Service adaptation: The Plastic approach
- » Mediator synthesis: The Connect project



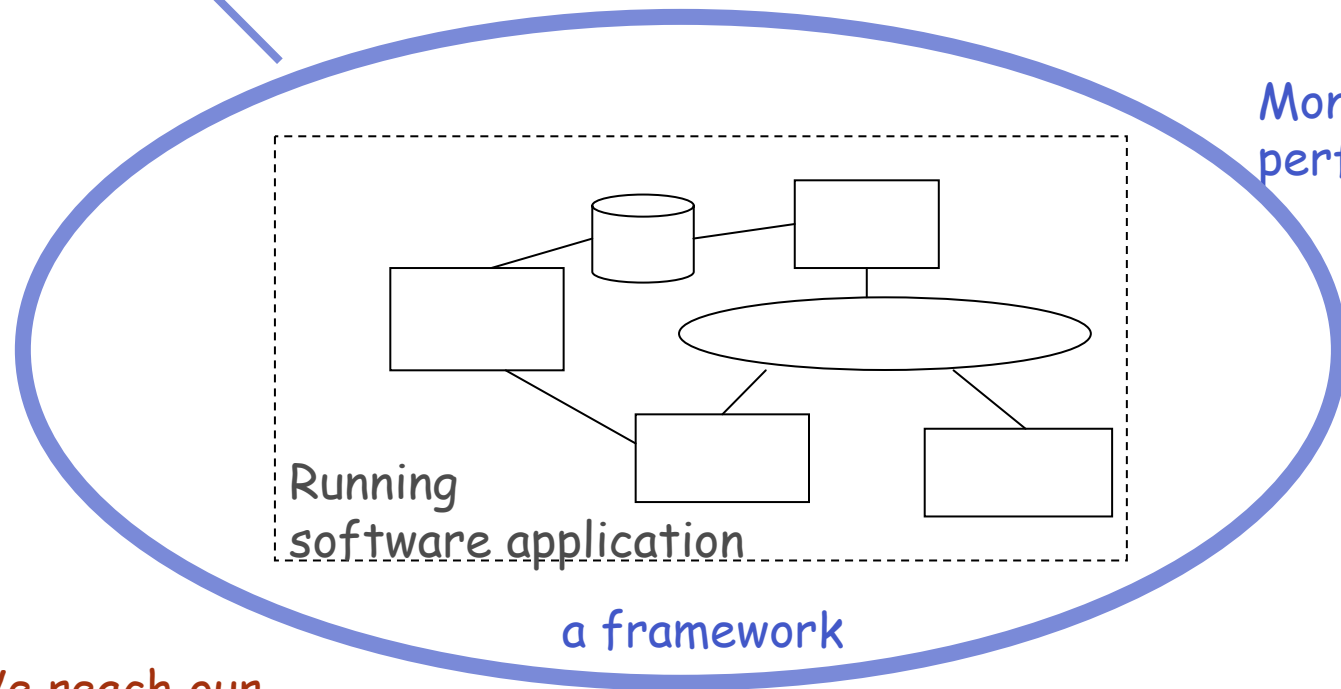
Ex. 2 - PERFORMANCE : system reconfiguration

Caporuscio-Di Marco-Inverardi

Reconfigure it dynamically

We want to ...

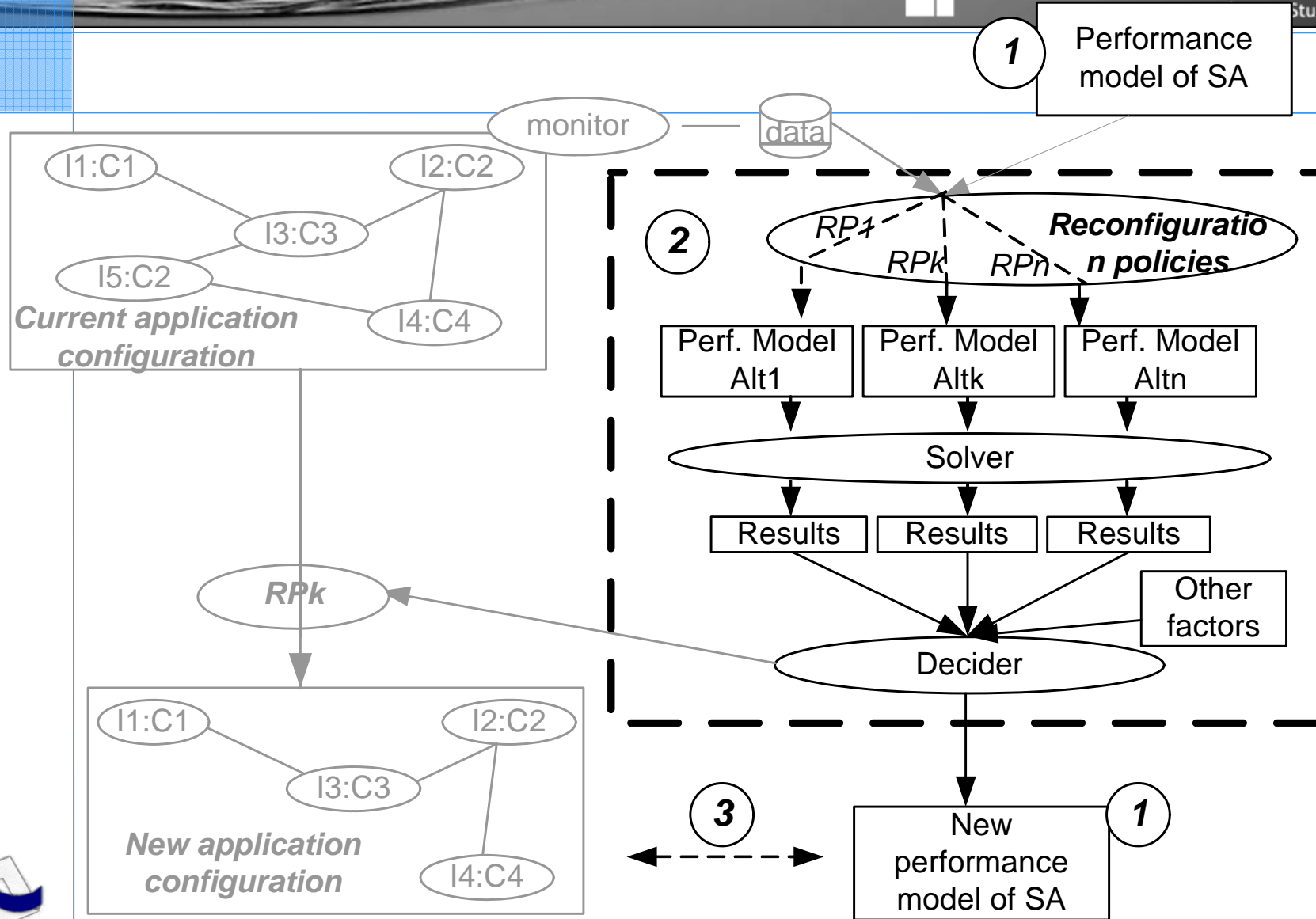
Monitor its performance



We reach our aims by means of ...

Decide its next running configuration





(b) Flow of the activities in an adaptation step

Interesting Issues

- » What is the relevant data to collect? And how to use it?
 - Data collected is more fine-grained than the performance model parameters.

- » 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,...)





CONNECT

Emergent Connectors for
Eternal Software Intensive Networked Systems



Context-Aware Adaptive Services: The PLASTIC Approach

*Marco Autili, Paolo Di Benedetto,
Paola Inverardi*

University of L'Aquila (Italy)



