Supporting the Composition of Distributed Business Processes

Paolo Traverso
ITC-IRST
Trento, Italy
traverso@itc.it

Joint work with: Marco Pistore,
Michele Trainotti, Piergiorgio Bertoli, Pierluigi Lucchese
Raman Kazhamiakin, Annapaola Marconi, Dmitry Shaparau,
Gabriele Zacco, Gaetano Calabrese, Fabio Barbon
and the ASTRO lab @ Trento

Context of the talk

- ***Starting Point:***

  web services as the universal basis for the integration of distributed business processes

- ***Challenge:***

  the automated composition of distributed business processes: effective, reliable, low-cost, and time-efficient composition

- ***Long Term Goal:***

  support the business process life-cycle, from design to execution

- ***Practical goal:***

  tools that automatically perform the time consuming and error prone tasks of analyzing business processes in detail, suggest solutions that can be adopted, refused, or refined by business analysts, designers, and programmers.

- ***Focus:***

  supporting synthesis , analysis and monitoring of distributed business processes.

Focus

- **Support at Design Time**
  - **Service Synthesis:**
    automatic generation of the web service compositions from their business requirements
  - **Service Analysis:**
    detect design and implementation errors wrt a set of business requirements
- **Support at Run-Time**
  - **Service Monitoring:**
    run-time (or simulation-time) analysis to detect violations of expected behaviors or to collect information about service executions

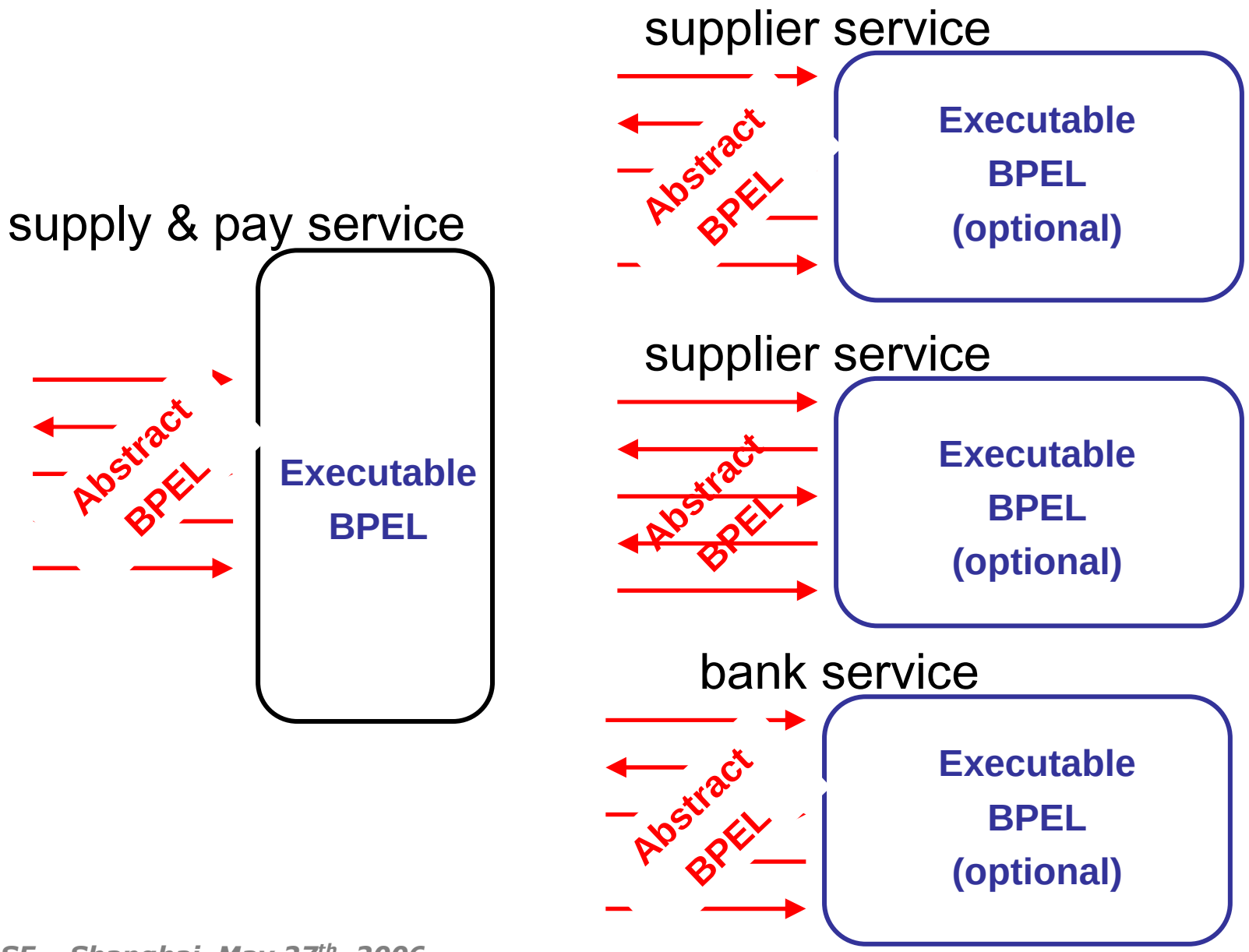
Outline

- *An approach to Service Composition*
  - *Some Work Hypothesis*
  - *Design Time: Synthesis & Analysis*
  - *Run Time: Monitoring*
  - *Some application scenarios*
  - *Demo*
- *Some other Approaches*
  - *Design Time: Synthesis and Analysis*
  - *Run Time: Monitoring*
- *Some Future Research Challenges*
  - *Design Time*
  - *Run Time*
  - *Continuous Design and Execution*

Some Working Hypotheses

- **BPEL (Business Process Execution Language)** as a Web Service description language
- BPEL offers a set of core process description concepts that permit to represent the **behavioral aspects of business process** interaction at two level of abstractions:
  - **Abstract BPEL** defines the interaction protocol (or Interface) of a service without exposing the internal behavior
  - **Executable BPEL** defines the actual code of a service; it can be deployed and executed on web service execution engines
- WS-**security**, WS-**reliability**, WS-**transactions** can be integrated within BPEL specifications
- Embedding in existing **business process development platforms**: Active WebFlow platform http://www.activebpel.org

Some Working Hypotheses: BPEL



Some Working Hypotheses: BPEL modeling

- **Difficulties in modeling BPEL processes**:
  - *BPEL is a full-powered, Turing-complete programming language*
  - *BPEL allows for specifying real time behaviors*

- **Difficulties in modeling communications among BPEL processes**:
  - *the interactions among processes is asynchronous*
  - *both outgoing and incoming messages are queued*
  - *there are multiple, unbounded queues*
  - *the order in which messages are received by a service may differ from the order in which the are consumed (message overpass)*

- **Implementation dependent behaviors**:
  - *real-time issues are engine-dependent*
  - *the details on the queue management is engine dependent*

Some Working Hypotheses: BPEL modeling

- **In theory:**
  - *It is impossible to decide correctness of BPEL processes…*
  - *… and hence to synthesize correct BPEL processes.*
  - *Correctness would be in any case engine-dependent.*
  - *Very inefficient modeling techniques due to the complex communication model and to the presence of real-time issues*

Some Working Hypotheses: BPEL in practice

- **In practice:**
  - No need to exploit the full power of BPEL:
    - *We want to model business processes, not algorithms!*
    - *BPEL is used to define the workflow, not the operations on data.*
    - *We can assume that data types are abstract (i.e., we do not specify their range)*
  - Robust BPEL compositions do not need complex queue mechanisms:
    - *No overpasses…*
    - *We can assume that messages are consumed as soon as they are received (synchronous communication model)*
  - Robust BPEL compositions should not depend on real-time behaviors:
    - *"business-level" timeouts are OK*
    - *critical runs are not*
- In practice, **BPEL processes translated into finite-state systems**

*Design Time: **Synthesis** of Compositions*

Automated generation of composite service
given a set of existing BPEL processes
and a composition requirement.

