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

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: define 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.

Changes in the Software Architecture

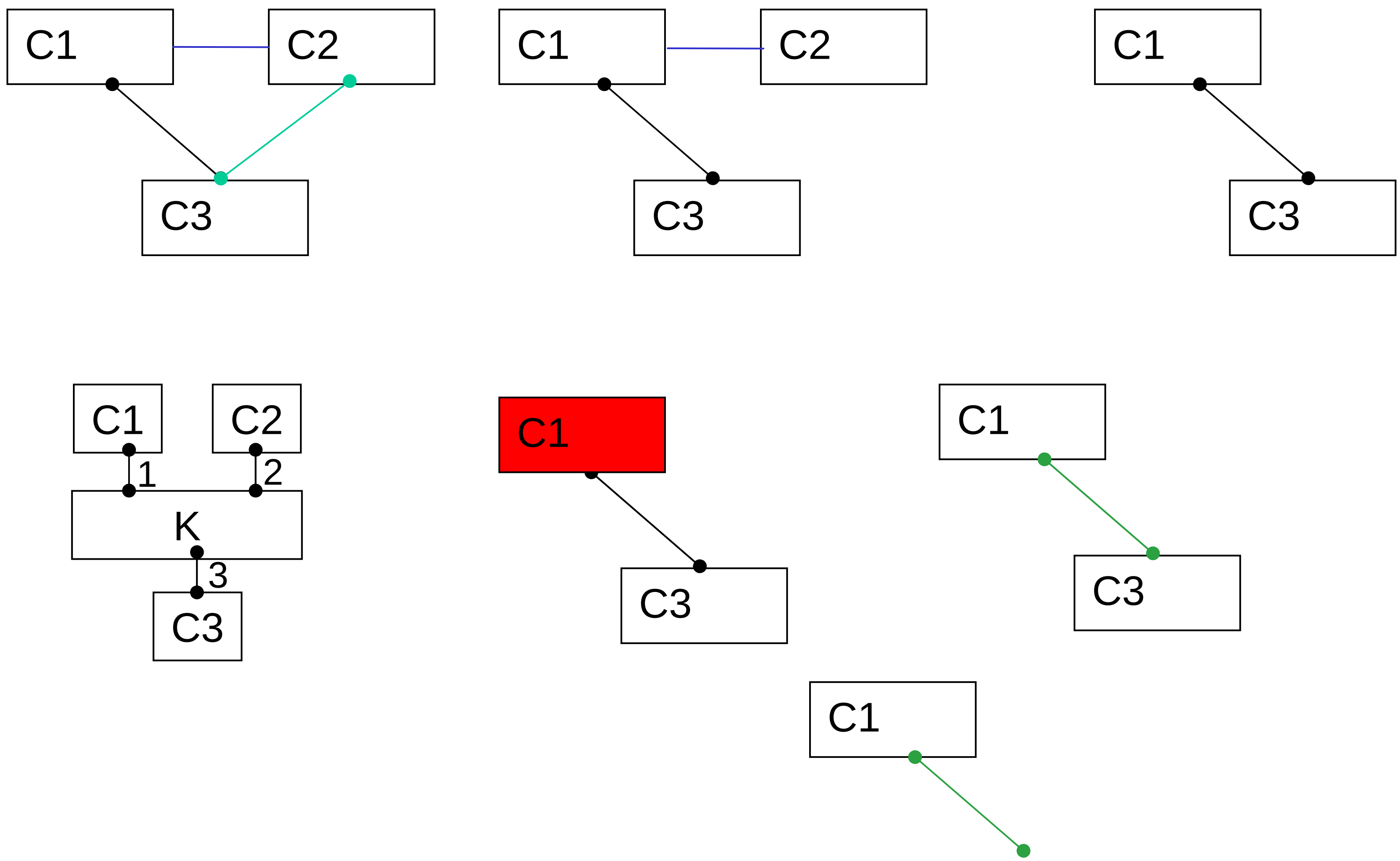
» Structure:

- components can get in and out, new connectors i.e. new connections and/or new interaction protocols

» Behavior:

- Components can change their functionality, connectors can change their protocols

Variability dimensions in SA



Software Architecture and dependability

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

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

The role of the SA in an open world

» Changes in both the context and user needs might imply architectural configuration changes

- e.g., addition/arrival, replacement, removal/departure of components

» The closed world assumption does not hold anymore

» **Dependability cannot be *deduced* only by composition anymore**

- it can be unfeasible to fix a priori the SA and, then, deduce dependability
- the experienced dependability might be not the wished one

» The role of the SA is inverted

» **Composition *induced* by dependability**

- a priori specification of a wished dependability degree
- dynamic induction of the SA that fulfills as best as possible the specified dependability