THALES

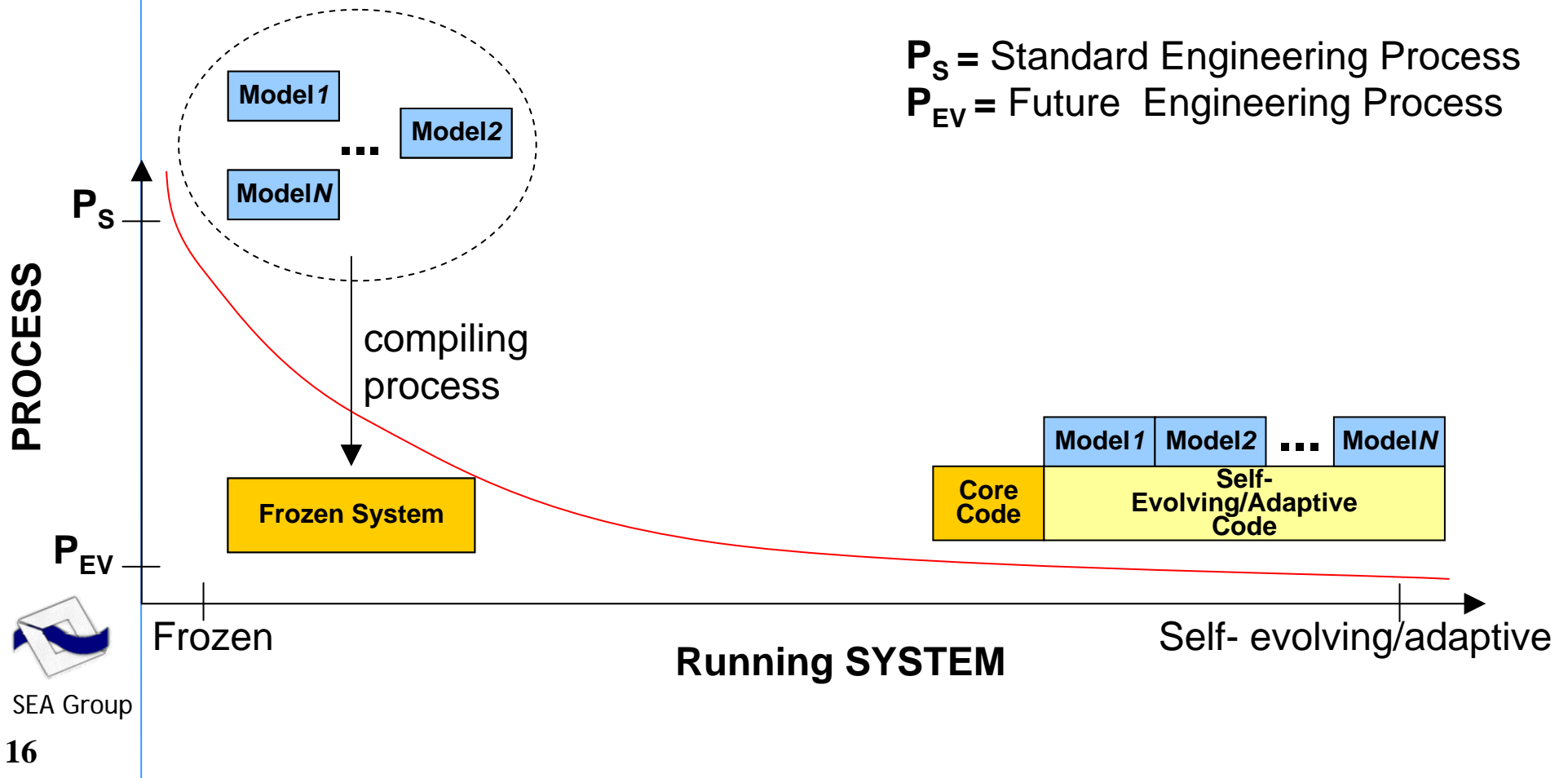


tu

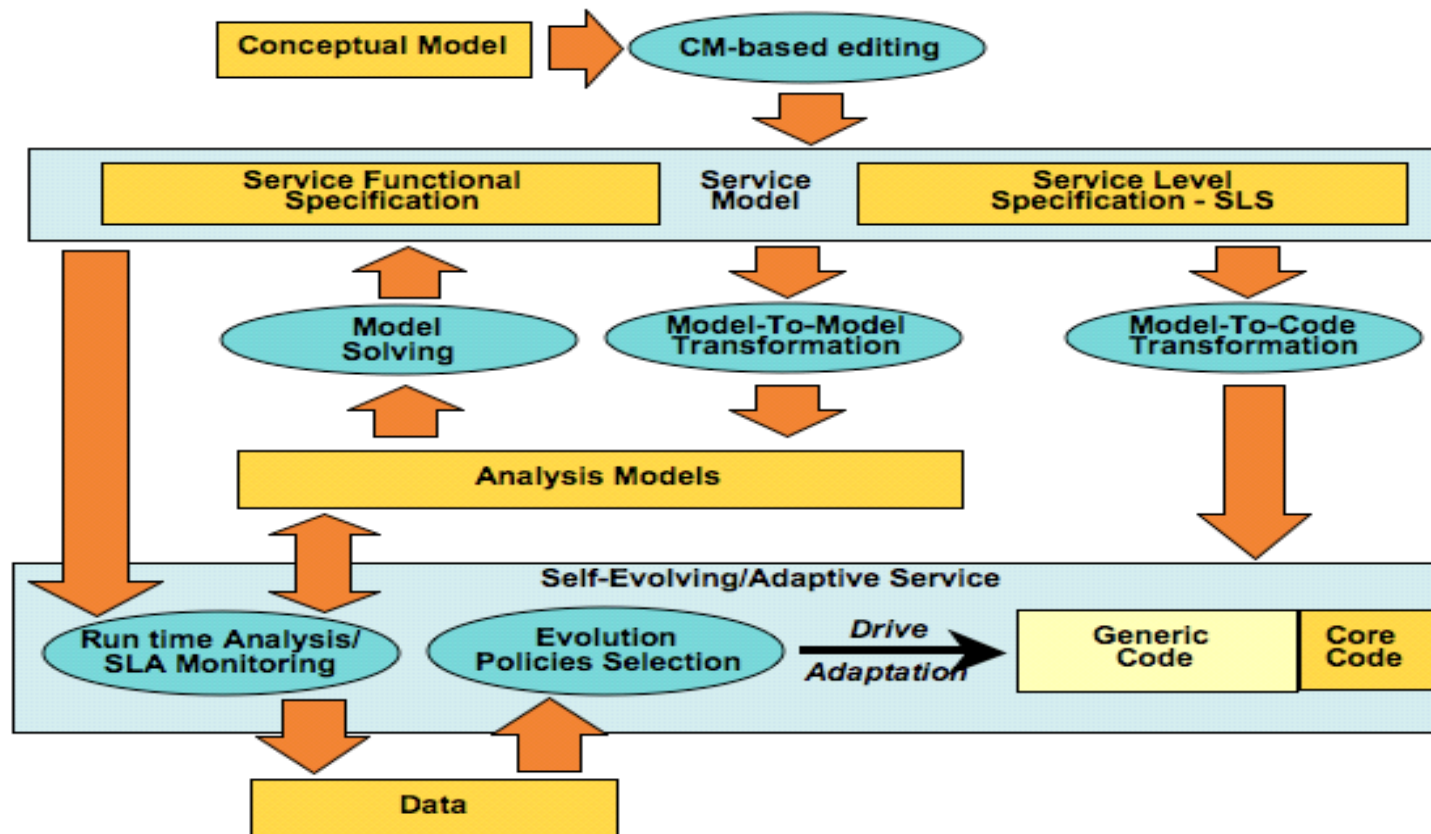
technische universität
dortmund



The Process View



PLASTIC Process



Introduction 1/2

- » *Ubiquitous networking* (empowered by B3G networks) allows mobile users to access software services across *heterogeneous infrastructures* through (*resource-constrained*) mobile devices

- » To offer users the best QoS according to their needs and specific execution environment, services need to be:
 - *Context-aware: aware of resources offered by the execution environment*
 - *SLS-aware: aware of user needs*
 - *Adaptive: capable of changing in order to comply with the current context conditions and user preferences*



Introduction 2/2

PLASTIC

IST project for development, deployment and validation of context-aware adaptive services over B3G networks

PLASTIC approach to adaptive services

- adaptive applications to provide and/or consume services
 - CHAMELEON: framework for the development and deployment of adaptive Java applications
- modified Service-Oriented Interaction Pattern
 - to publish and discover adaptive services
 - to account for non functional service properties and user preferences



PLASTIC Adaptation(s): SLS-based

- » **Service Level Specification (SLS)**
non-functional characteristics of the service
 - *Offered SLS* (provider)
 - *Requested SLS* (consumer)

- » **Service Level Agreement (SLA)**
conditions on the QoS under which the service is provisioned,
accepted by both the service consumer and the service provider

SLS-based adaptation strategy

The service provider publishes a "*generic*" service with different *Offered SLSs*.

The exposed service is adapted at discovery time to comply with the consumer *Requested SLS*.



