Synthesys and Composition of Web Services.

Marco Pistore
FBK-irst, Trento, Italy

June 3, 2009 - SFM-09:WS - Bertinoro

Services

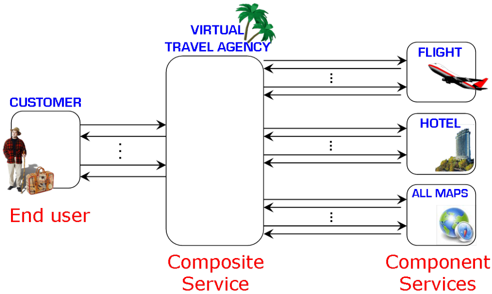
- **Service Oriented Computing:** new approach for building software applications by composing and configuring existing services.
- **Services:** software components devolped to be re-usable, which expose their definition and which are accessible by 3rd parties.
- **Web Services:** the most promising class of services, export their desciption and are accessible through standard network technologies
  - e.g. SOAP, WSDL, UDDI, WS-BPEL, WS-Transaction, ...

Web Service Composition

- **Web Service Composition:**
  combine existing services, available on the web, to provide added-value services featuring higher level functionalities.

Web Service Composition

- **Web Service Composition:**
combine existing services, available on the web, to provide added-value services featuring higher level functionalities.



Automated Web Service Composition

- Automatically compose a set of existing (component) services in order to satisfy some given composition requirements.

- what kind of requirements? what kind of components? which components? how to compose them? totally automatic? once and for all?...
- ..one definition, many different approaches.

Automated Web Service Composition

What kind of components?

- **Service-level** composition: components are atomic (request-response) services.
- **Process(flow)-level** composition: components are complex business workflows (control + data + QoS + security).

Automated Web Service Composition

What kind of requirements?

- **Control-flow** aspect: constraints on the execution of the composition (termination conditions, handling failures, transactional issues)
- **Data-flow** aspect: rule the flow and manipulation of messages within the composition (complex data structure, complex functions)

Automated Web Service Composition

What kind of requirements?

- **Control-flow** aspect: constraints on the execution of the composition (termination conditions, handling failures, transactional issues)
- **Data-flow** aspect: rule the flow and manipulation of messages within the composition (complex data structure, complex functions)
- **QoS** aspects: security, reliability, ..

Automated Web Service Composition

How to compose them?

- **Centralized** (mediated) composition: the result of the composition is a new service (mediator) that orchestrates the component services by properly exchanging messages.
- **Distributed** (peer2peer) composition: the execution of the composition is distributed among the component services.

Automated Web Service Composition

how to compose them?

- **Static** composition: services to be composed are decided at design time.
- **Dynamic** composition: run-time components selection, multiple-dynamic component instances.

Automated Web Service Composition

how to compose them?

- **Static** composition: services to be composed are decided at design time.
- **Dynamic** composition: run-time components selection, multiple-dynamic component instances.
- **Design-time** composition: design-time composition (and re-composition).
- **Run-time** composition: run-time re-composition, run-time adaptation.

Automated Web Service Composition

how to compose them?

- Requirements Specification

Automated Web Service Composition

how to compose them?

- **Requirements Specification**
- **Discovery**: dinamic search of the available services (public-private repositories) on the basis of (functional - non functional) composition requirements.

Automated Web Service Composition

how to compose them?

- **Requirements Specification**
- **Discovery**: dinamic search of the available services (public-private repositories) on the basis of (functional - non functional) composition requirements.
- **Selection**: among all the avilable services choose those participating to the composition (component services)

Automated Web Service Composition

how to compose them?

- **Requirements Specification**
- **Discovery**: dinamic search of the available services (public-private repositories) on the basis of (functional - non functional) composition requirements.
- **Selection**: among all the avilable services choose those participating to the composition (component services)
- **Composition**

Automated Web Service Composition

how to compose them?

- **Requirements Specification**
- **Discovery**: dinamic search of the available services (public-private repositories) on the basis of (functional - non functional) composition requirements.
- **Selection**: among all the avilable services choose those participating to the composition (component services)
- **Composition**
- **Monitoring**: detect events (failures, unexpected behaviors, unavailability, policy violation, ..) affecting the composition execution and react (alert, adapt, re-compose, ...) to changes.

Approaches to Web Service Composition

- **Berardi et Al.**
    - logic-based approach: WS composition as a satisfiability problem
    - require to fully specify the composition protocol
    - data flow requirements: message forwarding

Approaches to Web Service Composition

- **Berardi et Al.**
  - logic-based approach: WS composition as a satisfiability problem
  - require to fully specify the composition protocol
  - data flow requirements: message forwarding
- **Ponnekanti et Al.**
  - rule-based approach
  - component services are defined in terms of their inputs and outputs
  - rules indicate which outputs can be obtained given which inputs
  - can deal only with simple component services (atomic, deterministic)

Approaches to Web Service Composition

- **Berardi et Al.**
  - logic-based approach: WS composition as a satisfiability problem
  - require to fully specify the composition protocol
  - data flow requirements: message forwarding
- **Ponnekanti et Al.**
  - rule-based approach
  - component services are defined in terms of their inputs and outputs
  - rules indicate which outputs can be obtained given which inputs
  - can deal only with simple component services (atomic, deterministic)
- **Mc Ilraith et Al.**
  - AI planning-based approach (Golog - situation calculus)
  - Semantic Web community (OWL-S services)
  - automatically generate the composition protocol
  - services as black boxes, data aspect not supported

Approaches to Web Service Composition (cont.)

- **Wu, Sirin, et Al.**
  - AI planning-based approach (HTN planning)
  - Semantic Web community (OWL-S services)
  - complex services, scale well on large domains
  - composition requirements as tasks to be executed

Approaches to Web Service Composition (cont.)