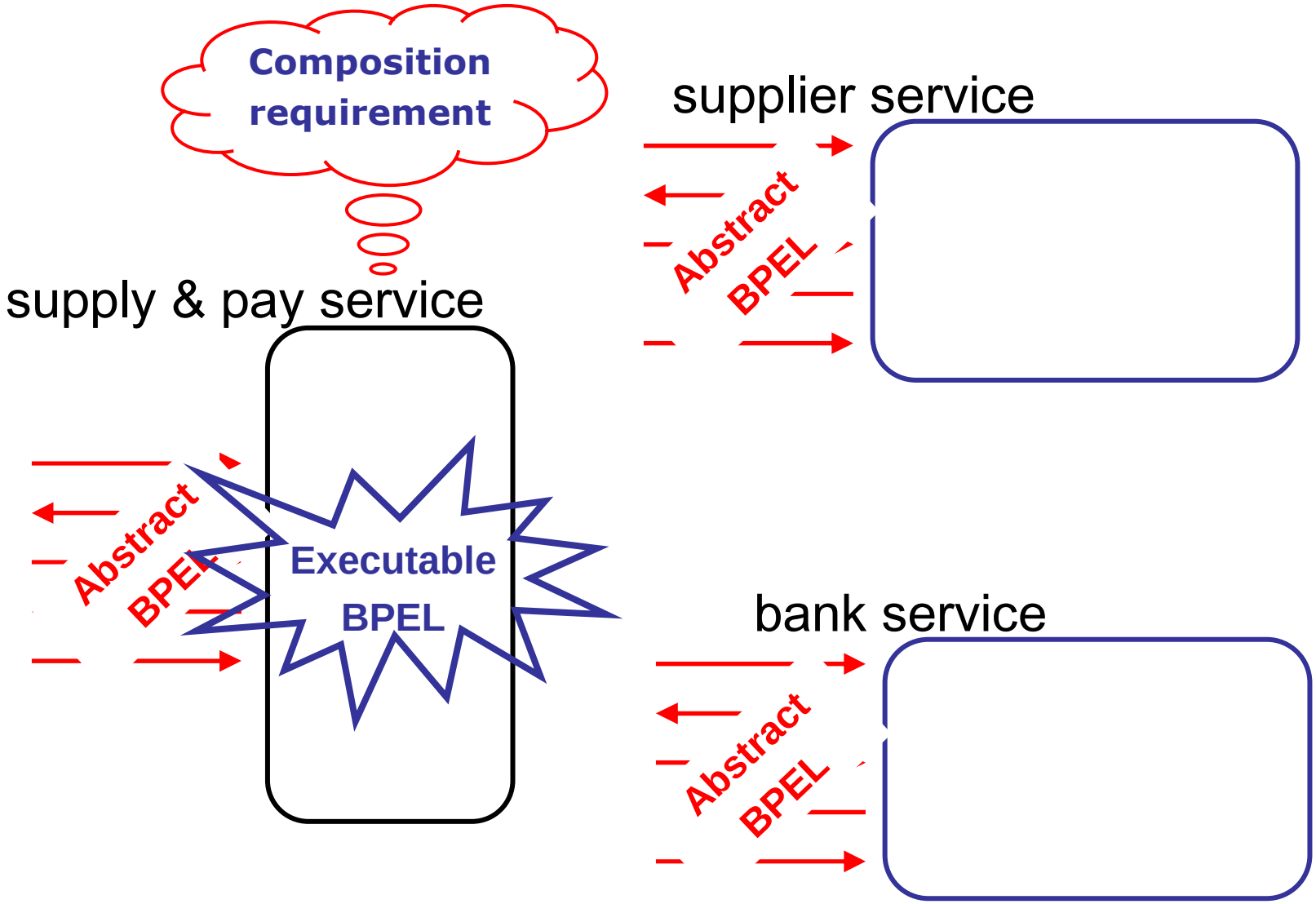
Inputs:

- *A set of component services (defined as Abstract BPEL interfaces)…*
- *A set of composition requirements*

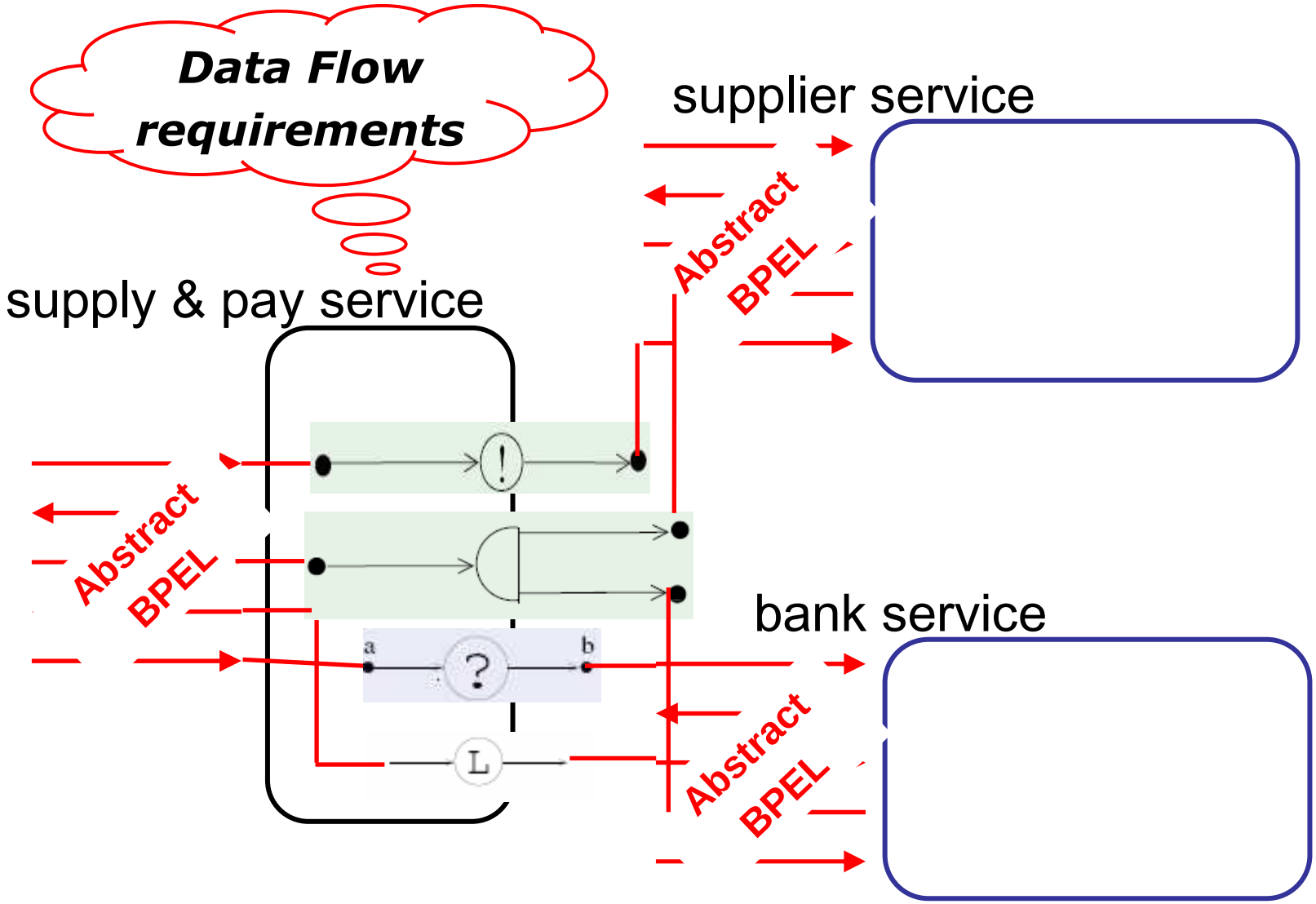
Output:

- *A composed service implementing the composition defined as an executable BPEL*

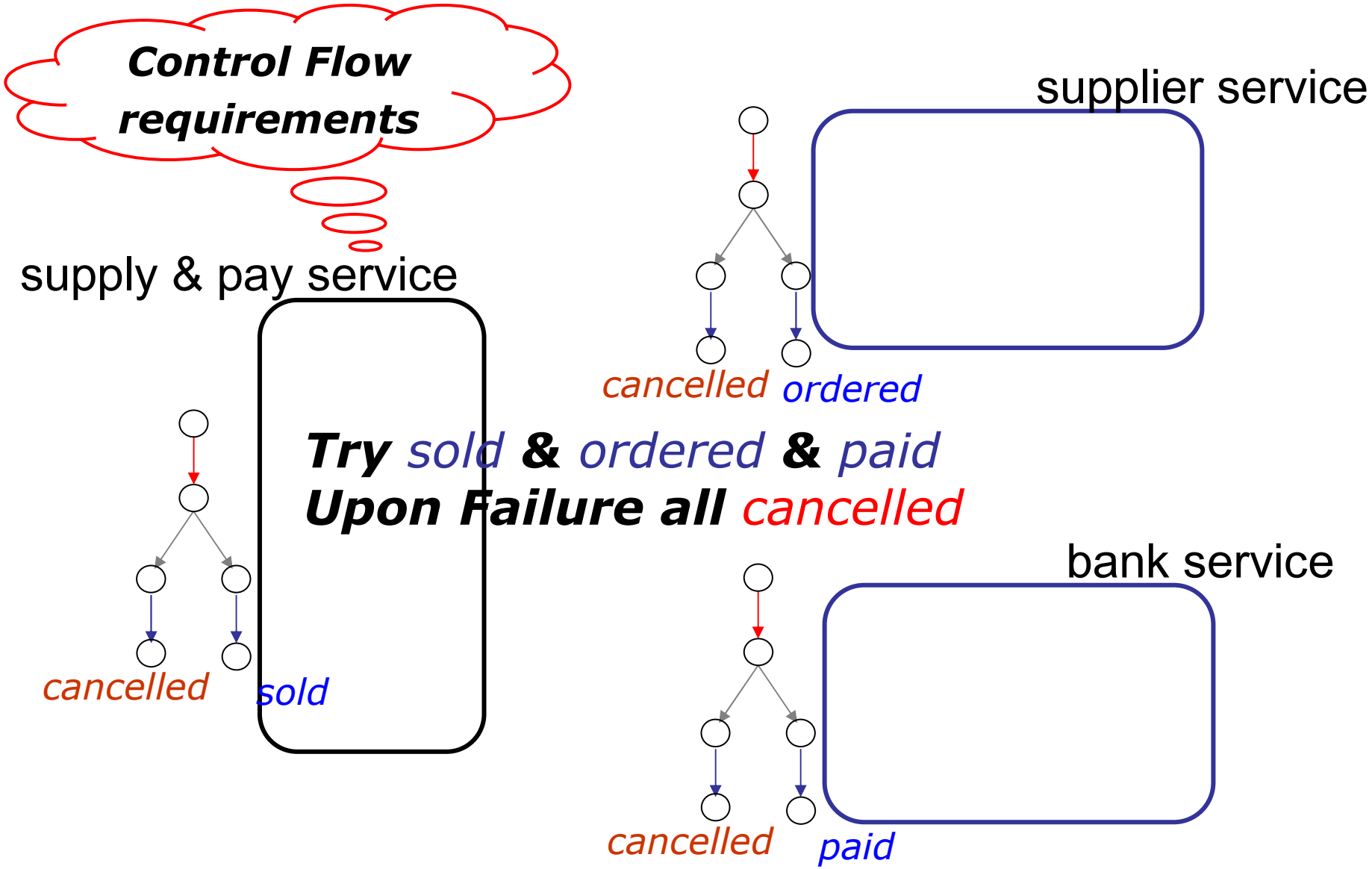
Design Time: *Synthesis* of Compositions