PLASTIC Adaptation(s): context-aware

Context

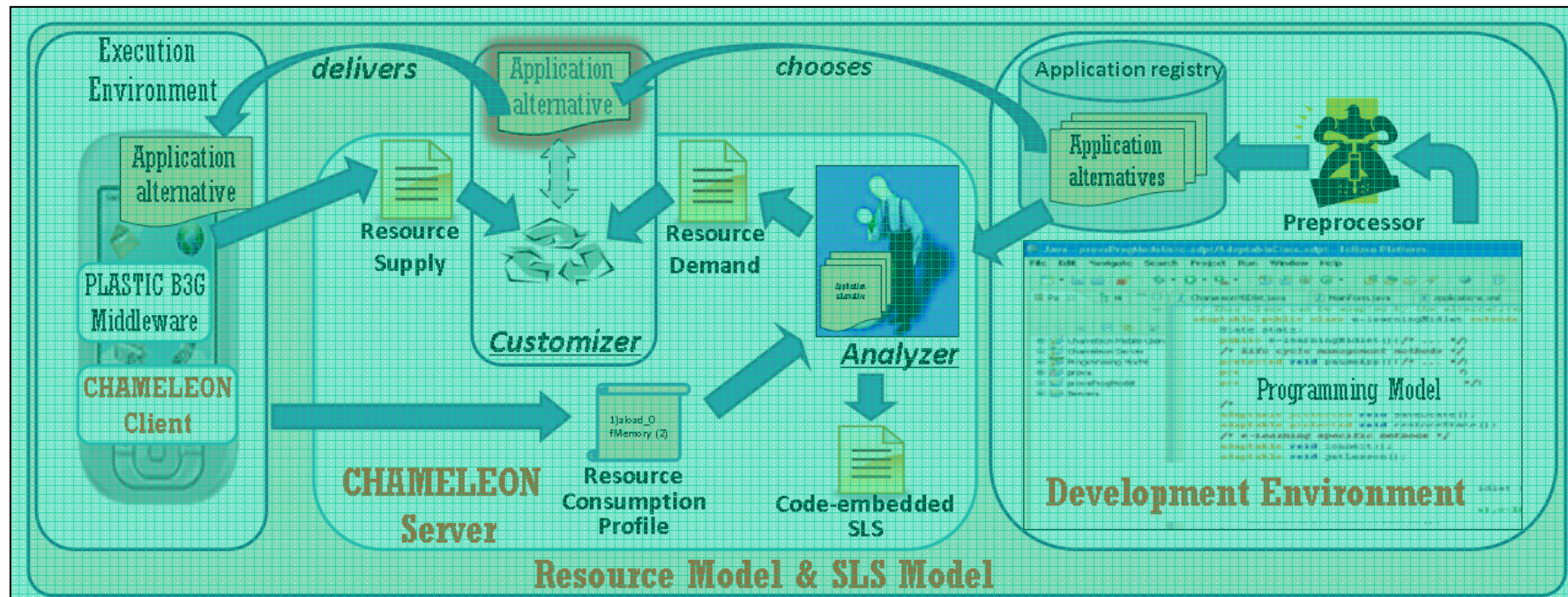
resource characteristics of the provider, network and consumer environments in which a service is provisioned and consumed.

Context-aware adaptation strategy

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



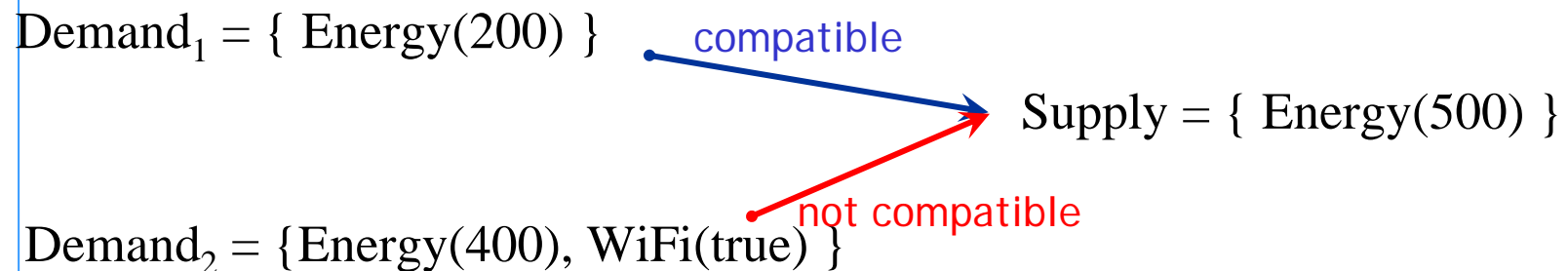
- Development of adaptive Java (*generic-code*) applications.
- Deployment of Java (*standard-code*) tailored applications
- Fully implemented in Java



CHAMELEON Resource Model

- » Formal model for characterizing resources
- » Provides definitions and operators to:
 - define the resources provided by an execution environment (**Resource Supply**)
 - compute the resources necessary to correctly execute an application (**Resource Demand**)
 - determine if an application can run safely in an execution environment (**Resource Compatibility**)

defineRES Energy as Natural
defineRES WiFi as Boolean



CHAMELEON SLS Model

- » Formal model for specifying non-functional preferences
- » Provides definitions and operators to:
 - specify the **Offered** and **Requested SLS**
 - derive the non-functional preferences tied to the implementation of a service (**Code-embedded SLS**)
 - determine if a Offered SLS satisfies a Requested SLS (**SLS Compatibility**)
 - defineSLS Throughput as {low, medium, high}
 - defineSLS Mobility as {low, medium, high}

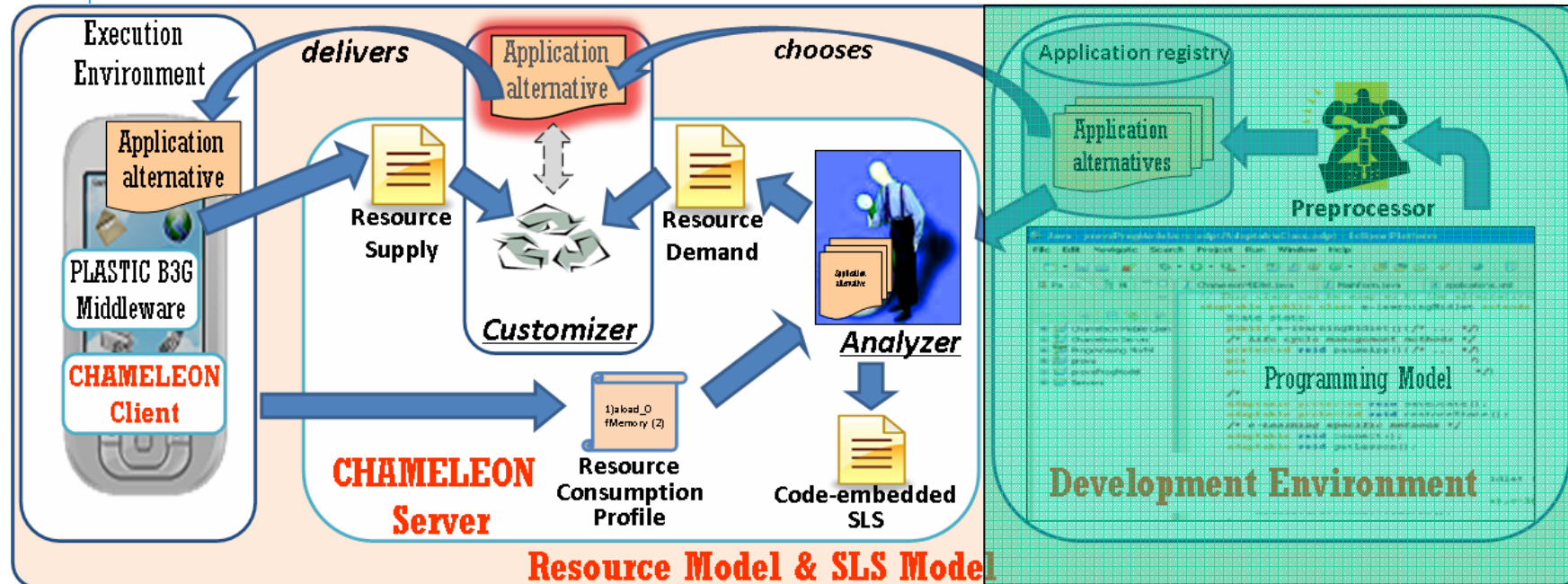
Requested_{SLS} = { Throughput (medium) }