- **Wu, Sirin, et Al.**
  - AI planning-based approach (HTN planning)
  - Semantic Web community (OWL-S services)
  - complex services, scale well on large domains
  - composition requirements as tasks to be executed
- **Brogi et Al.**
  - component services as YAWL workflows + semantic annotations
  - ontology-matching mechanisms to derive data-flow information linking operations of component services

Approaches to Web Service Composition (cont.)

- **Wu, Sirin, et Al.**
  - AI planning-based approach (HTN planning)
  - Semantic Web community (OWL-S services)
  - complex services, scale well on large domains
  - composition requirements as tasks to be executed
- **Brogi et Al.**
  - component services as YAWL workflows + semantic annotations
  - ontology-matching mechanisms to derive data-flow information linking operations of component services
- **Ambite, Knoblock and Takkar**
  - data matching techniques to dynamically compose atomic services
  - semantic annotations on service input-outputs
  - user query: provided inputs and requested outputs

Approaches to Web Service Composition (cont.)

- ...
- ...
- ...
- ...
- ...
- ...
- ...
- ...
- ...
- ...
- ...
- ...

In this Lecture

- Focus on one specific approach
  - the **ASTRO** approach developed in Trento
- Illustrate:
  - the **theoretical foundation** of the approach
  - the aspects related to its **practical application**
- Draw some **conclusions**
  - on the **ASTRO** approach
  - on the **usage of Formal Methods** for Web Services

Outline

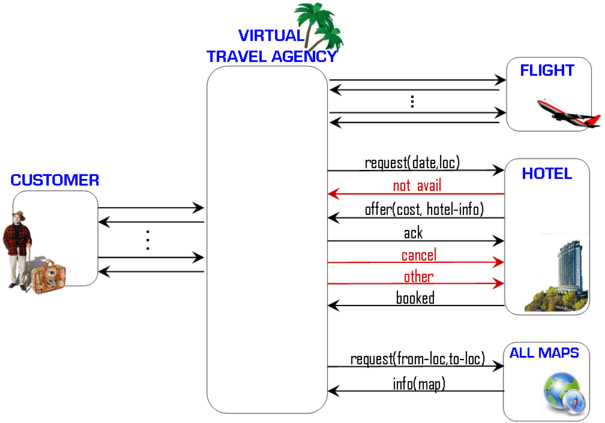
The ASTRO Automated Composition Approach

Current Composition Flavour

- **Centralized**: syntesize a ready to run new executable process.
- **Process-level**: components are complex and stateful workflows.
- **Design-time**: We have already selected the set of services we want to compose (no discovery, no selection)
- **Requirements**: both control and data flow requirements.

Web Service Composition in ASTRO

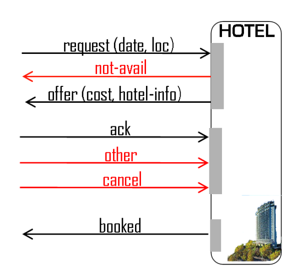
Web Services **are not necessarily atomic** (receive-response)

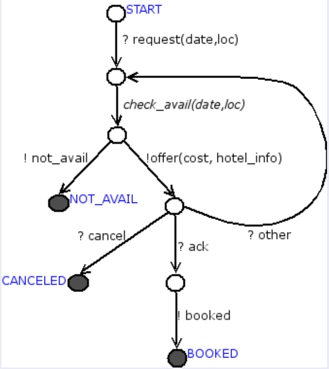


Web Service Composition in ASTRO

Web Services as **stateful business processes**

Hotel Service





WS-BPEL

Business Process Execution Language for WS

- Inspired by process algebras (pi-calculus) and by workflows (Petri-nets)
- Offers a set of core process description concepts to represent the behavioral aspects of business process interaction

WS-BPEL

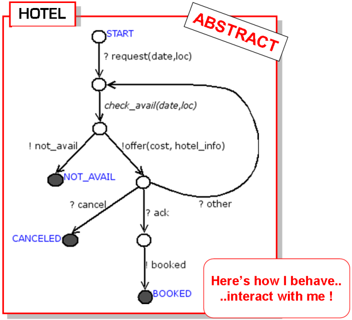
Business Process Execution Language for WS

- Inspired by process algebras (pi-calculus) and by workflows (Petri-nets)
- Offers a set of core process description concepts to represent the behavioral aspects of business process interaction
- **Communication constructs:**
  - interacting with external services by receiving and sending messages (both asynch. and synch.)
- **Control flow constructs:**
  - data manipulation, sequential execution, conditional branching, iterate execution, guarded choice, parallel execution
- **Advanced features:**
  - handling faults and out-of-band events, recover from failure (rollback)

WS-BPEL: abstract vs executable processes

WS-BPEL abstract process

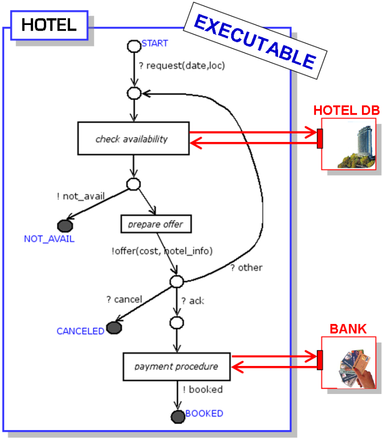
- define the interaction protocol
- hide implementation and personal details



WS-BPEL: abstract vs executable processes

WS-BPEL executable process

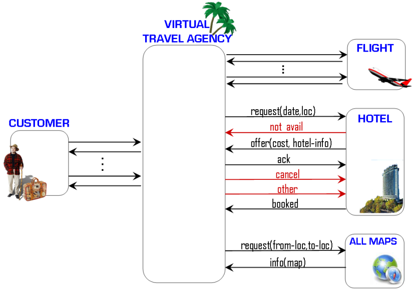
- fully specify the behavior of the process
- is **one** possible implementation of the published (abstract) protocol
- ready to be deployed and run



The ASTRO Automated Composition Approach

What is the composition input?