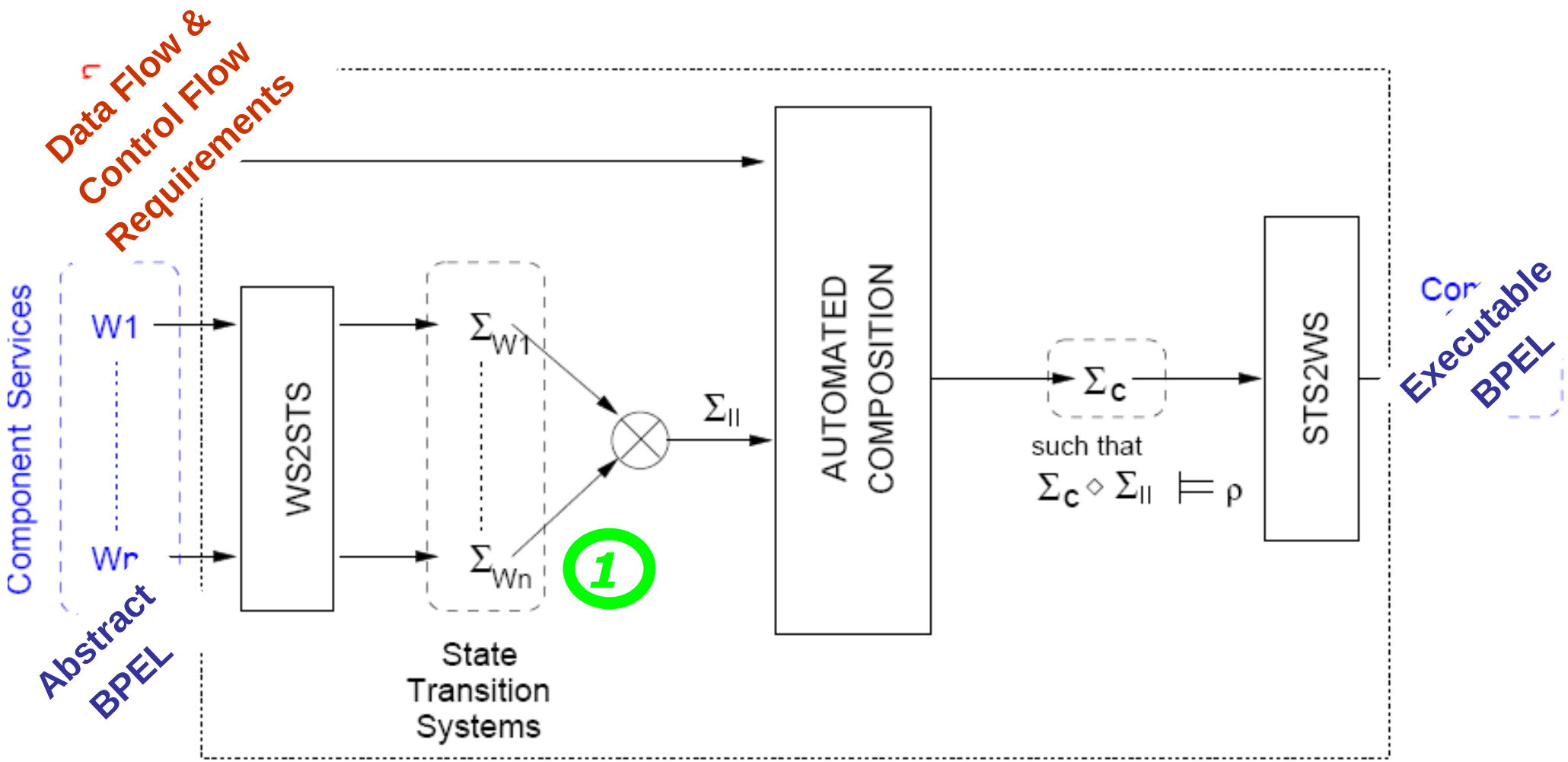
Synthesis: Data Flow & Control Flow Requirements



Synthesis: *Data Flow & Control Flow Requirements*

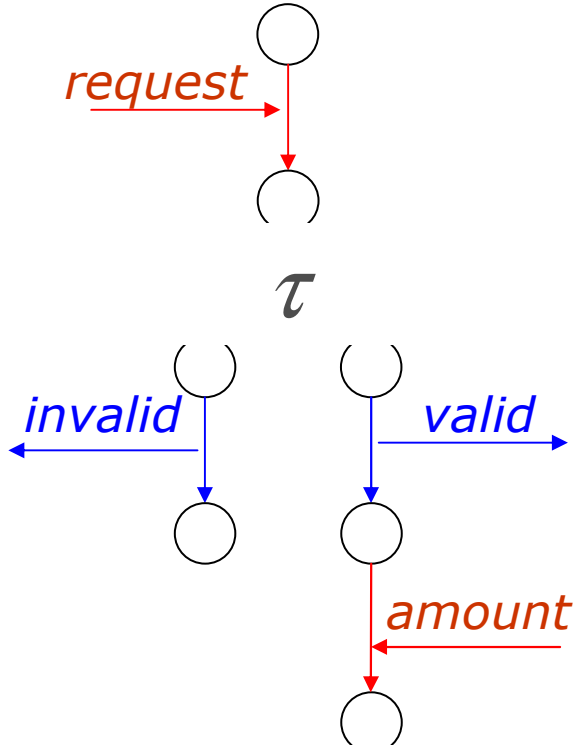


Design Time: *Synthesis* of Compositions



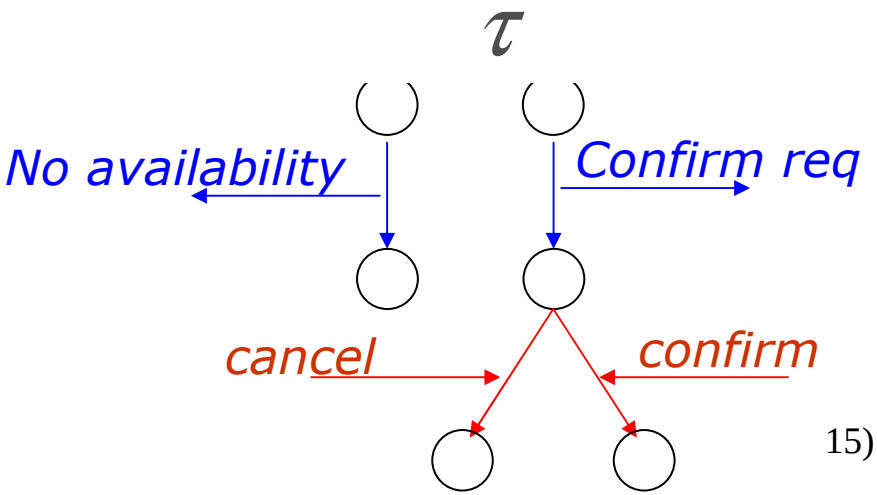
Synthesis: BPEL as state transition systems

- We translate Abstract BPEL processes into State Transition Systems
  - *Input actions $\mathcal{I}$* (reception of messages)
  - *Output actions $\mathcal{O}$* (message sent)
  - *Internal action τ* (internal evolutions that are not visible to external services)