compatible

Offered_{SLS} = { Mobility(low), Throughput (high) }



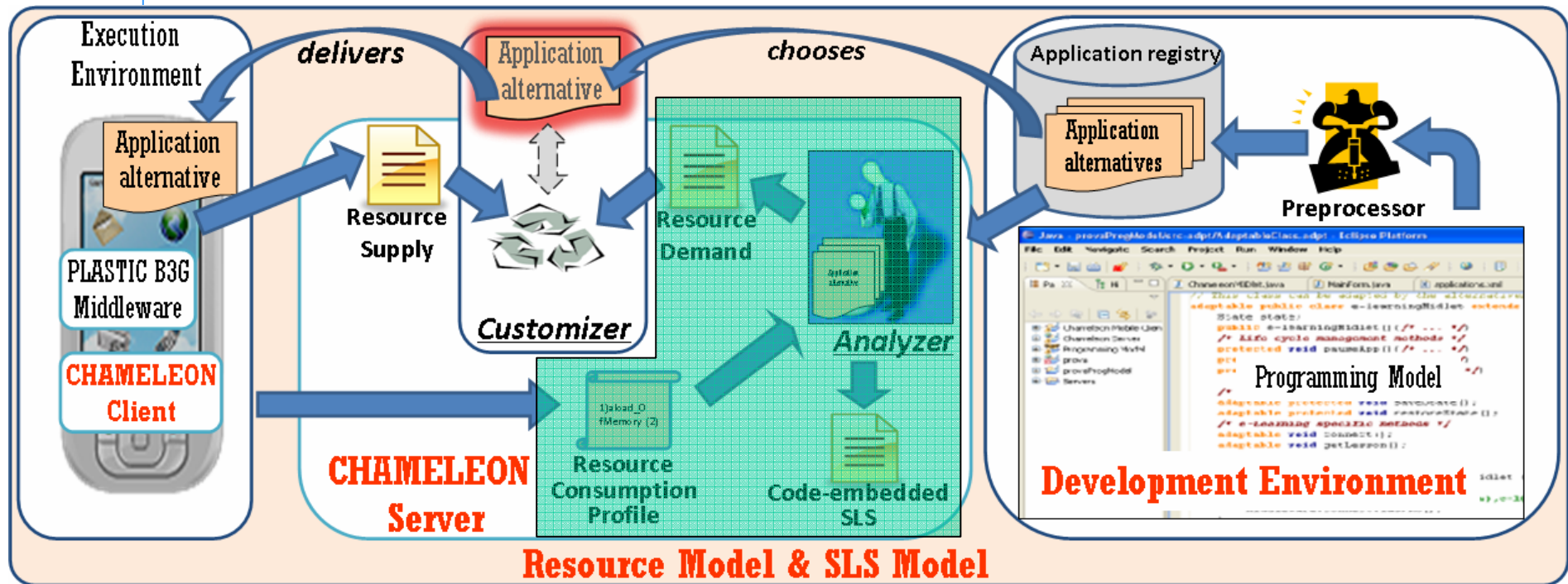
CHAMELEON framework



- **Programming Model:** permits to implement applications in terms of *generic code* (extension to the Java language).
- **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.



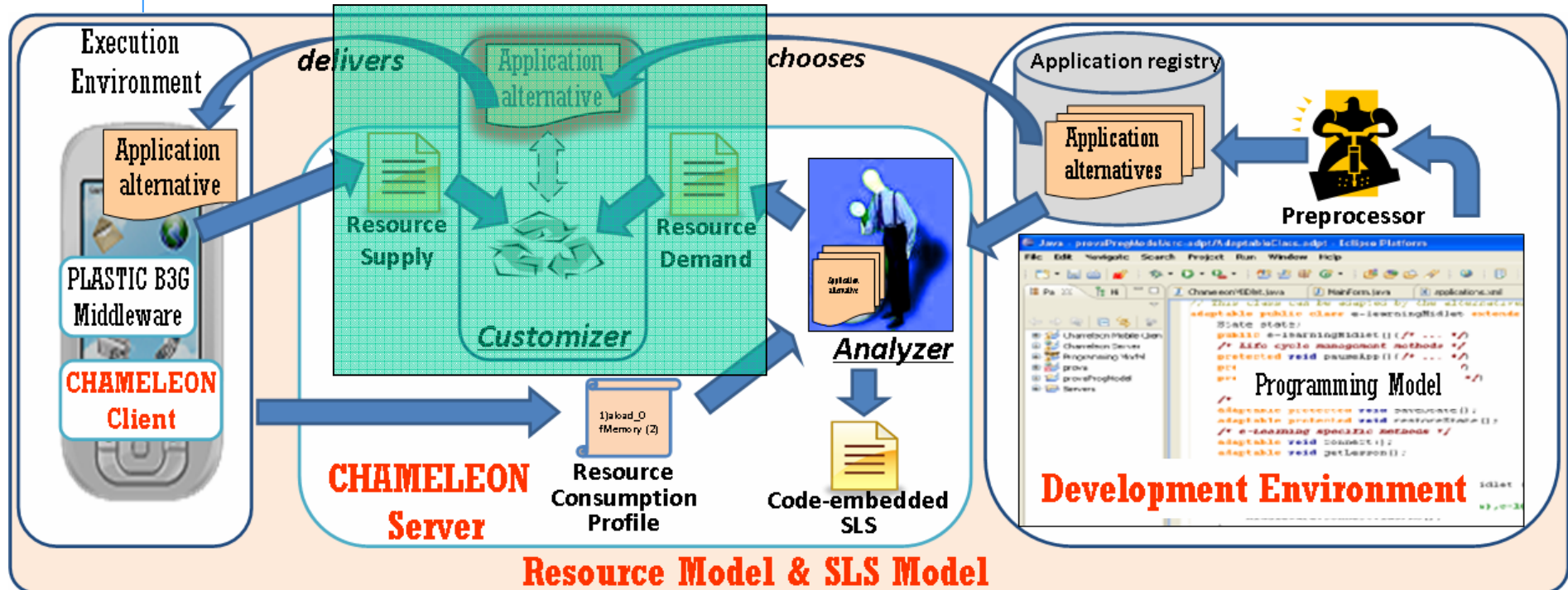
CHAMELEON Analyzer



- Statically analyzes each application alternative
- Abstracts a standard Java Virtual Machine
- Derives the **Resource Demand** and the **Code-embedded SLS**