Composition induced by user-level dependability requirements 1/2

» Promising technologies

- service mash-up
- widget Uis
  - > SAMSUNG Widgets
  - > Win Vista, Yahoo, MAC OS Gadgets

» They shift composition from the developer-level to the end-user-level

- to ease the consideration of user-level dependability requirements

» However, they are still conceived to be used with the closed-world assumption in mind

Composition induced by user-level dependability requirements 2/2

» While keeping a high-level composition mechanism, suitable technologies should

- allow the user to **specify dependability requirements**
- **propose the architectural configuration** enabling the composition **that fulfills dependability**
- dependability should be kept **despite of possible context changes**
  - > dynamic induction and evolution of the system SA

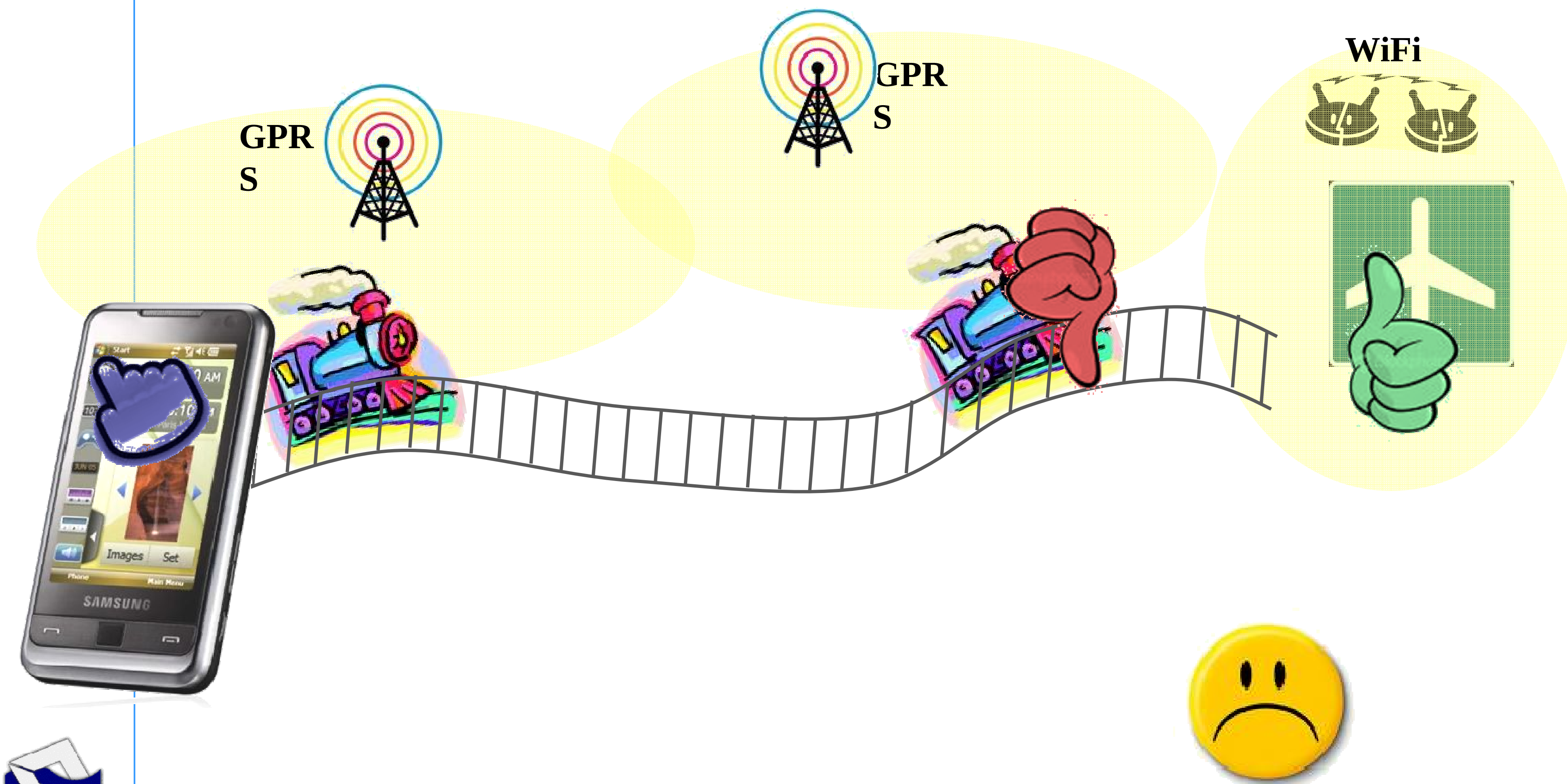
Widget UIs in e-learning

» Two possible scenarios illustrating

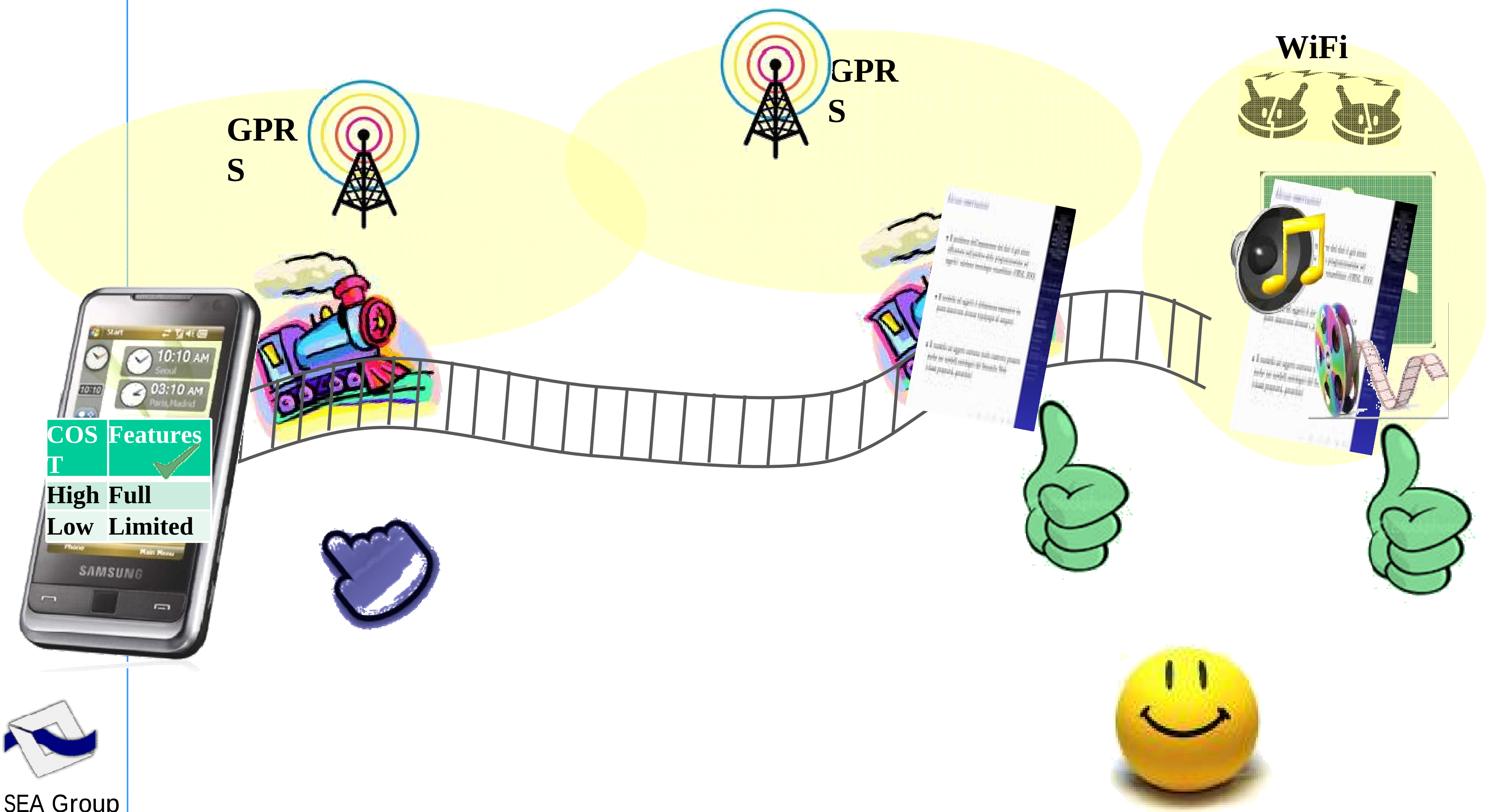
(a) how, in an open world, a SA fixed a priori can imply, a possibly, unexpected dependability

(b) how, instead, dependability specified a priori can imply the "best possible" SA

e-Learning scenario (a)



e-Learning scenario (b)