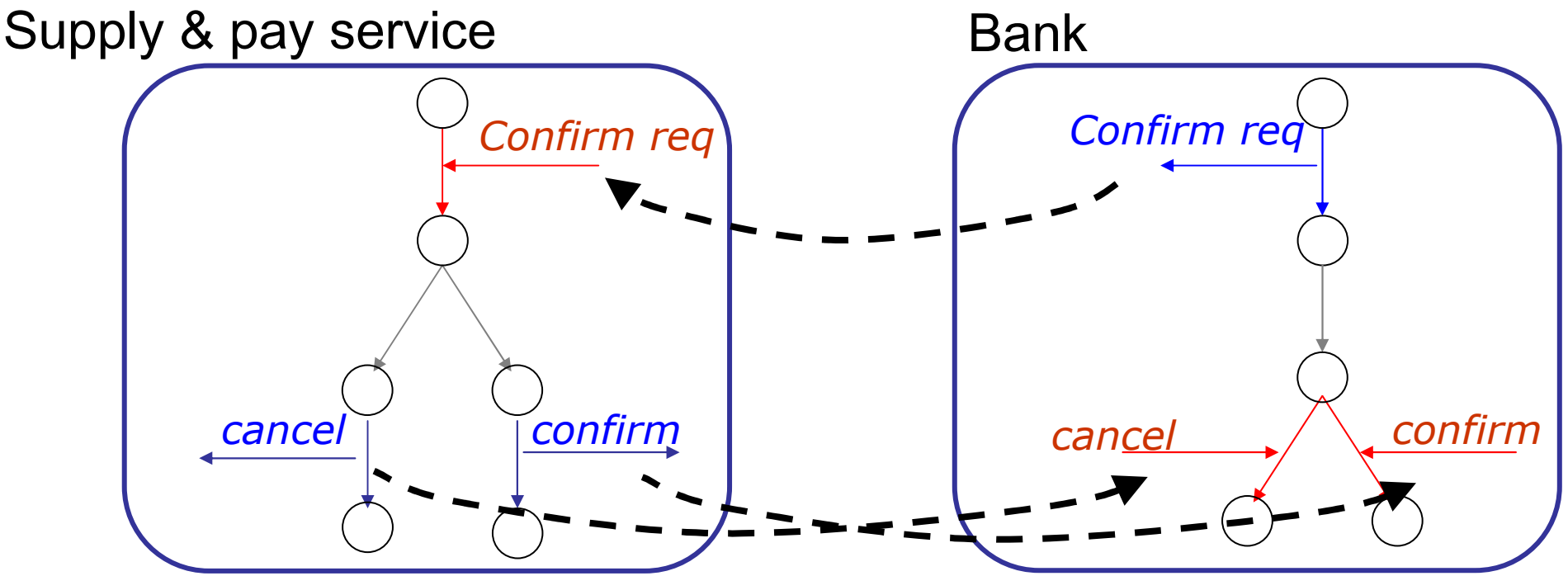


Definition. A state transition system Σ is a tuple $\langle \mathcal{S}, \mathcal{S}^0, \mathcal{I}, \mathcal{O}, \mathcal{R} \rangle$ where:

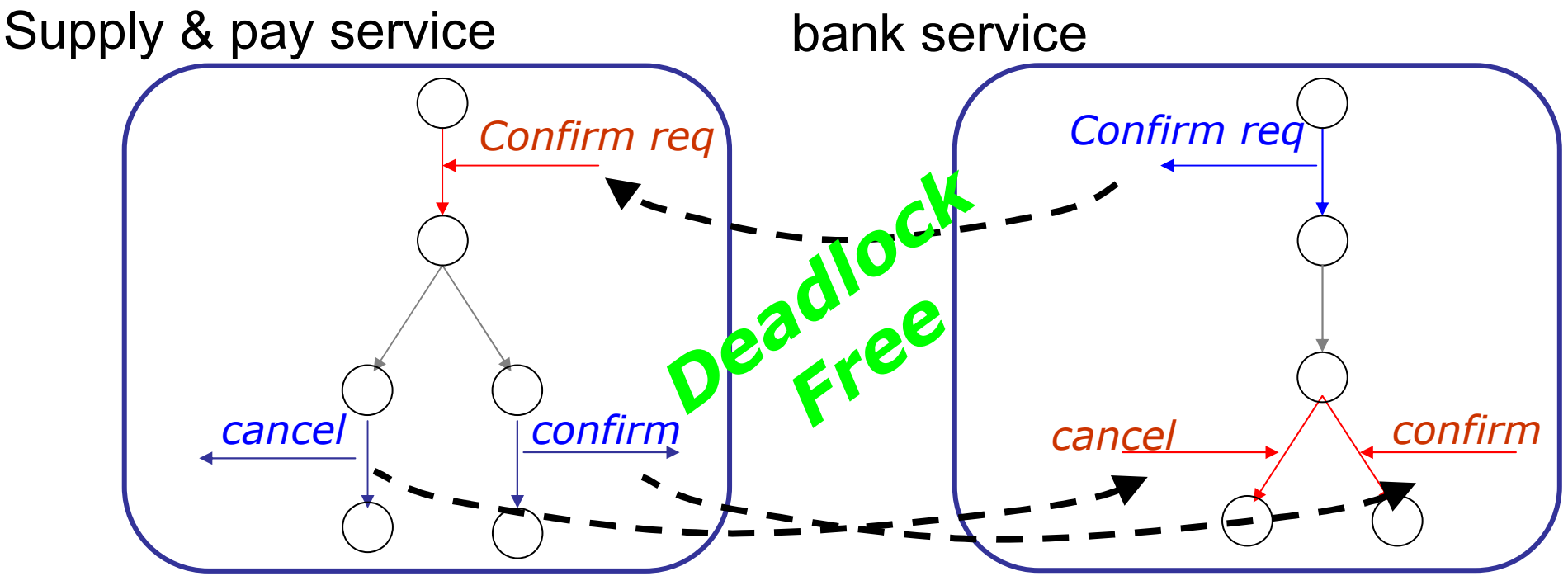
- $\mathcal{S}$ is the finite set of states;
- $\mathcal{S}^0 \subseteq \mathcal{S}$ is the set of initial states;
- $\mathcal{I}$ is a finite set of input actions;
- $\mathcal{O}$ is a finite set of output actions;
- $\mathcal{R} \subseteq \mathcal{S} \times (\mathcal{I} \cup \mathcal{O} \cup \{\tau\}) \times \mathcal{S}$ is the transition relation.



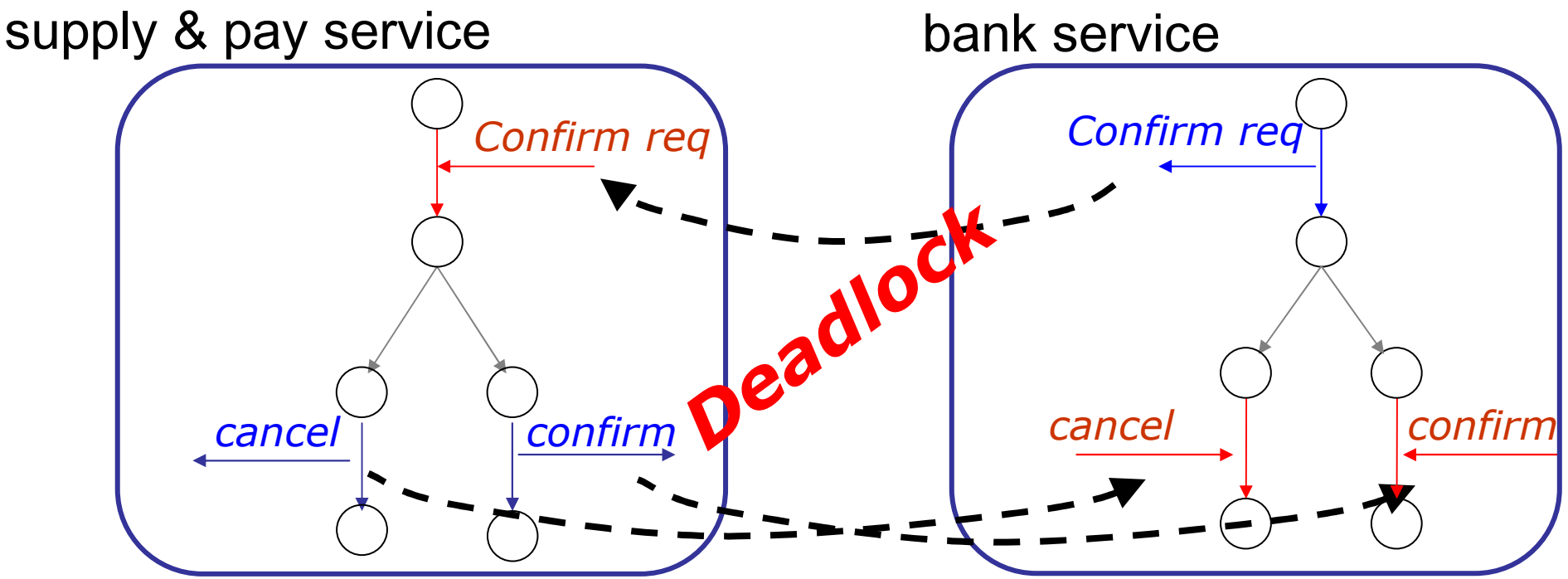
Synthesis: Controlled System



Synthesis: Deadlock free composition



Synthesis: Deadlock free composition



Design Time: **Synthesis** of Compositions

Automated generation of composite service given a set of existing BPEL processes and a composition requirement.