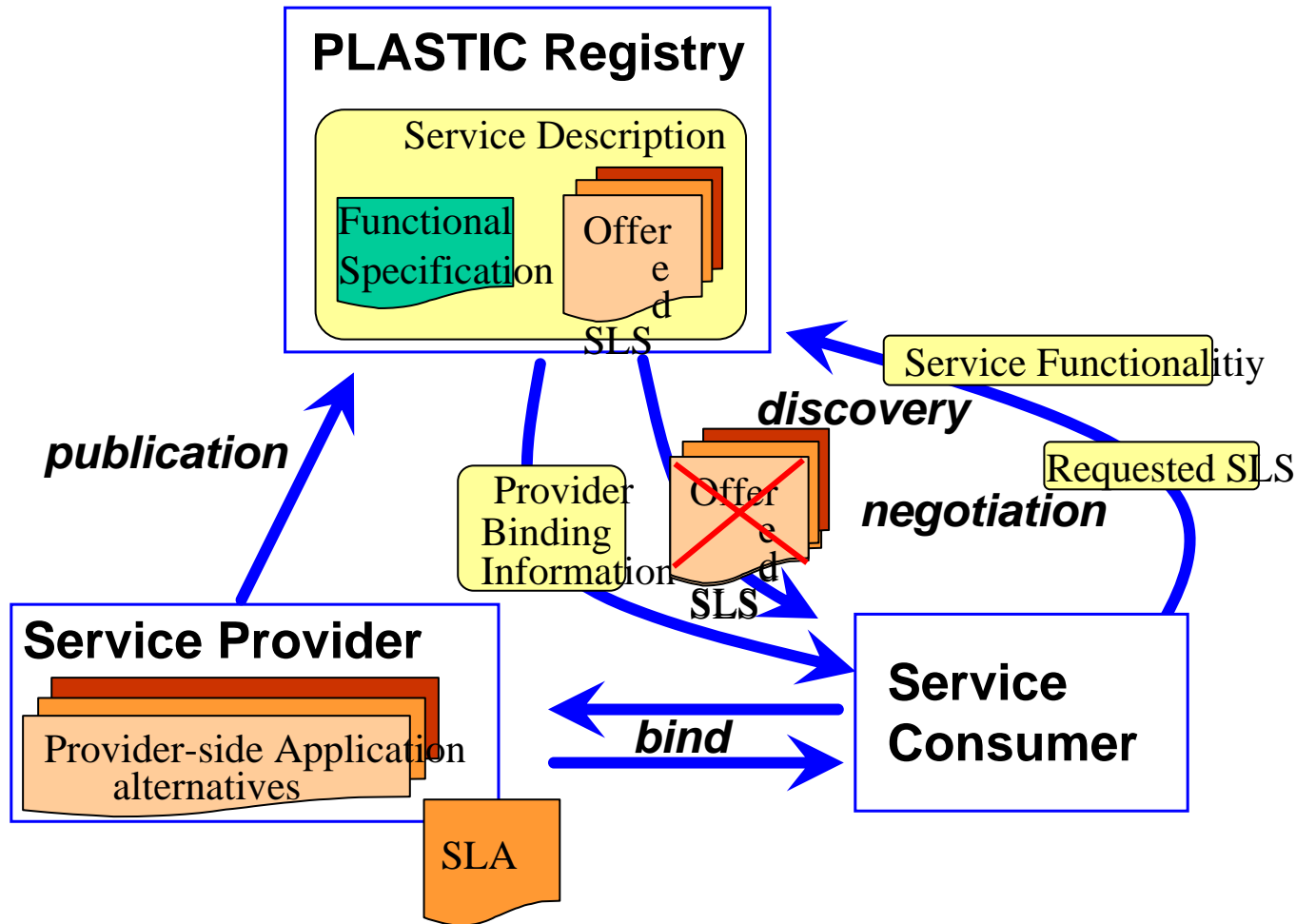
CHAMELEON Customizer



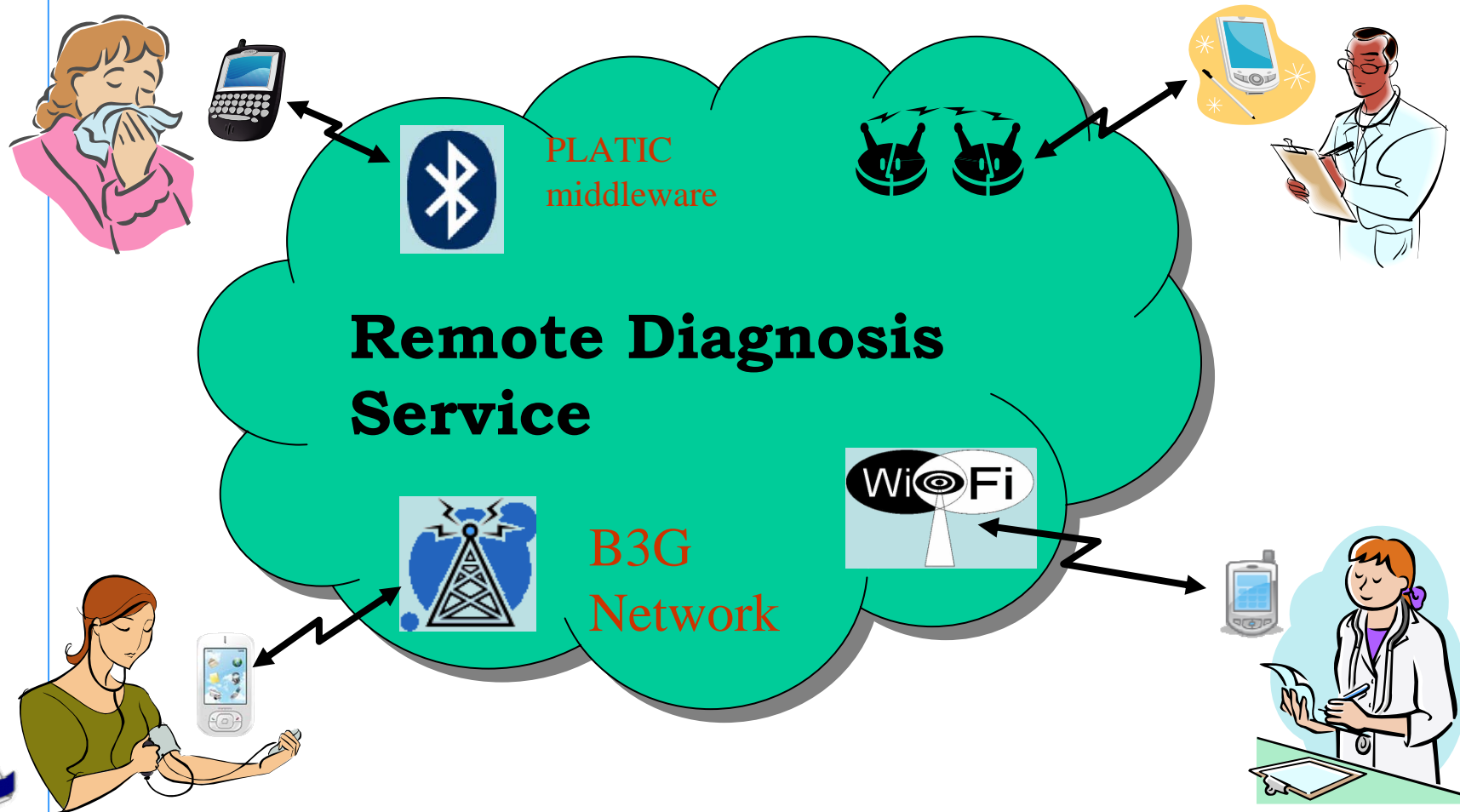
- Compares the *resource demand* of the alternatives with the *resources supplied* by the execution context
- Determines the application alternatives that can run safely in the execution context (i.e., *compatibility*)



PLASTIC Service-Orien. Interaction Pattern



A sample scenario 1/3



A sample scenario 2/3

Alternative	Resource Demand	Code-embedded SLS
Cons_1	{ Energy(200) }	{ Throughput(low) }
Cons_2	{ WiFi(true), Energy(400) }	{ Throughput(high) }

Alter.	Features	Resource Demand	Code-embedded SLS
P1	Transmits images from patient camera, stores and makes available manually inserted vital parameters values.	{ GPRS(true), ..., Memory(128), Energy(500),... }	{ Throughput(low), Mobility(high) }
P2	Transmits images and streams video from patient camera, collects and makes available data automatically retrieved by measurement instruments.	{ WiFi(true), ..., HRM(true), Memory(512), Energy(1000), Termometer(true), ..., PressureMeter(true), ... }	{ Throughput(high), Mobility(medium) }

	Cons_1	Cons_2
P1	{ Throughput(low), Mobility(high) }	{ Throughput(low), Mobility(high) }
P2	{ Throughput(low), Mobility(medium) }	{ Throughput(high), Mobility(medium) }

Combined Code-embedded SLS



A sample scenario 3/3

CHAMELEON Application Registry

Remote Diagnosis

	Resource Demand	Cons_1	Cons_2
P1	{ GPRS(true), Memory(128), }	{ Throughput(low), Mobility(high) }	{ Throughput(low), Mobility(high) }
P2	{ Memory(512), HRM(true), WiFi(true), }	{ Throughput(low), Mobility(medium) }	{ Throughput(high), Mobility(medium) }

	Resource Demand	Code-embedded SLS
Cons_1	{ Energy(200) }	{ Throughput(low) }
Cons_2	{ WiFi(true), Energy(400) }	{ Throughput(high) }

PLASTIC Registry

Remote Diagnosis

	Alt	Cons_1	Cons_2
George' PLASTIC@	P1	{ Throughput(low), Mobility(high) }	{ Throughput(low), Mobility(high) }
George' PLASTIC@	P2	{ Throughput(low), Mobility(med) }	{ Throughput(high), Mobility(med) }

- Resource Supply
- Resource Consumption Profile

	George' PLASTIC@, P1	George' PLASTIC@, P2
Cons_1	{ Throughput(low), Mobility(high) }	{ Throughput(low), Mobility(med) }

- Supply={ Energy(500) }
- Resource Consumption Profile
- Requested SLS={ Mobility(n

George
Patient (provider)
P2 P1

{ Throughput(low),
Mobility(high), Rate(low) }

Dr. Smith

Cons_1 → George@PLASTIC, P1

{ Throughput(low),
Mobility(high), Rate(low) }

Doctor (consumer)

bind


Conclusions and Future Work

- » The **PLASTIC** approach to context-aware adaptive services
- » In PLASTIC 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



Conclusions and Future Work

- » Provide self-evolving services
 - *Evolution by dynamically un-deploying the no longer apt application alternative and subsequently (re)deploying a new alternative that can preserve the SLA

 - Make (limited) self-evolving adaptable applications:
 - > knowledge of user *mobility patterns*
 - > deployment of all the alternatives necessary to preserve the SLA for all the execution contexts specified by the mobility pattern

<http://www.di.univaq.it/chameleon>

* Autili M., Di Benedetto P., Inverardi, P., Tamburri D.A.: Towards self-evolving context-aware services. - DisCoTec 2008



The End

» The **PLASTIC** approach to context-aware adaptive services



» Approach key points:

- twofold adaptation
 - > SLS-based adaptation
 - > Context-aware adaptation
- CHAMELEON adaptive applications to provide and consume services

Modified SOA interaction pattern to publish and discover context-aware and SLS-aware adaptive services

CHAMELEON
<http://www.di.univaq.it/chameleon/>

PLASTIC
<http://www.ist-plastic.org/>



Related Work

» Resource-oriented analysis

- D. Aspinall, K. MacKenzie. Mobile Resource Guarantees and Policies. In Construction and Analysis of Safe, Secure, and Interoperable Smart Devices. 2006.
- G. Barthe. MOBIUS, Securing the Next Generation of Java-Based Global Computers. ERCIM News, 2005.
- C. Seo, S. Malek, N. Medvidovic. An energy consumption framework for distributed java-based systems. In ASE. 2007.
- Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Staschulat, J., Stenström, P.: The worst-case execution-time problem—overview of methods and survey of tools. Trans. on Embedded Computing Sys. 7(3), 1–53 (2008)
- Java Path Finder (W. Visser, K. Havelund, G. Brat, S. Park, and F. Lerda. Model checking programs. ASE journal, 10(2), 2003.)

» Context-aware services

- R. Rouvoy, F. Eliassen, J. Floch, S. O. Hallsteinsen, E. Stav. Composing Components and Services Using a Planning-Based Adaptation Middleware. In SC. 2008.
- N. Paspallis, G. A. Papadopoulos. An Approach for Developing Adaptive, Mobile Applications with Separation of Concerns. In COMPSAC. 2006.