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

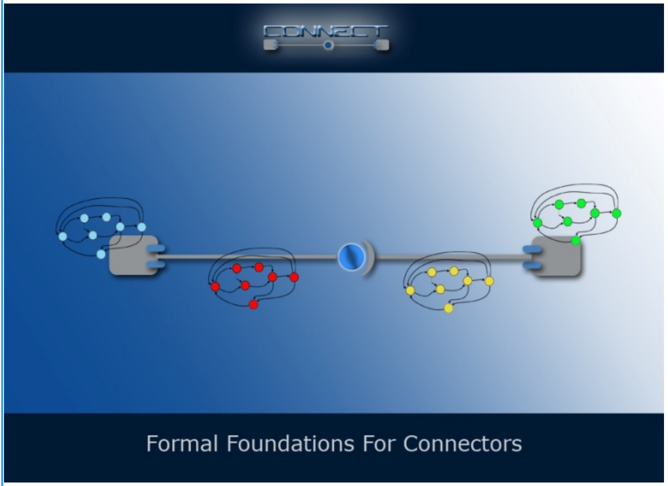
» No global SA assumed

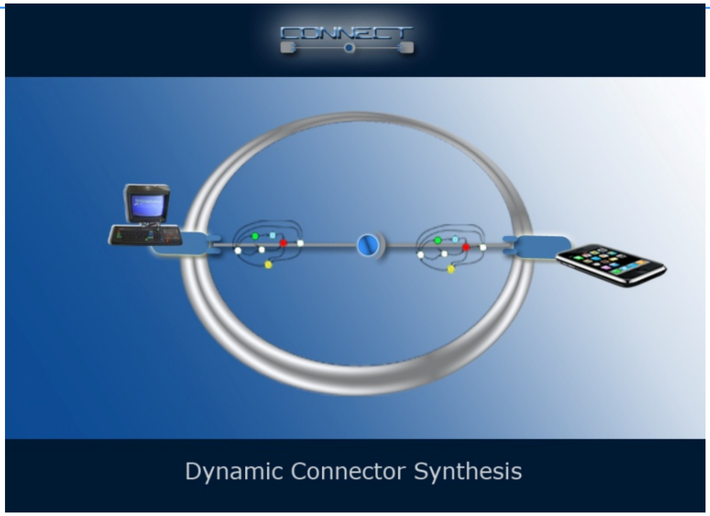
» The SA in terms of components and connectors results from the completion of the process

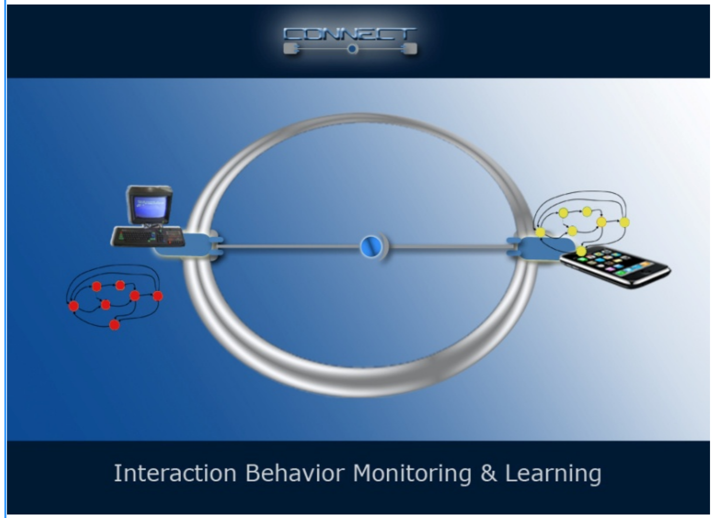
» and dependability … ? It is built in the composition e.g. embedded in the connectors (ref. Synthesis, de Lemos08).