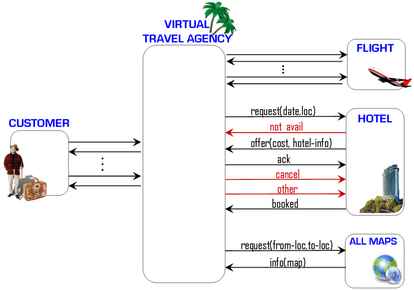
- The WSDL and abstract BPEL of the "component" services (including the end user)



The ASTRO Automated Composition Approach

What is the composition input?

- The WSDL and abstract BPEL of the "component" services (including the end user)



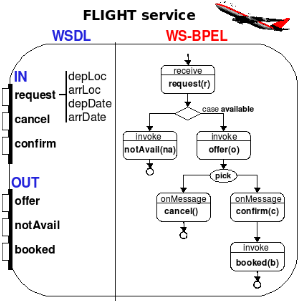
- The requirements specifying the expected behaviour of the composite service

The ASTRO Automated Composition Approach

What kind of components?

- Business processes described with WS standards (WSDL and WS-BPEL).

- Complex stateful protocols
- Non-deterministic, partial observable behavior
- Asynchronous interactions
- Complex data and expressions



The ASTRO Automated Composition Approach

What kind of composition requirements?

- **Data-flow aspect**:
  - Forwarding messages, data mediation, internal computation

The ASTRO Automated Composition Approach

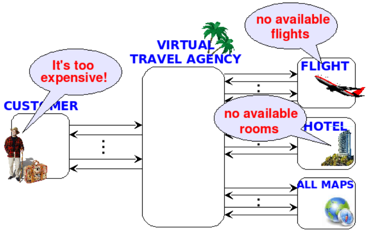
What kind of composition requirements?

- **Data-flow aspect**:
  - Forwarding messages, data mediation, internal computation
- **Control-flow aspect**:
  - Termination conditions, including failure handling

The ASTRO Automated Composition Approach

What kind of composition requirements?

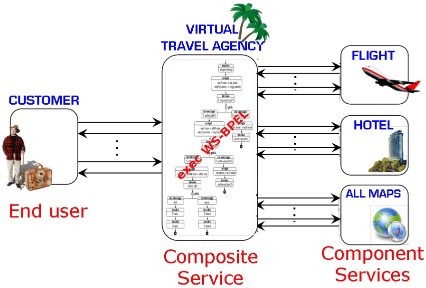
- **Data-flow aspect**:
  - Forwarding messages, data mediation, internal computation
- **Control-flow aspect**:
  - Termination conditions, including failure handling



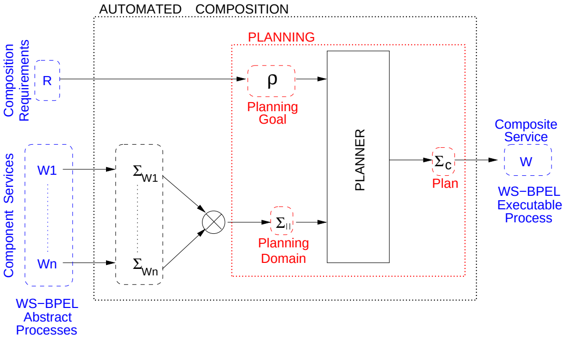
The ASTRO Automated Composition Approach

What is the composition outcome?

- A ready to run **executable process** described with WS standards (WSDL and WS-BPEL).



The ASTRO Automated Composition Approach



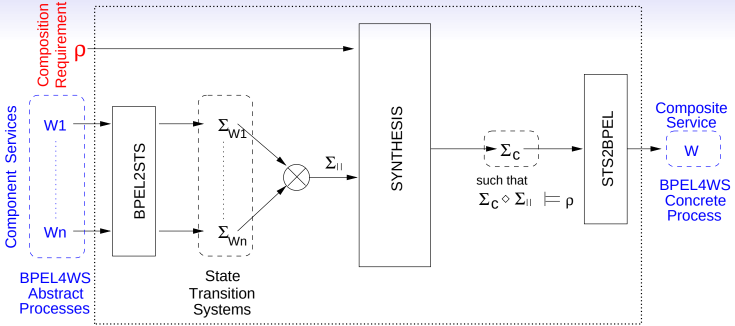
ASTRO automated composition approach

Sophisticated AI planning techniques

- asynchronous domains, non-determinism, partial observability
- complex goals: preferences and recovery conditions

Outline

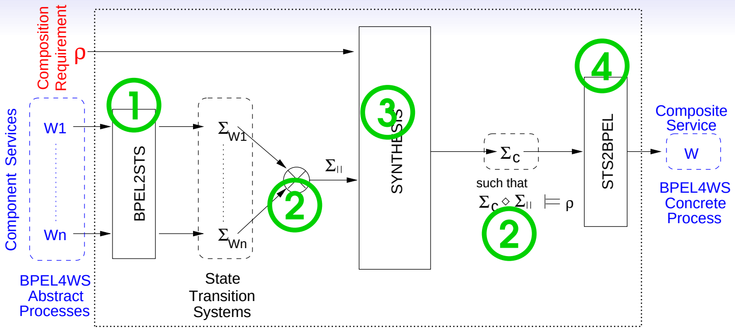
Theoretical Framework: Goal



GOAL: define a theoretical framework for web service composition which:

- allows for an **efficient automated generation** of the composite service
- is **compliant to web service execution engines**