A completely open scenario: CONNECT

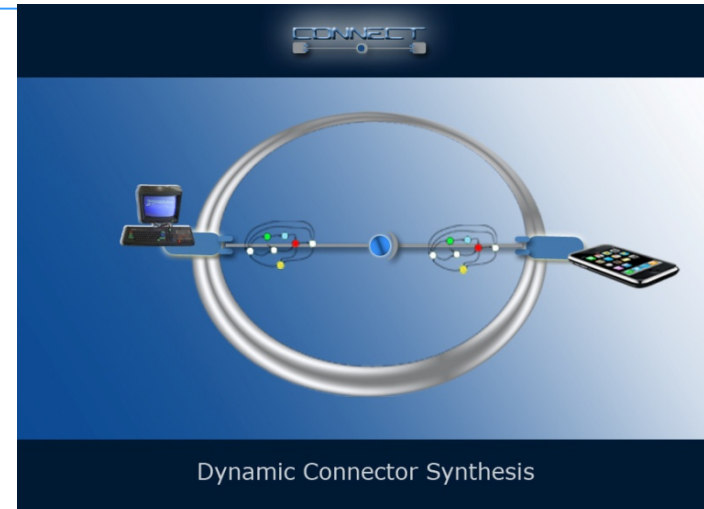
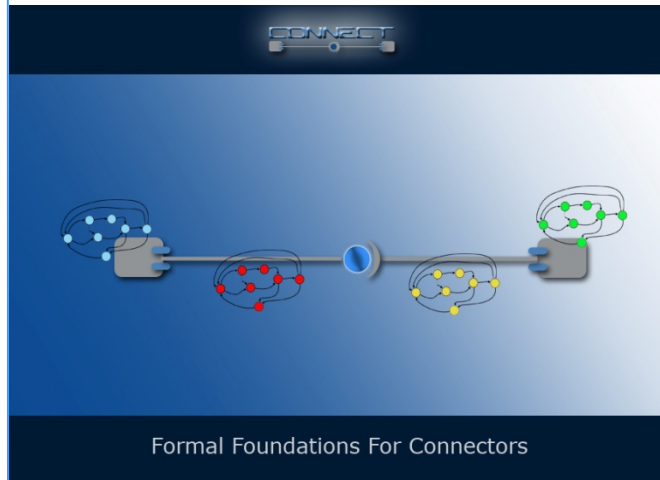
- » Ubiquitous systems: components travel around willing to communicate with only their own knowledge
- » Exploit the process: discover-learn-mediate-communicate
- » No global SA assumed
- » The SA in terms of components and connectors results from the completion of the process
- » and dependability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



CONNECT scenario



CONNECT process





Emergent Connectors for
Eternal Software Intensive Networked Systems



CONNECT

Emergent Connectors for Eternal Software Intensive Networked Systems

FET ICT Forever yours

7FP-Call 3 - ICT-2007

Coordinated by Valerie Issarny INRIA

<http://connect-forever.eu/>



NTT
docomo
DOCOMO Euro-Labs

LANCASTER
UNIVERSITY



THALES



tu technische universität
dortmund



Introduction

- » Challenge 3
 - the automated synthesis of CONNECTors according to the interaction behaviors of networked systems seeking to communicate.

Main Objectives:

- » to devise automated and compositional approaches to the run-time synthesis of connectors that serve as mediators of the networked applications' interaction at both application- and middleware-layer
 - synthesis of application-layer conversation protocols
 - synthesis of middleware-layer protocols
 - model-driven synthesis tools

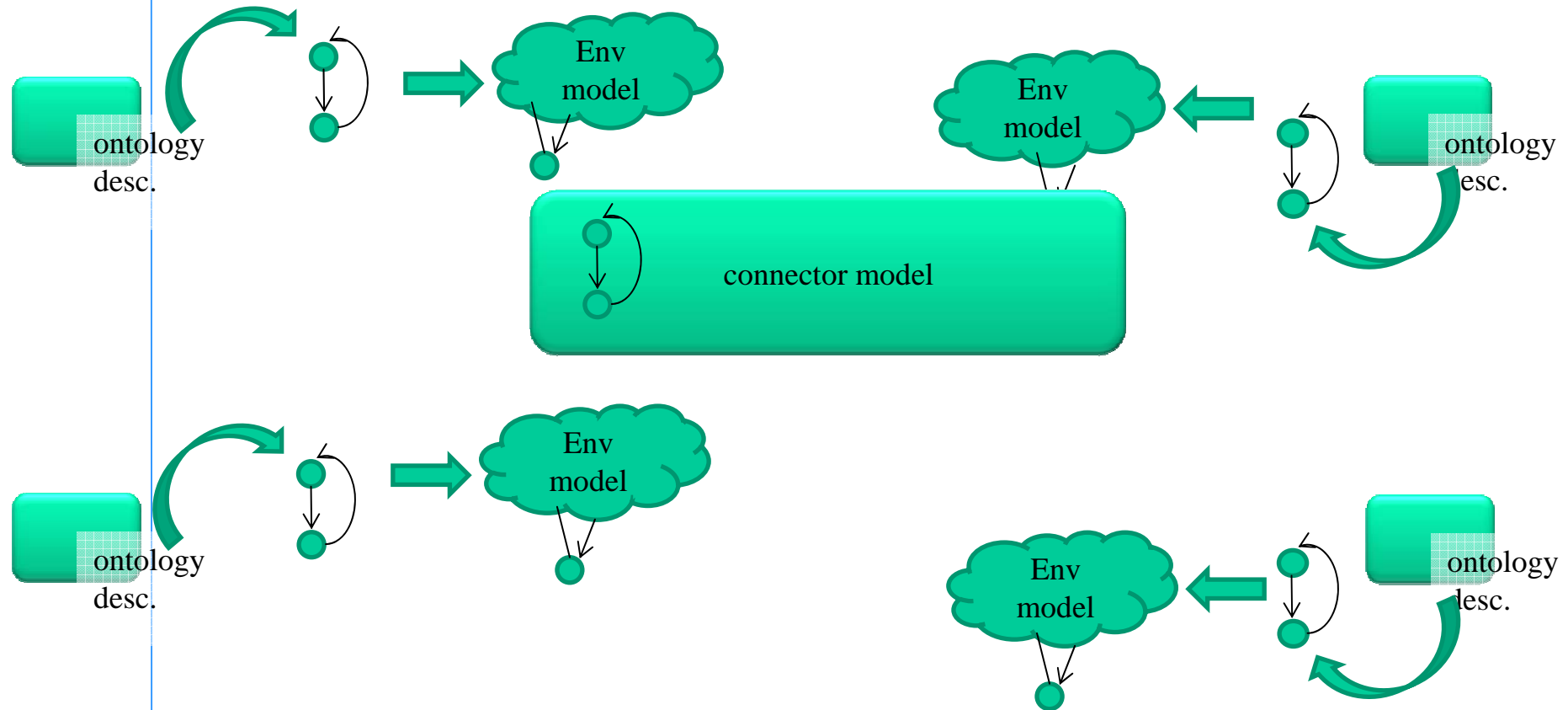


Synthesis of application-layer conversation protocols

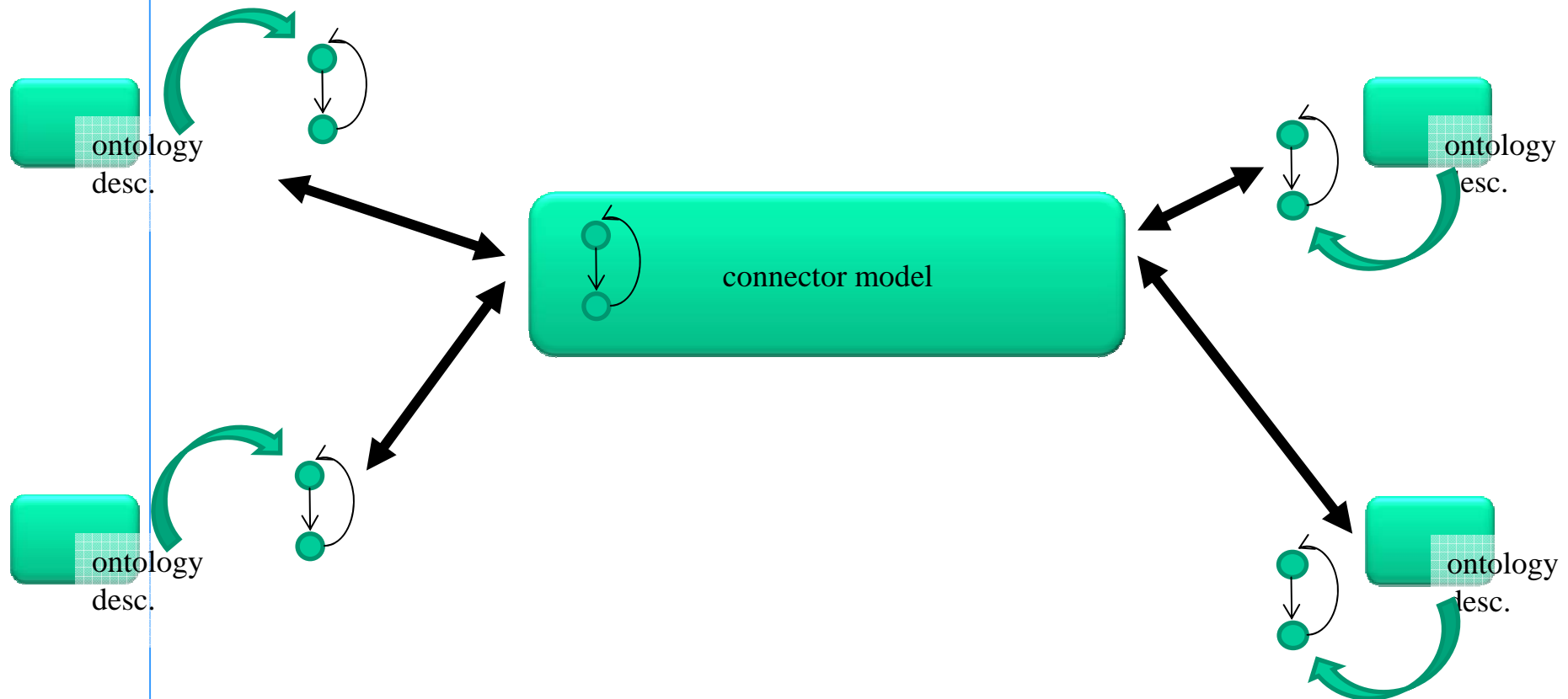
- » To support the automated construction of application-layer connector models
 - 1: identifying the conditions on the networked applications interaction and composition that enable run-time connector synthesis
 - > SA and connector patterns
 - 2: the synthesis process is seen as a behavioral model unification process
 - > ontologies
 - > modeling notations
 - > unifying know and unknown information
- » The challenge
 - compositionality and evolution