CONNECT scenario

CONNECT process







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/

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

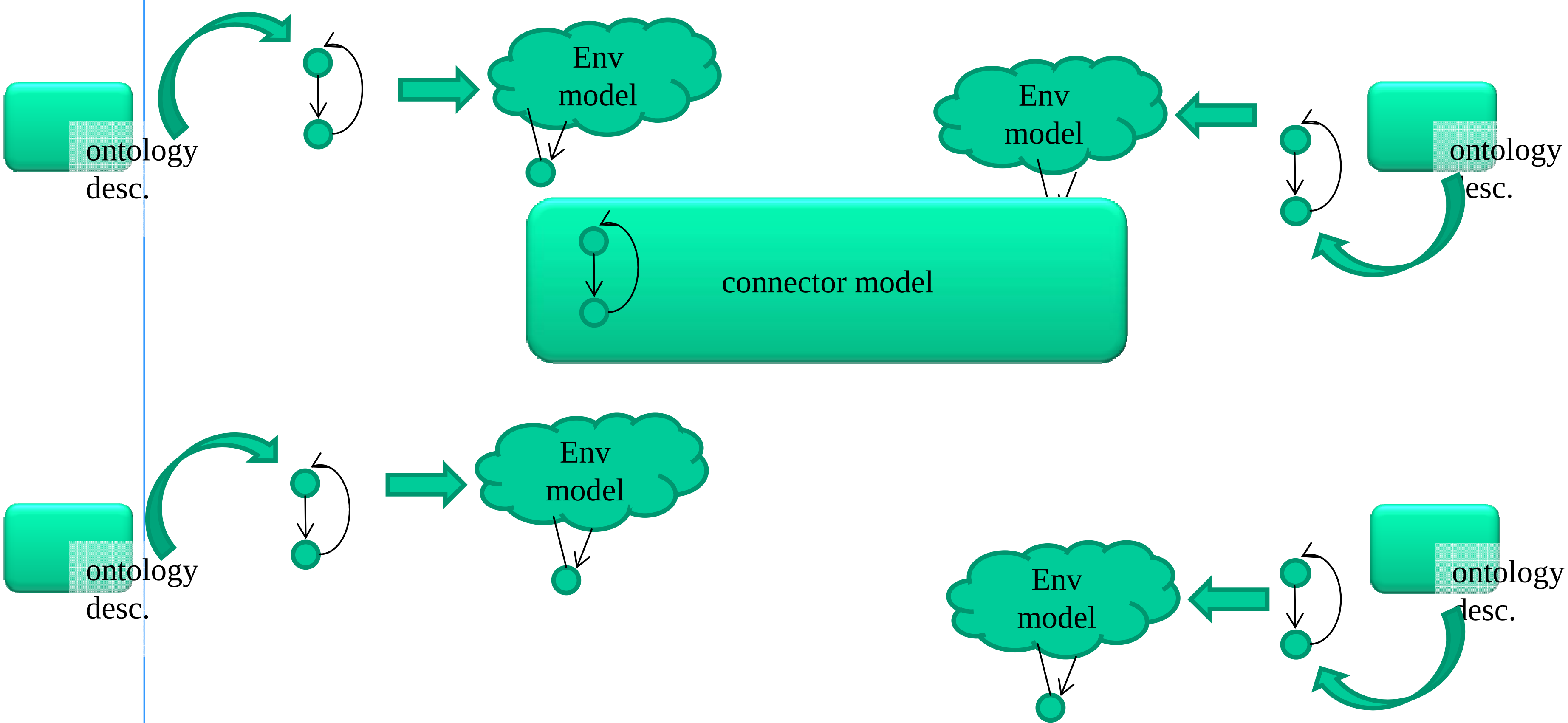
» To support the automated construction of application-layer connector models