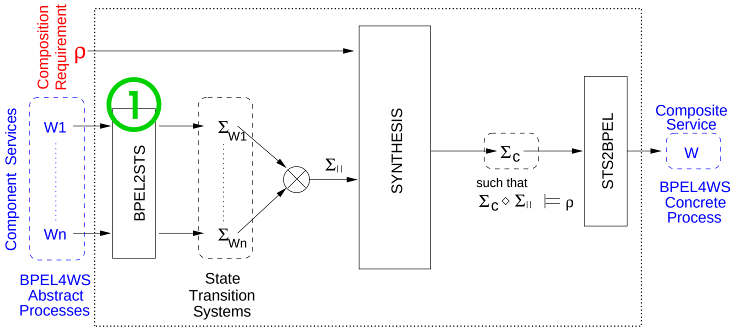
Key issues



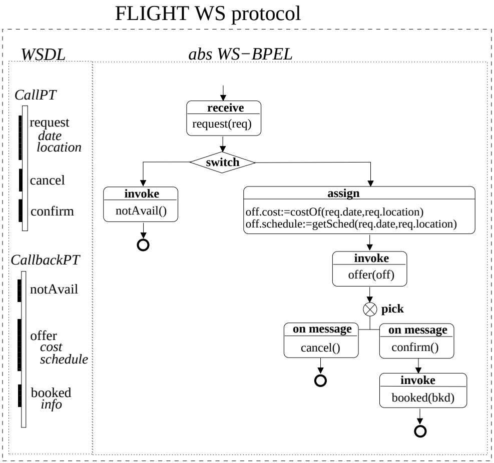
Key issues in the definition of the framework:

1. How to map BPEL4WS into (finite-state) state transition systems
2. How to model interactions among BPEL4WS processes
3. How to automatically synthesize the composition
4. How to map the composite STS into BPEL4WS

Mapping BPEL4WS into STS



BPEL4WS: Example



BPEL4WS to STS

- **In theory** BPEL4WS **cannot** be translated into finite-state systems, since it is a Turing complete language.
- **In practice**:
  - business process modeling requires to model the workflow (business process) separately from the operations on data (business rules);
  - BPEL4WS is used to define the workflow, not the operations on data;
  - business process composition can be done at the workflow level, independently from the business rules.

  That is, in web service composition we can assume that:
  - data types are abstract (i.e., we do not specify their range);
  - functions are uninterpreted (we do not specify the operations they perform).

  Under this assumption, BPEL4WS can be translated into finite-state systems.

State Transition Systems

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.

Example of State Transition System

```
SERVICE Flight

TYPES
    dateT: ABSTRACT
    locationT: ABSTRACT
    costT: ABSTRACT
    scheduleT: ABSTRACT
    ...
    boolean: {T,F}

VARIABLES
    req_date: dateT
    req_location: locationT
    off_cost: costT
    off_schedule: scheduleT
...
    available: boolean

FUNCTIONS
    costOf: (dateT,locationT): costT
    getSched: (dateT,locationT): scheduleT
```

```
INPUTS
    request(req_date,req_location)
    cancel()
    confirm()
    ...

OUTPUTS
    offer(off_cost,off_schedule)
    ...

LOCATIONS
    pc: l1,l2,l3,....

TRANSITIONS
    pc=l1 -[INPUT request(req_date,req_location)]-> pc:=l2
    pc=l2 -[TAU]-> pc:=l3
    pc=l3 AND available=T -[TAU]-> pc:=l4
    ...
    pc=l7 -[OUTPUT offer(off_cost,off_schedule)]-> pc:=l8
    pc=l8 -[INPUT cancel()]-> pc:=l9
    ...
    pc=l8 -[INPUT confirm()]-> pc:=l10
    ...
```

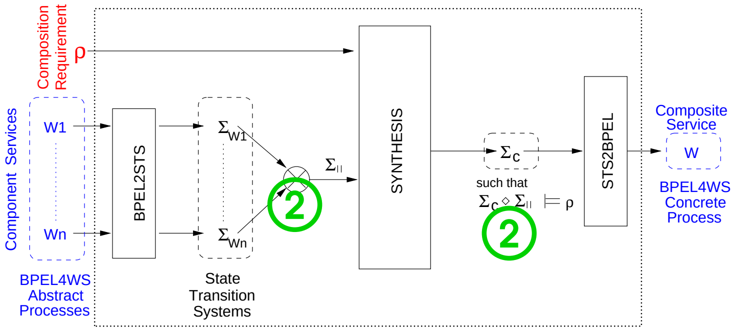
Example of State Transition System

Remarks:

- The STS just described is parametric wrt the definition of the ASBTRACT data types:
  - To define a finite-state STS, a finite set of values has to be associated to such data types
    - Singletons are not enough (service outputs can be predicted)
    - Large sets affect the performance
  - Pragmatic rule: two values per data type

- The STS just described is parametric wrt the definition of FUNCTIONS:
  - Once the data types are finitized, the definition of functions can be modeled as a set of static variables

Interactions among BPEL4WS processes



Interactions among BPEL4WS processes

In existing BPEL4WS engines:

- the interactions among processes is **asynchronous**: both outgoing and incoming messages are queued;
- the **order** in which messages are received by a service may differ from the order in which the are consumed (message overpass);
- the details on the queue management is **engine dependent.**

If we model explicitly asynchronous interactions and message queues:

- the automated generation of the composition becomes terribly inefficient
- the composition becomes engine dependent

In our framework:

- we model interactions as synchronous communications ($\Rightarrow$ **efficiency**)
- we define conditions under which this synchronous model is is adequate to execution engines ($\Rightarrow$ **correctness**)

A synchronous model of process interactions

Definition. Let $\Sigma_1 = \langle \mathcal{S}_1, \mathcal{S}_1^0, \mathcal{I}, \mathcal{O}, \mathcal{R}_1 \rangle$ and $\Sigma_2 = \langle \mathcal{S}_2, \mathcal{S}_2^0, \mathcal{O}, \mathcal{I}, \mathcal{R}_2 \rangle$ be two complementary state transition systems.