Inputs:

- *A set of component services (defined as Abstract BPEL interfaces)…*
- *A composition requirement*

Output:

- *A composed service implementing the composition defined as an executable BPEL*

Key aspects:

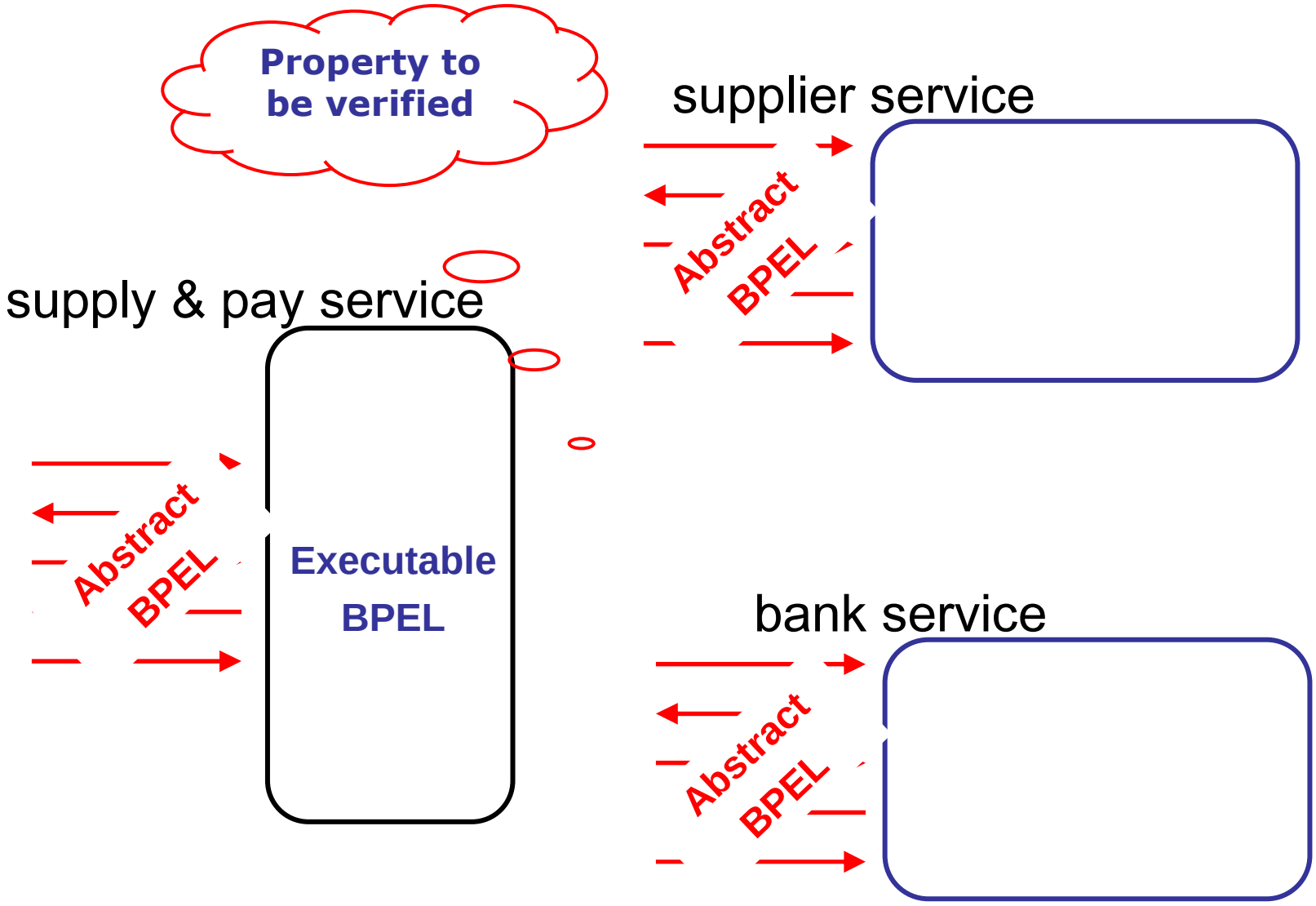
- *Expressive notations to define the composition requirements*
- *Advanced automated planning and synthesis techniques to manage:*
  - *control flow (non-determinism)*
  - *data flow (exchanged data values)*

Design Time: Analysis of Compositions

Design-time analysis of web service compositions

- Off-line verification of a set of (abstract and executable) BPEL processes
- Different kinds of requirements
  - Deadlock (livelock) freedom
  - Behavioral properties
  - Timed properties
- Verification using symbolic model checking techniques
  - Efficiency
  - Error traces are generated in case requirements are not satisfied

Design Time: *Analysis* of Compositions



Design Time: **Analysis** of Compositions

Design-time analysis of web service compositions

Inputs:

- *A set of component services (defined as Abstract BPEL interfaces)…*
- *A composed service (defined as Executable BPEL) …*
- *A property to be verified*

Output:

- *either "Verified!" + (example)*
- *"Error" + counterexample*

Design Time: Monitoring of Compositions

Run-time analysis of web service compositions

Inputs:

- *A set of component services (defined as Abstract BPEL interfaces)…*
- *A composed service (defined as Executable BPEL) …*
- *A monitoring specification*

Output:

- *Notification of violation of expected behavior …*
- *Notification of situations of interest …*
- *Aggregated/statistical information on process(es) behaviours*