synthesis process steps



synthesis process steps



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 of middleware-layer protocols

- » Developing protocol translators
 - to make heterogeneous middleware interoperate
 - w.r.t. required non-functional properties
- » The challenges
 - interoperability of both data transfer protocols and interaction schemes
 - ensuring, at run-time, end-to-end properties
 - > availability, reliability, security, timeliness



A Formalization of Mediating Connectors: Towards on the fly Interoperability

R. Spalazzese (romina.spalazzese@di.univaq.it)

P. Inverardi (paola.inverardi@di.univaq.it)

V. Issarny (valerie.issarny@inria.fr)

Wicsa 2009

Mediating connectors (aka Mediators)

- » In modern networked systems many heterogeneity dimensions arise and need to be mediated
 - mediation of data structures
 - > data level mediators
 - > ontologies
 - mediation of functionalities
 - > functional mediators
 - > logic-based formalism
 - mediation of business logics
 - > application-layer protocol mediators
 - > process algebras, finite state machines, LTSs
 - mediation of message exchange protocols
 - > middleware-layer protocol mediators
 - > composition of basic mediation patterns



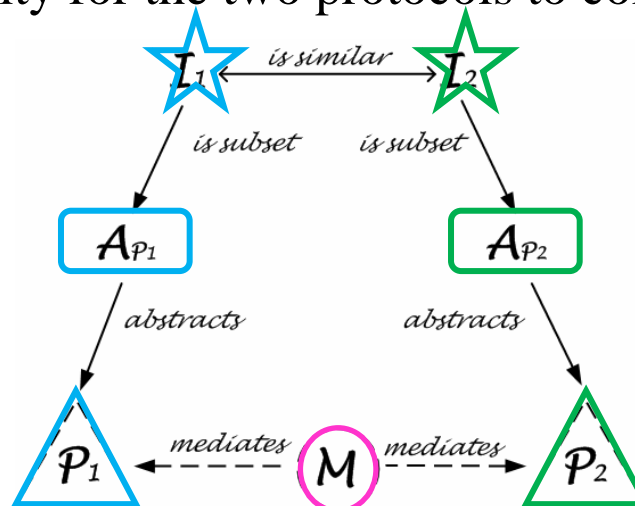
Foundations for the automated mediation of heterogeneous protocols

- » 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



Overview of our framework

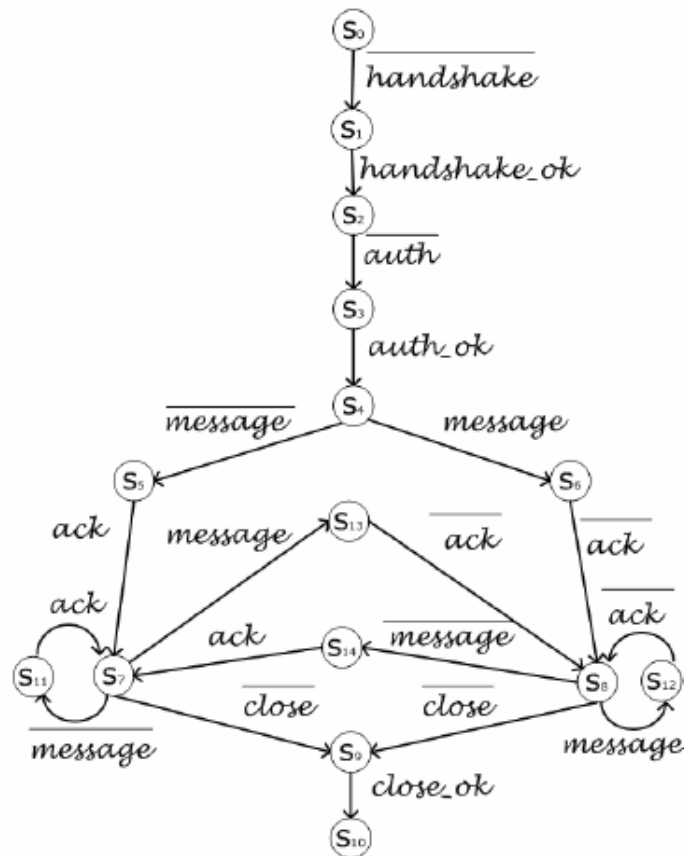
1. The approach, considering two protocols
 - *abstracts* the behavioral description of the protocols highlighting some *structural characteristics*
 - checks the possibility for the two protocols to communicate



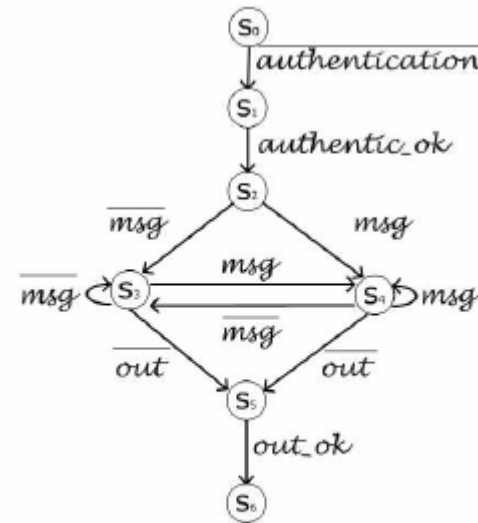
2. If the check succeeds (there is the possibility to communicate)
 - synthesize a suitable mediating connector

The instant messaging example

do they “share the same intent”?



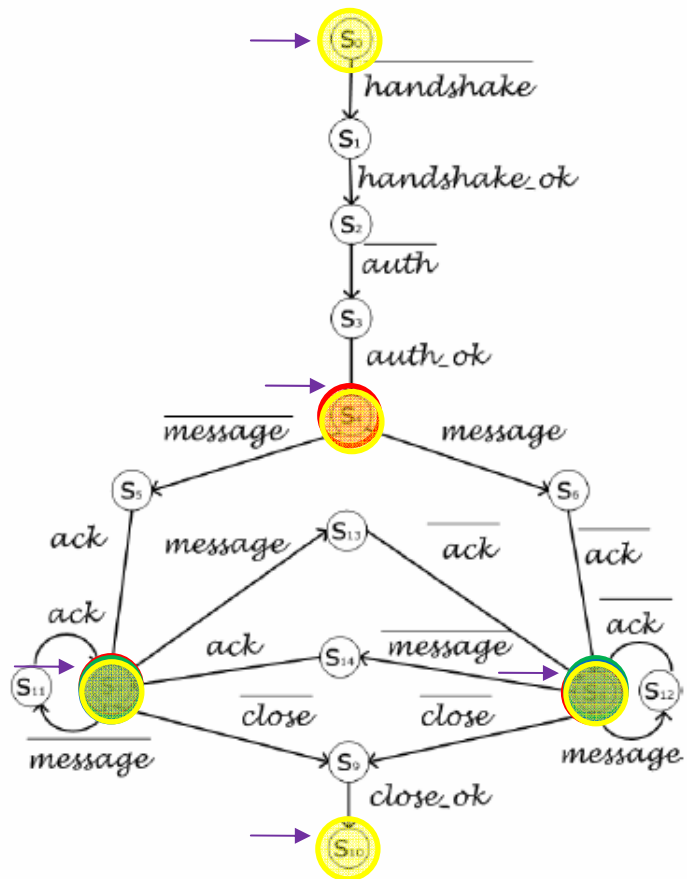
(a) Windows Messenger protocol



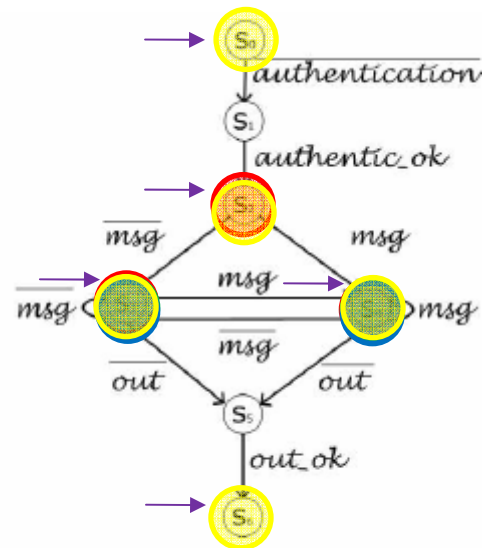
(b) Jabber protocol



The instant messaging example



(a) Windows Messenger protocol



(b) Jabber protocol

do they have similarities in the structure of their protocol models?