The **closed STS** $\Sigma_1 \triangleright \Sigma_2$ is defined as:

$$\Sigma_c \triangleright \Sigma = \langle \mathcal{S}_1 \times \mathcal{S}_2, \mathcal{S}_1^0 \times \mathcal{S}_2^0, \emptyset, \emptyset, \mathcal{R}_1 \triangleright \mathcal{R}_2, \rangle$$

where $\langle (s_1, s_2), \tau, (s_1', s_2') \rangle \in (\mathcal{R}_1 \triangleright \mathcal{R}_2)$ if

- $\langle s_1, \tau, s_1' \rangle \in \mathcal{R}_1$ and $s_2 = s_2'$;
- $\langle s_2, \tau, s_2' \rangle \in \mathcal{R}_2$ and $s_1 = s_1'$;
- $\langle s_1, a, s_1' \rangle \in \mathcal{R}_1$ and $\langle s_2, a, s_2' \rangle \in \mathcal{R}_2$ with $a \in \mathcal{I} \cup \mathcal{O}$.

Correctness wrt the execution engines

- When executed on existing engines, the sender can emit a message also if the receiver is not ready to consume it.
- To guarantee the correctness of the synchronous model wrt the execution engines, we require that:
  - the message is eventually consumed by the receiver (**no message is lost**)
  - no other message is sent or received before that message is consumed (**no overpasses**)

  A composition satisfying the requirements above is said **deadlock-free**.
- This corresponds to require that the composition is **robust wrt the relative speed** of the processes **and wrt critical runs** of messages.
- In a deadlock-free composition, the synchronous model and the real executions differ only for (irrelevant) details on the precise moment a message is emitted.

Deadlock free composition

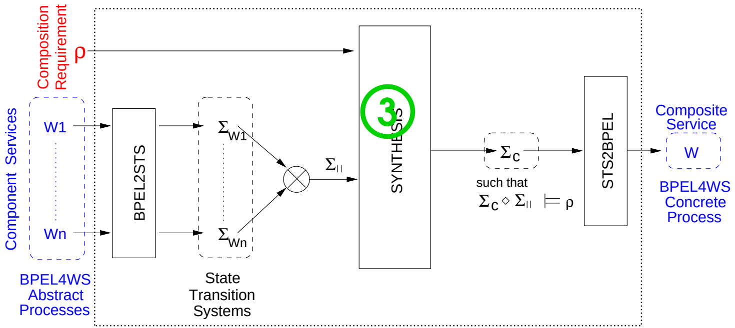
Definition. Let $\Sigma_1 = \langle \mathcal{S}_1, \mathcal{S}_1^0, \mathcal{I}, \mathcal{O}, \mathcal{R}_1 \rangle$ and $\Sigma_2 = \langle \mathcal{S}_2, \mathcal{S}_2^0, \mathcal{O}, \mathcal{I}, \mathcal{R}_2 \rangle$ be two STS.

The closed system $\Sigma_1 \triangleright \Sigma_2$ is said to be **deadlock free** if all states $(s_1, s_2) \in \mathcal{S}_1 \times \mathcal{S}_2$ satisfy the following conditions:

- if $\langle s_1, a, s_1' \rangle \in \mathcal{R}_1$ with $a \in \mathcal{O}$ then there is some $s_2' \in \tau\text{-closure}(s_2)$ such that $\langle s_2', a, s_2'' \rangle \in \mathcal{R}_2$ for some $s_2'' \in \mathcal{S}_2$;
- if $\langle s_2, a, s_2' \rangle \in \mathcal{R}_2$ with $a \in \mathcal{I}$ then there is some $s_1' \in \tau\text{-closure}(s_1)$ such that $\langle s_1', a, s_1'' \rangle \in \mathcal{R}_1$ for some $s_1'' \in \mathcal{S}_1$.

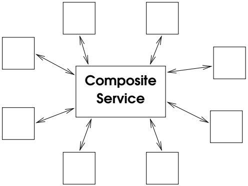
With $\tau\text{-closure}(s)$ we denote the set of states reachable from s performing transitions labelled by τ.

Automated synthesis



Automated synthesis

- We assume a star architecture: **there is no interaction among the component services**, but only between them and the composite service:



- Under this assumption, the starting point of the composition is the **parallel product** $\Sigma_{\|} = \Sigma_1 \parallel \Sigma_2 \parallel \cdots \parallel \Sigma_n$ of the component services.

- **Automated synthesis:** given $\Sigma_{\|}$ and composition goal ρ, find a STS Σ_c such that $\Sigma_c \triangleright \Sigma_{\|}$ is deadlock free and $\Sigma_c \triangleright \Sigma_{\|} \models \rho$.

Parallel product of STS

Definition. Let $\Sigma_1 = \langle \mathcal{S}_1, \mathcal{S}_1^0, \mathcal{I}_1, \mathcal{O}_1, \mathcal{R}_1 \rangle$ and $\Sigma_2 = \langle \mathcal{S}_2, \mathcal{S}_2^0, \mathcal{I}_2, \mathcal{O}_2, \mathcal{R}_2 \rangle$ be two STSs with $(\mathcal{I}_1 \cup \mathcal{O}_1) \cap (\mathcal{I}_2 \cup \mathcal{O}_2) = \emptyset$.

The **parallel product** $\Sigma_1 \parallel \Sigma_2$ of Σ_1 and Σ_2 is defined as:

$$\Sigma_1 \| \Sigma_2 = \langle \mathcal{S}_1 \times \mathcal{S}_2, \mathcal{S}_1^0 \times \mathcal{S}_2^0, \mathcal{I}_1 \cup \mathcal{I}_2, \mathcal{O}_1 \cup \mathcal{O}_2, \mathcal{R}_1 \| \mathcal{R}_2 \rangle$$

where:

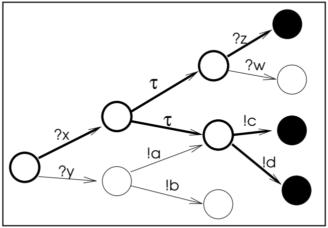
- $\langle (s_1, s_2), a, (s_1', s_2) \rangle \in (\mathcal{R}_1 \| \mathcal{R}_2)$ if $\langle s_1, a, s_1' \rangle \in \mathcal{R}_1$;
- $\langle (s_1, s_2), a, (s_1, s_2') \rangle \in (\mathcal{R}_1 \| \mathcal{R}_2)$ if $\langle s_2, a, s_2' \rangle \in \mathcal{R}_2$.