- 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

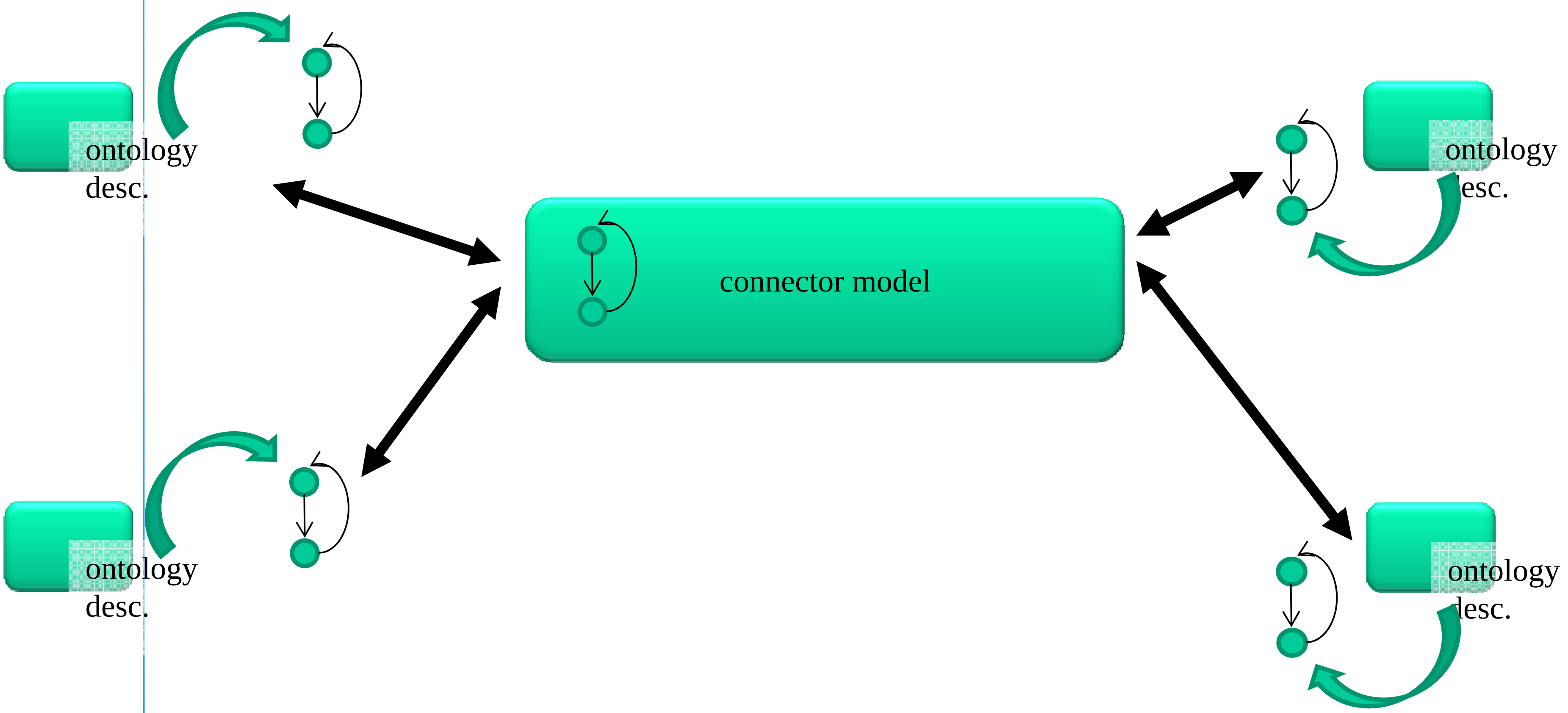
» The challenge

- 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

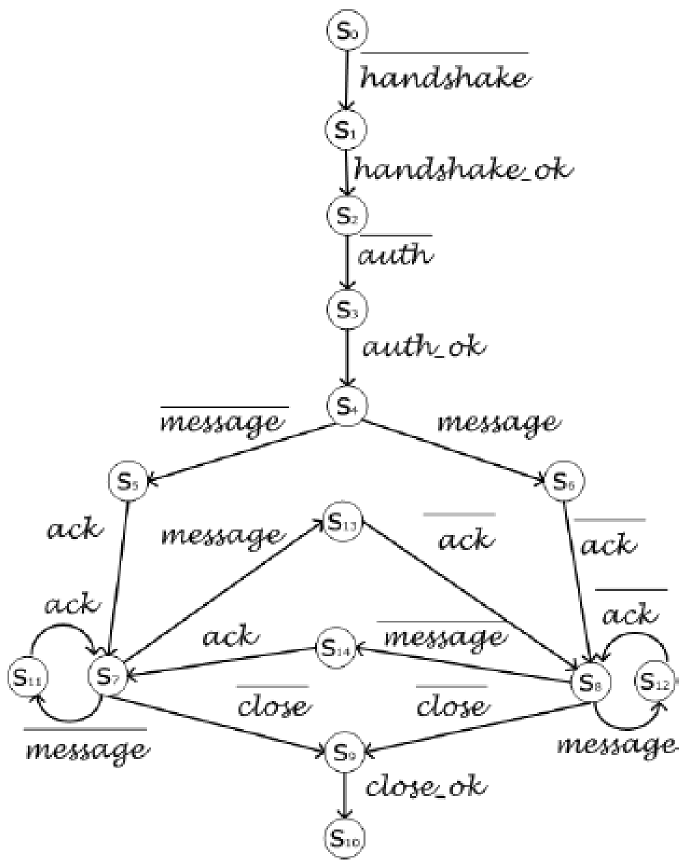
- necessary (but non-sufficient) conditions for protocol interoperability
  - > e.g., *"sharing the same intent"*
- data and functional mediations are assumed to be provided

» Mapping algorithm for the matching protocol models

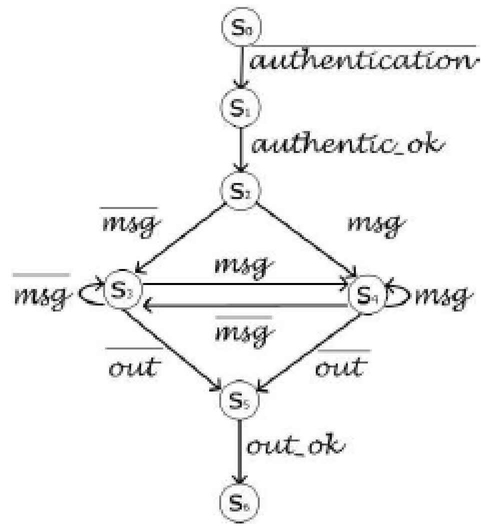
- sufficient (and "most permissive") conditions for protocol interoperability
  - > e.g., *"talking, at least partly, a common language"*
- a concrete mediator as final output

The instant messaging example

do they "*share the same intent*"?