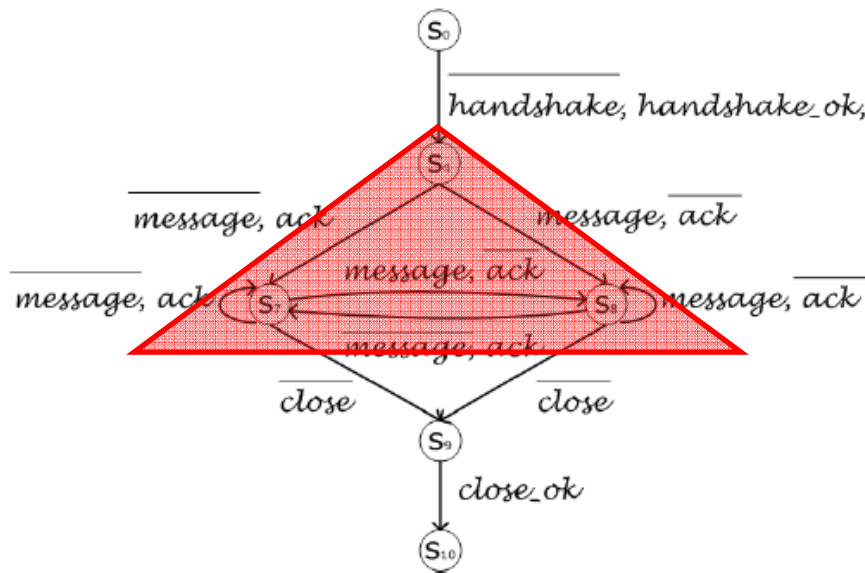
- **branch** states
- **entry cycle** states
- **convergence** states
- **rich** states
- **successive rich** states



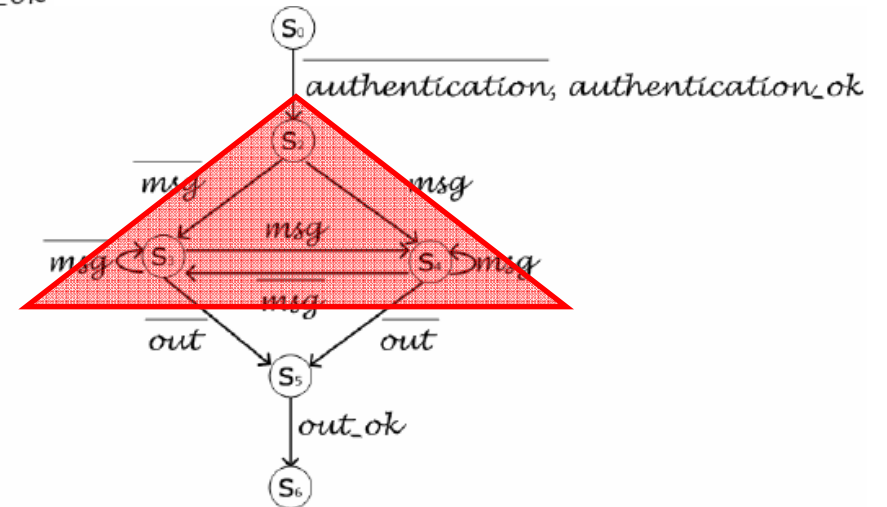
Common language structure

Ontology

"message, ack" \leftrightarrow "msg"



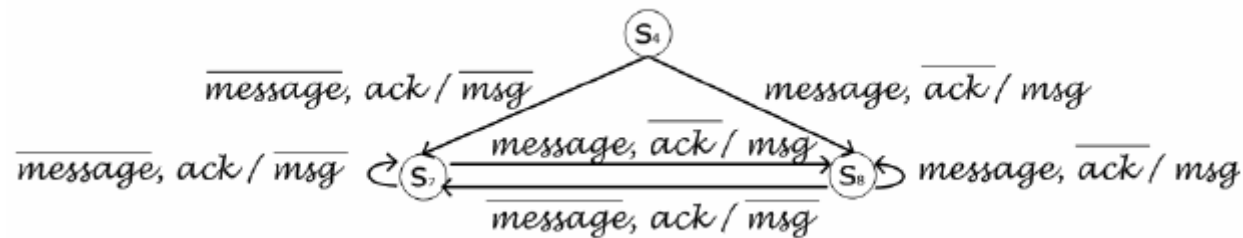
(a) Windows Messenger structure



(b) Jabber Messenger structure



Abstract mediator model

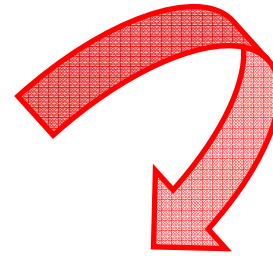
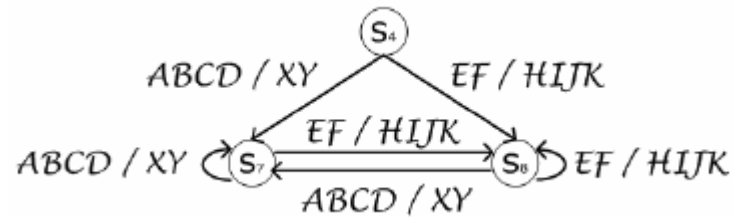


Indeed:

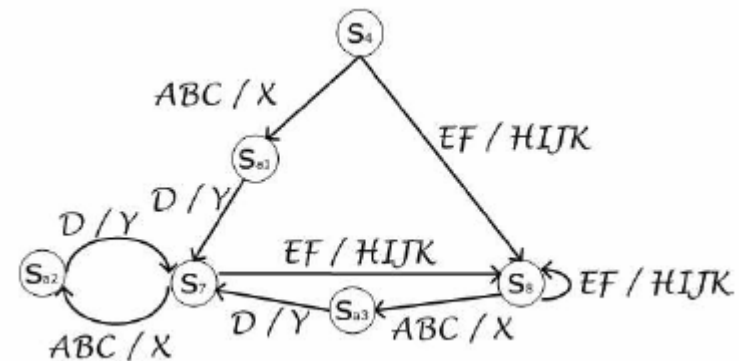
- the concrete mediator also provides the needed complementary behaviors to let the two protocols evolve;
- the concrete mediator “*simulates*” also the actions that should be exchanged with third parties;
- the concrete mediator takes into account also portions of complementary protocols for the part of their structure that is not the common language structures.



Refinement of the abstract mediator model



Ontology:
 "ABC" $\leftarrow - \rightarrow$ "X"
 "D" $\leftarrow - \rightarrow$ "Y"



Conclusion

- » first formalization of mediating connectors in the direction of the on the fly interoperability
- » The approach partially covers the existing mismatches
- » Assumptions:
 - partial structural similarities
 - data is not considered



Future work

- » Automation
- » Compositionality
- » Model-driven techniques for the synthesis of the mediator actual code
- » Evolution
- » Non-functional characteristics of the protocol behavior
- » *Dependability assurances*



References

[Betty H. C. Cheng](#), [Rogério de Lemos](#), [Holger Giese](#), Paola Inverardi, [Jeff Magee](#): Software Engineering for Self-Adaptive Systems [Springer 2009](#)

[Patrizio Pelliccione](#), Paola Inverardi, [Henry Muccini](#): CHARMY: A Framework for Designing and Verifying Architectural Specifications. [IEEE Trans. Software Eng.](#) **35**(3): 325-346 (2009)

Paola 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)

Romina Spalazzese, Paola Inverardi, Valerie Issarny: Towards a Formalization of Mediating Connectors for on the Fly Interoperability. [WICSA 2009](#).

[Marco Autili](#), [Paolo Di Benedetto](#), Paola Inverardi: Context-Aware Adaptive Services: The PLASTIC Approach. [FASE 2009](#): 124-139

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

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).