Run Time: Monitoring of Compositions

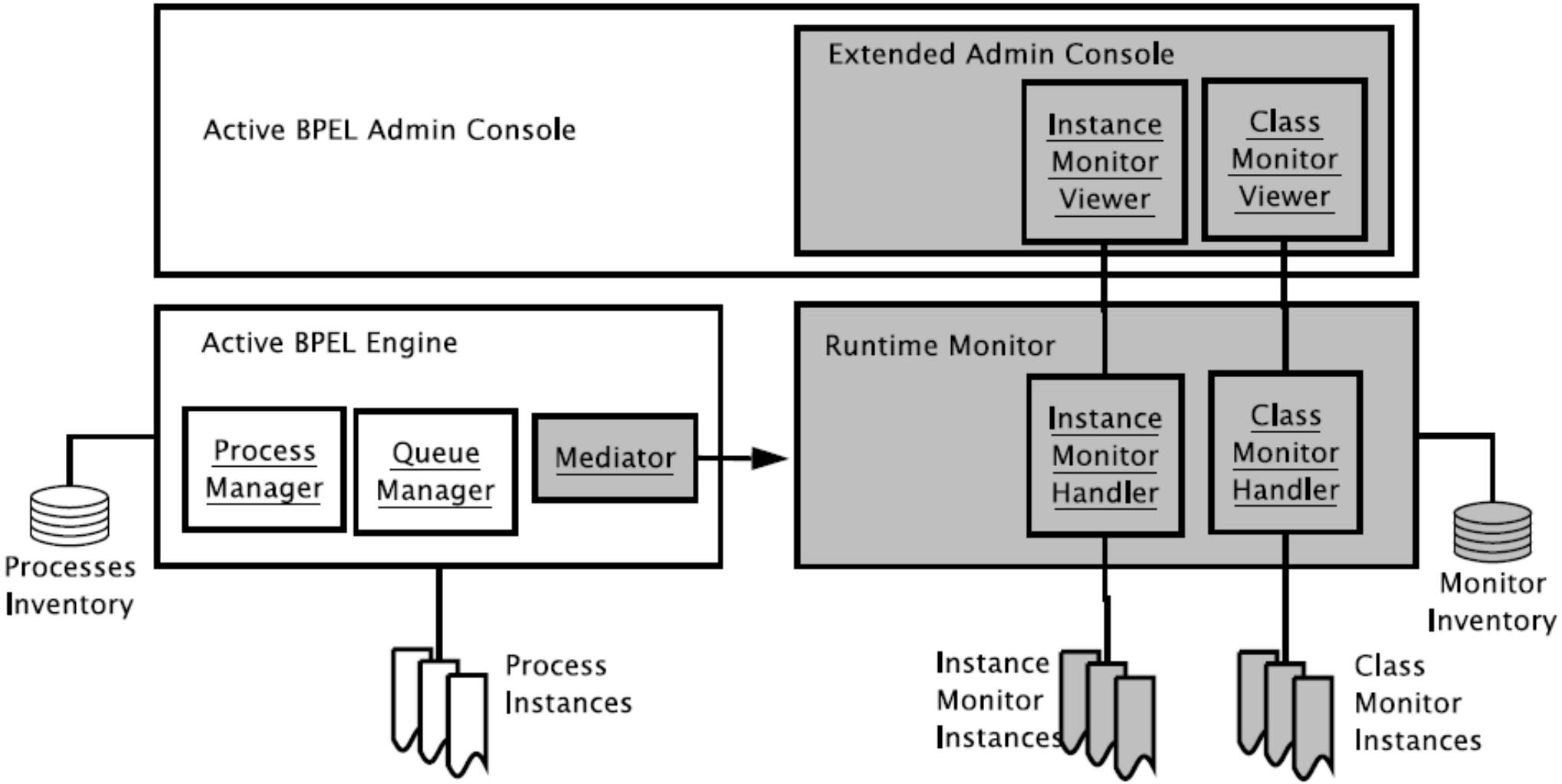
- Monitors are programs executed in parallel to BPEL processes
  - They intercept messages from/to these processes
  - They report boolean or statistical information

- **Instance monitor:** associated to a specific instance of a BPEL process
  - Violation of interaction protocols
  - Violation of functional requirements
  - Performance analysis

- **Class monitor:** associated to all instances of a BPEL process
  - Aggregated information on protocol/requirements violations
  - Statistical information on process behavior
  - QoS

- Monitors can be:
  - written by hand, or
  - generated from a high-level description of the condition to be detected

Run Time: Monitoring of Compositions



Monitor Specification: Instance Monitors

- **Boolean properties,** e.g.,
  - The credentials of the shop are not refused by the bank
  - The interaction with the bank does not start before the user has accepted an offer

- **Statistics,** e.g.,
  - Number of times a requested item is not available
  - Number of items offered before acceptance

- **Time properties,** e.g.,
  - Time to finalize the payment with the bank

$$
\begin{aligned}
b ::= {} & e \mid \mathsf{Y}\, b \mid \mathsf{O}\, b \mid \mathsf{H}\, b \mid b\, \mathsf{S}\, b \mid \\
& n = n \mid n > n \mid \neg b \mid b \wedge b \mid \mathsf{true} \\
n ::= {} & \mathsf{count}\,(b) \mid \mathsf{time}\,(b) \mid \\
& n + n \mid n - n \mid \; n * n \mid n/n \mid 0 \mid 1 \mid \ldots
\end{aligned}
$$

Monitor Specification: Class Monitors

- **Boolean properties,** e.g.,
  - The credentials of the shop are never refused by the bank
- **Statistics,** e.g.,
  - Average number of times user refuse an offer
- **Time properties,** e.g.,
  - Average time to process a payment with the bank

$$
\begin{aligned}
B ::= & \mathsf{And}\,(b) \mid \mathsf{Y}\,B \mid \mathsf{O}\,B \mid \mathsf{H}\,B \mid B\ \mathsf{S}\ B \mid \\
& N = N \mid N > N \mid \neg B \mid B \wedge B \mid \mathsf{true} \\
N ::= & \mathsf{Count}\,(b) \mid \mathsf{Sum}\,(n) \mid \\
& N + N \mid N - N \mid\ N * N \mid N/N \mid 0 \mid 1 \mid \ldots
\end{aligned}
$$

where b and n are instance monitor formulas.

The ASTRO lab @Trento (http://www.astroproject.org)

- ***Cluster of projects at IRST and University of Trento***
  Research Projects
  Verticalizations on application domains (e-Gov, Telco, Logistics …)

- ***European Projects:***
  SENSORIA – FET IP (Global Computing, Distributed Systems)
  KNOWLEDGE WEB – NoE (Semantic Web Services)
  APOSDLE – IP (Advanced Process-Oriented Learning Environments)
  …
- ***National Projects:***
  KLASE – FIRB (Knowledge Level Automated Software Engineering)
  STRAP – PRIN (AI and Services)
  ICAR – MIT (Interoperability and Cooperation for Italian Regions)
  …
- ***Industrial Projects and Collaborations***
  SAP, DeltaDator – e-Government
  MPS – e-Banking
  TELECOM, SESA – Telco
  OPERA21 – Logistics
  …