Synthesis: reachability goal

Reachability goal: ρ expresses a condition that has to hold in all final states reached by executing the composite service.

Synthesis: find a sub-graph of the STS $\Sigma_{||}$ which satisfies the following conditions:

- all terminal states satisfy condition ρ
- if a state belongs to the sub-graph, then all states reachable via τ and output transitions belong to the sub-graph
- there are no loops (**strong** solution)

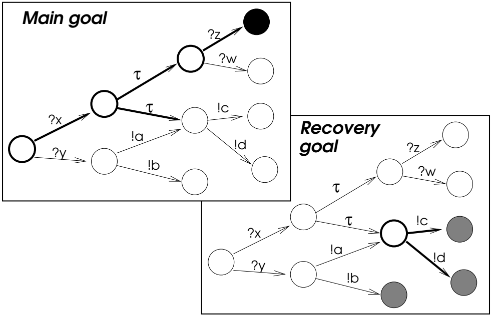


Synthesis: recovery goal

Reachability goal with recovery condition: $\rho = \text{TryReach } p \text{ Fail DoReach } q$, where p is the main goal and q is the recovery goal.

Synthesis:

- two copies of the STS $\Sigma_{\|}$, resp. for main and recovery goal;
- the sub-graph contains states from both copies of the STS;
- the sub-graph stays in the first copy as much as possible.



Theoretical Framework: Conclusions

We have defined a **theoretical framework for web service composition** which:

- allows for an efficient synthesis of the composite service, and
- guarantees the correct execution of the generated composite service, independently from the execution engine.

The framework is based on the following **assumptions**:

- only abstract types and functions are used in the BPEL4WS processes;
- the interactions among processes do not allow for message overpasses;
- the links among processes define a star pattern.

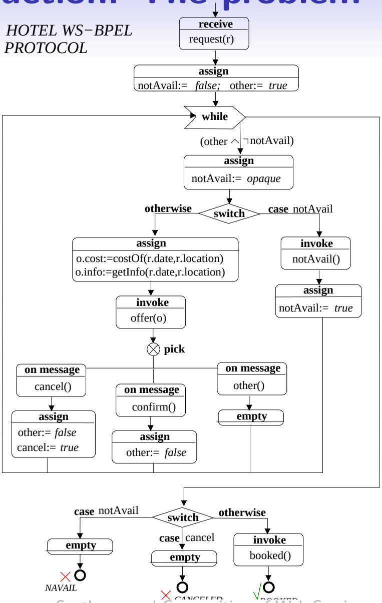
Outline

Knowledge Level Abstraction: The problem

Abstracting away data from the composition domain potentially invalidates composition outcome...

... and modeling abstract data types with small sets of ranges is not the best solution!

Knowledge Level Abstraction: The problem



Abstracting away data from the composition domain potentially invalidates composition outcome...

... and modeling abstract data types with small sets of ranges is not the best solution!

Knowledge Level Abstraction: The problem

Reasoning on the data values exchanged by the web services participating to the composition.

Knowledge Level Abstraction: The problem

Reasoning on the data values exchanged by the web services participating to the composition.

Problems:

- data domains used by the WS are often infinite (e.g. XSD types)
- semantics of data operations is complex (e.g. XPath functions)

Knowledge Level Abstraction: The problem

Reasoning on the data values exchanged by the web services participating to the composition.

Problems:

- data domains used by the WS are often infinite (e.g. XSD types)
- semantics of data operations is complex (e.g. XPath functions)

Existing Approaches

- Abstract away data from composition domain
  - problem: limited applicability
- Explicit model of data values
  - problem: scalability for large sets of data values

Knowledge Level Abstraction: The idea

Reasoning on the data values exchanged by the web services participating to the composition.

Knowledge Level Abstraction: The idea

Reasoning on the data values exchanged by the web services participating to the composition.

Remark: the data flow is relevant for a correct composition, but the actual data values are not important!

Example:

- it is a data-flow constraint on the date that forces the Flight to be invoked before the Hotel however, the actual date is never inspected by the composite service (it is only forwarded to the Hotel)
- when functions are computed in the composite service (e.g., to aggregate cost of hotel and flight), the behavior of the composite service does not depend on the actual definitions of the functions.

Knowledge Level Abstraction: The idea

Reasoning on the data values exchanged by the web services participating to the composition.

Remark: the data flow is relevant for a correct composition, but the actual data values are not important!

Example:

- it is a data-flow constraint on the date that forces the Flight to be invoked before the Hotel however, the actual date is never inspected by the composite service (it is only forwarded to the Hotel)
- when functions are computed in the composite service (e.g., to aggregate cost of hotel and flight), the behavior of the composite service does not depend on the actual definitions of the functions.

Apply knowledge level planning techniques ⇒ Bacchus and Petrick 2002

It is sufficient to encode in each state of the domain a description of which data values (and relations on these values) are known to the composite service.