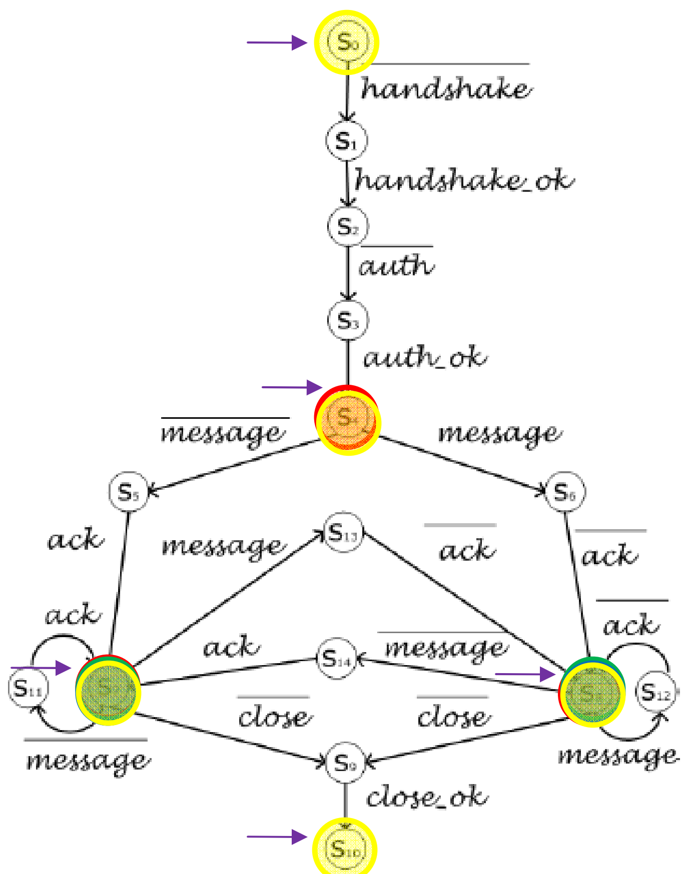


(a) Windows Messenger protocol

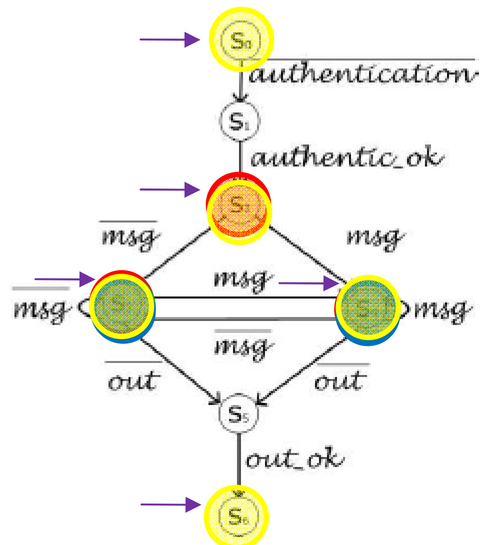


(b) Jabber protocol

The instant messaging example



(a) Windows Messenger protocol



(b) Jabber protocol

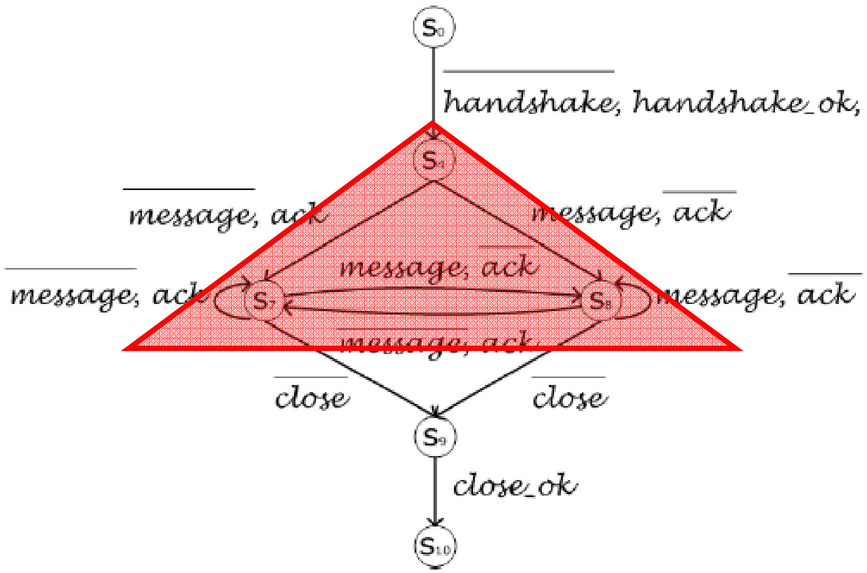
do they have similarities in the structure of their protocol models?