Some other approaches

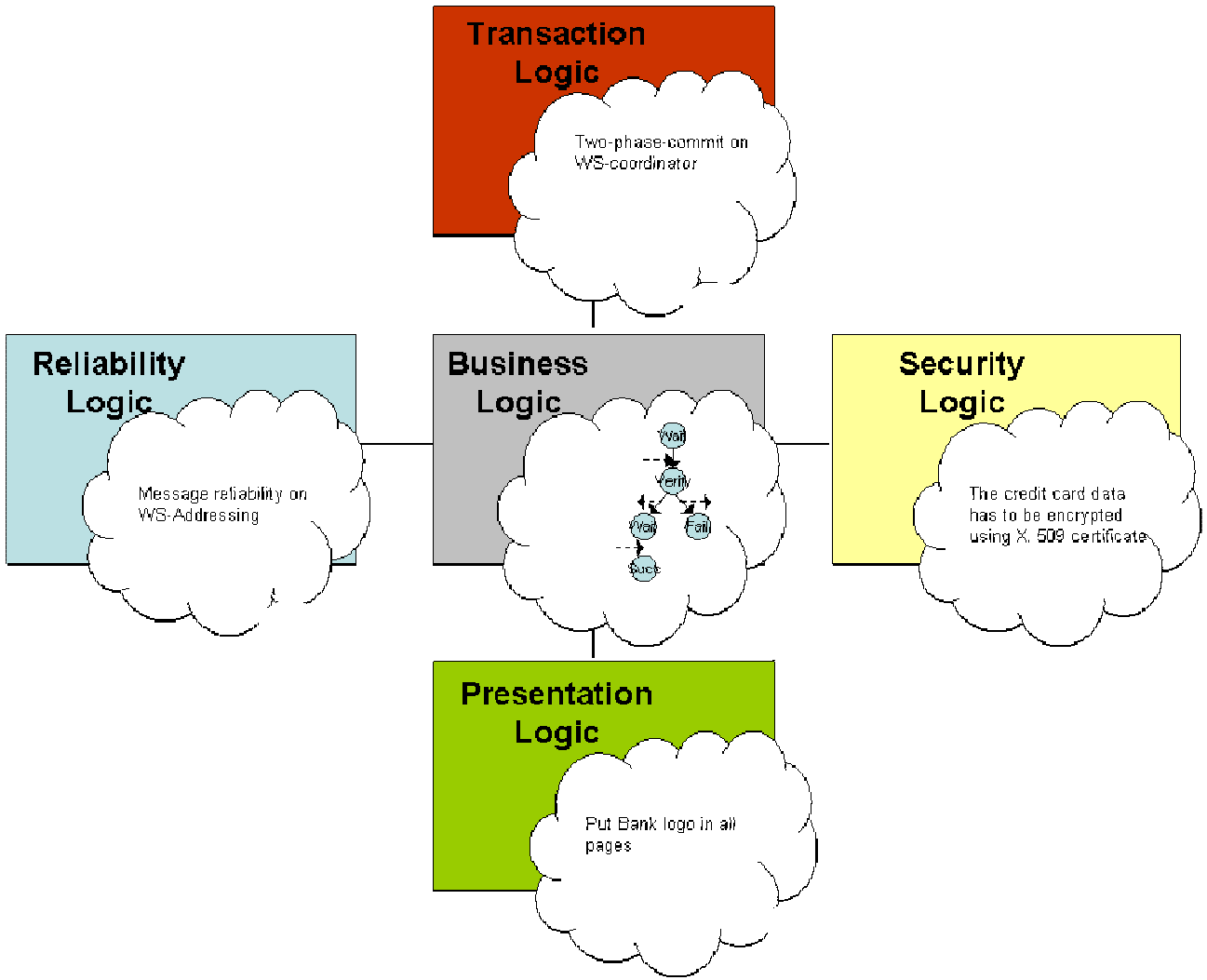
- **Synthesis**
  - Automata-based Approaches [Hull,Berardi]
  - Semantic Web Services [McIlraith,OWL-s,WSMO]
  - Planning for web services [Nau, Knoblock, …]
- **Analysis**
  - Automata-based approach [Bultan, …]
- **Monitoring**
  - Assertion-based monitoring [Baresi, Ghezzi & Guinea]
  - Requirement-based monitoring [Mahbub & Spanoudakis]
  - Service Level Agreement monitoring [Casati et al.]

Some Future Research Challenges

- **Present:** automated tools that perform time consuming and error prone tasks (e.g., detailed analysis, detect interaction problems, monitor execution step by step,...)
  - Synthesis
  - Analysis
  - Monitoring

- **Future**: supporting the life-cycle of Web services
  - Design-time (off-line):
    - **Aspect oriented service engineering**
  - Run-time:
    - **"Bounded" autonomics**

Design Time: "Aspects" of Requirements



Design Time: Aspects Oriented Software Engineering

- **Separate models** of the different "aspects" of each service
  - Business logics (central aspects)
  - Transactions, security, reliability…
  - SLA, rules, policies…
- **Composition** of aspect-oriented services
  - Composition of the business logics
  - Composition of the transactional behavior
  - Negotiation of SLAs
- **Deployment** of the executable services
  - BPEL
  - WS-Transaction
  - Monitors

Run Time: Autonomics

- **Autonomic systems**: systems able to adapt themselves without the intervention of humans
  - self-configuration
  - self-optimization
  - self-healing
  - self-adaptation
- **Autonomic services**: apply this concept to web services
- **At the moment**: focus on the "technical" aspects of service interactions
  - Detection of failures in external services
  - Automated optimization of number of trials
  - Load distribution among different service providers
  - …
- **Challenge**: move autonomics at the "service" level

Run Time: Autonomic Composition

- **Self-* of a service composition**
  - Given some business level requirements for the composition, automatically build the implementation:
    - combine in suitable ways the participating services (self-configuring composition)
    - guarantee the maximization of some expected reward (self-optimizing composition)
    - detect requirements that are no longer satisfied (self-healing composition)
    - adapt to unexpected changes in external services (self-adapting composition)

- **But, how much self-*?**
  - Unbounded autonomics is dangerous
  - Autonomic compositions should not take strategic decisions
  - The control should be in the hand of the analysts

- **Bounded autonomics:**
  - Set clear bounds to the self-* of a systems
  - Requirements are needed to define these bounds

Ongoing Collaborations

- Aspect-oriented service engineering
  - Monte Paschi di Siena
    - usage of aspects in the composition of e-shop applications
- Autonomic services
  - Massimo Paolucci (DocomoLab)
    - Semantic annotations for autonomic services
  - Telecom Italia Lab
    - Service composition in the Telco domain
  - Dagstuhl Seminar on "Automomic Services" (February 2007)
    - with Martin Wirsing (Munich), Amit Sheth (USA), Jana Koehler (IBM)

References (see also http://astroproject.org)

- **Synthesis**
  - P. Traverso and M. Pistore. Automated Composition of Semantic Web Services into Executable Processes. ISWC 2004
  - M. Pistore, P. Traverso, P. Bertoli, and A. Marconi. Synthesis of Composite BPEL4WS Web Services. ICWS 2005.
  - M. Pistore, A. Marconi, P. Bertoli, and P. Traverso. Automated Composition of Web Services by Planning at the Knowledge Level. IJCAI 2005.
  - M. Pistore, L. Spalazzi, and P. Traverso. A Minimalist Approach to Semantic Annotations for Web Processes Compositions. ESWC 2006.
- **Analysis**
  - M. Pistore, M. Roveri, and P. Busetta. Requirements-Driven Verification of Web Services. WSFM 2004.
  - R. Kazhamiakin, M. Pistore, and L. Santuari. Analysis of Communication Models in Web Service Compositions. WWW 2006.
- **Monitoring**
  - F. Barbon, P. Traverso, M. Pistore, and M. Trainotti. Run-Time Monitoring of the Execution of Plans for Web Service Compositions. ICAPS 2006.
  - F. Barbon, P. Traverso, M. Pistore, and M. Trainotti. Run-Time Monitoring of Instances and Classes of Web Service Compositions. ICWS 2006.

Conclusions

Thank you for your attention!