Knowledge Level Abstraction: Key Challenges

Key challenges

Define a suitable **knowledge base** such that

- knowledge level models can be **extracted automatically** from the WS-BPEL processes description
- a (correct) plan is found for a **relevant set of** realistic **problems**
- **efficient** composition

Knowledge Level Approach

A **knowledge base** is a set of propositions of the following form:

- $K(t = t')$, where t, t' are atomic terms
  - atomic terms are variables or non nested functions: $T \equiv x \mid f(x_1, \ldots, x_n)$

"we know that t and t' have the same value"

Knowledge Level Approach

A **knowledge base** is a set of propositions of the following form:

- $K(t = t')$, where t, t' are atomic terms
  - atomic terms are variables or non nested functions: $T \equiv x \mid f(x_1, \ldots, x_n)$

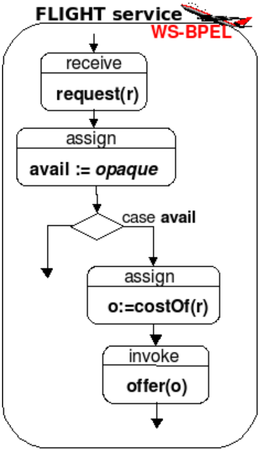
"we know that t and t' have the same value"

Basic operations

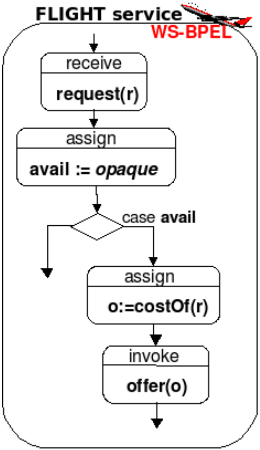
- **Delete**: $del(K, p_1, \ldots, p_n)$ is the knowledge base $K \setminus \{p_1, \ldots p_n\}$
- **Add**: $add(K, p_1, \ldots, p_n)$ is the knowledge base $K \cup \{p_1, \ldots p_n\}$
- **Closure**: $close(K)$ is the knowledge base containing all the propositions that can be deduced (inference rules) from the propositions in K.

Knowledge Base Evolution: Intuitive Example

$K_0 = \emptyset$



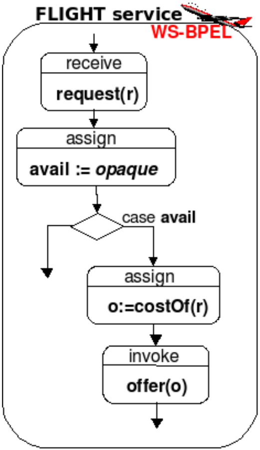
Knowledge Base Evolution: Intuitive Example



$K_0 = \emptyset$

1. *(VTA)* **invoke request(Creq)**
   (Flight) **receive request(r)**
   $K_1 = \{Creq = r\}$

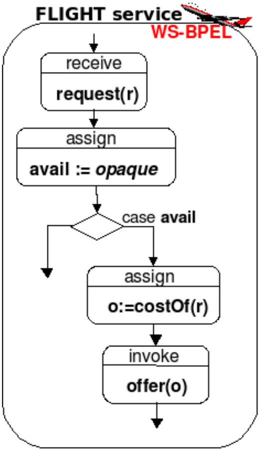
Knowledge Base Evolution: Intuitive Example



$K_0 = \emptyset$

1. *(VTA)* **invoke request(Creq)**
   (Flight) **receive request(r)**
   $K_1 = \{Creq = r\}$
2. *(Flight)* *assign* **avail := opaque**
   $K_2 = \{Creq = r\}$

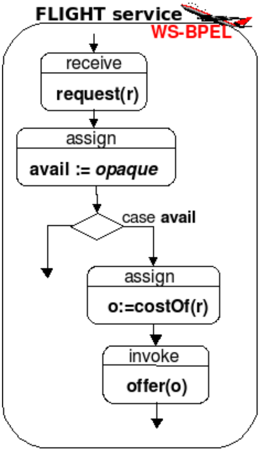
Knowledge Base Evolution: Intuitive Example



$K_0 = \emptyset$

1. *(VTA)* **invoke request(Creq)**
   (Flight) **receive request(r)**
   $K_1 = \{Creq = r\}$
2. *(Flight)* *assign* **avail := opaque**
   $K_2 = \{Creq = r\}$
3. *(Flight)* **case avail**
   $K_3 = \{Creq = r, avail = TRUE\}$

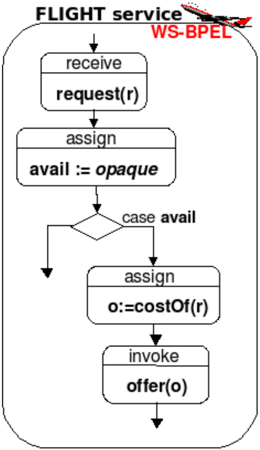
Knowledge Base Evolution: Intuitive Example



$K_0 = \emptyset$

1. *(VTA)* **invoke request(Creq)**
   (Flight) **receive request(r)**
   $K_1 = \{Creq = r\}$
2. *(Flight)* *assign* **avail := opaque**
   $K_2 = \{Creq = r\}$
3. *(Flight)* **case avail**
   $K_3 = \{Creq = r, avail = TRUE\}$
4. *(Flight)* **assign o:=costOf(r)**
   $K_4 = \{Creq = r, avail = TRUE, o = costOf(r),$
   $o = costOf(Creq)\}$

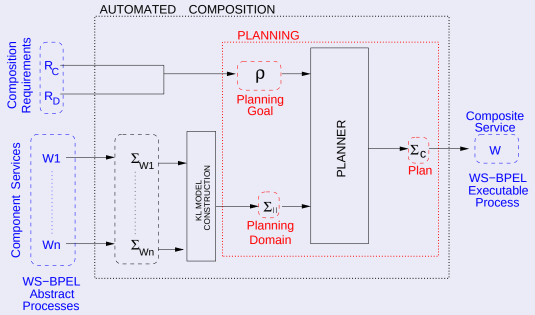
Knowledge Base Evolution: Intuitive Example