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

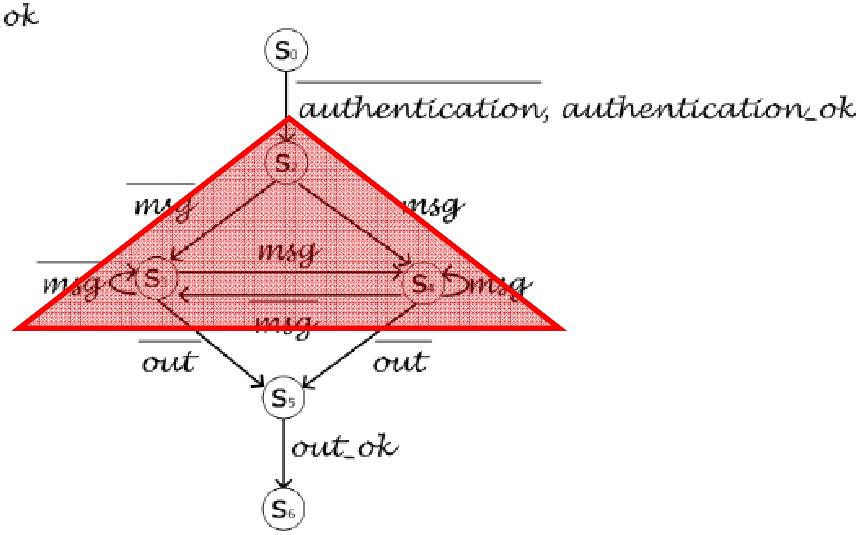
Common language structure

Ontology

"message, ack" <--> "msg"

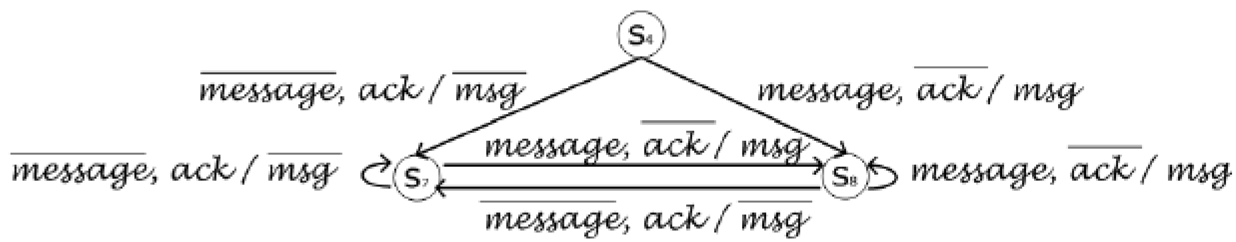


(a) Windows Messenger structure



(b) Jabber Messenger structure

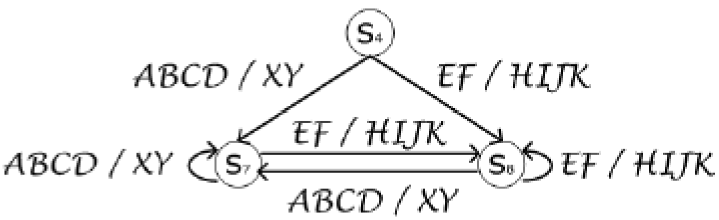
Abstract mediator model



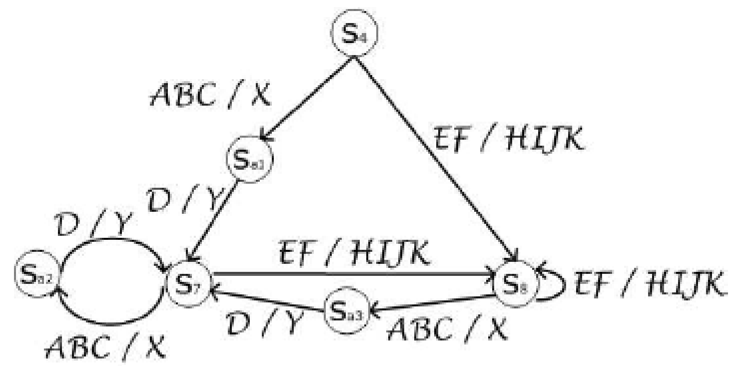
Indeed:

- 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" <- -> "X"
"D" <- -> "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 [outcome of a Dagstuhl Seminar] 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)

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).

Patrick H. S. Brito, Rogério de Lemos, Cecília M. F. Rubira: Development of Fault-Tolerant Software Systems Based on Architectural Abstractions. ECSA 2008: 131-147