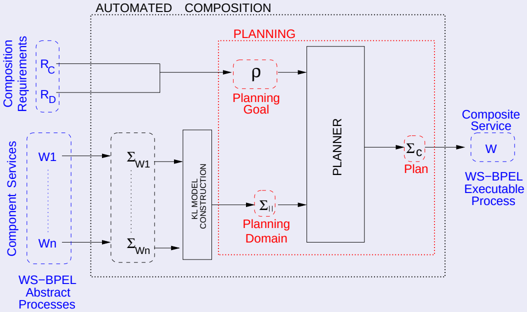
$K_0 = \emptyset$

1. *(VTA)* **invoke request(Creq)**
   (Flight) **receive request(r)**
   $K_1 = \{Creq = r\}$
2. *(Flight)* *assign* **avail := opaque**
   $K_2 = \{Creq = r\}$
3. *(Flight)* **case avail**
   $K_3 = \{Creq = r, avail = TRUE\}$
4. *(Flight)* **assign o:=costOf(r)**
   $K_4 = \{Creq = r, avail = TRUE, o = costOf(r),$
   $o = costOf(Creq)\}$
5. *(Flight)* **invoke offer(o)**
   (VTA) **receive offer(Fcost)**
   $K_5 = \{Creq = r, avail = TRUE, o = costOf(r),$
   $o = costOf(Creq), o = Fcost, Fcost = costOf(Creq)\}$

Knowledge Level Composition Approach



Knowledge Level Composition Approach



Theorem: Correctness of the Knowledge Level approach

Each execution of the new composite service, when orchestrating the component services, satisfies the composition requirements.

Knowledge Level Approach: Results and Considerations

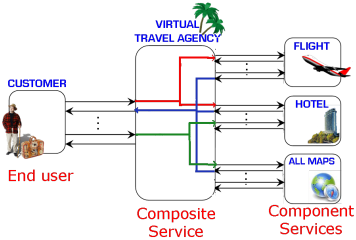
Remarks:

- This very simple knowledge model seems sufficient for most of the realistic cases we considered.
- Operation *close* is heavy (fixed point over more than 200 axioms for the VTA example) and must be executed for each transition
- In the abstract model only a subset of all the possible propositions is considered.
  - the more the set of considered propositions increases, the more the abstraction gets close to the real domain..
  - and the more the size of the planning domain increases.

Outline

Data Net Language: The Idea

Idea: to explicitely constrain the flow of data among the Web Services participating in the composition.



Data Net Language: The Idea

Idea: to explicitely constrain the flow of data among the Web Services participating in the composition.

Define the valid routings and manipulations of messages that the new composite service can perform

- How incoming messages must be used, forwarded or manipulated, to obtain outgoing messages

Data Net Language: The Idea

Idea: to explicitely constrain the flow of data among the Web Services participating in the composition.

Define the valid routings and manipulations of messages that the new composite service can perform

- How incoming messages must be used, forwarded or manipulated, to obtain outgoing messages

- There is no need to reason on actual values
  $\Rightarrow$ abstract data, uninterpreted functions
- Specifying the order in which messages must be sent is not an issue of this language

Data Net Language

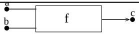
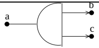
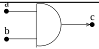
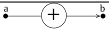
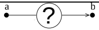
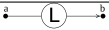
The data flow requirements are collected in a graph called **data-net**

- the **nodes** model IO ports (message parts) of the existing services
- the **arcs** define basic manipulations performed by the composed service
- the **paths** in the graph define the possible routes of the messages

Data Net Language

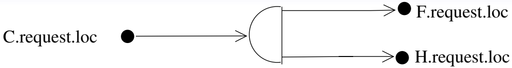
The data flow requirements are collected in a graph called **data-net**

- the **nodes** model IO ports (message parts) of the existing services
- the **arcs** define basic manipulations performed by the composed service
- the **paths** in the graph define the possible routes of the messages

	**forwarder**: simply forwards data received on the input node to the output node
	**function**: upon receiving data on all input nodes, applies the function result and emits the result
	**fork**: forwards data received on the input node to all the output nodes
	**merge**: forwards data received on some input node to the output node, preserving temporal order
	**cloner**: forwards, one or more times, data received from the input node to the output node
	**filter**: receives data on the input node and either forwards it to the output node or discards it
	**last**: forwards to the output node the last data received on the input node and discards all previous

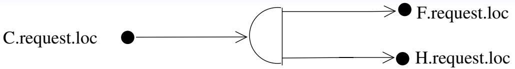
The Virtual Travel Agency Case Study

C.request.loc must be **forwarded both** to **F.request.loc** and **H.request.loc**

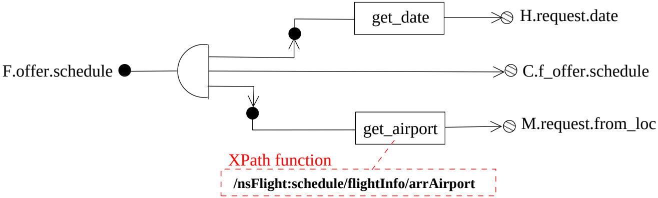


The Virtual Travel Agency Case Study

C.request.loc must be **forwarded both** to **F.request.loc** and **H.request.loc**

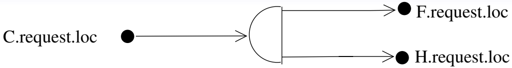


F.offer.schedule must be **manipulated** and **forwarded** to different services

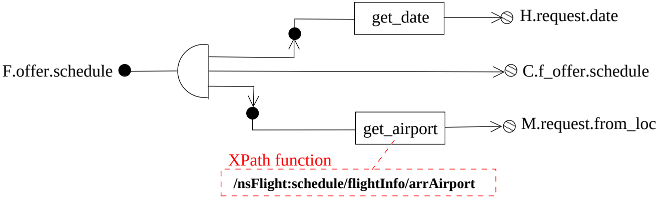


The Virtual Travel Agency Case Study

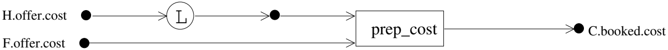
C.request.loc must be **forwarded both** to **F.request.loc** and **H.request.loc**



F.offer.schedule must be **manipulated** and **forwarded** to different services

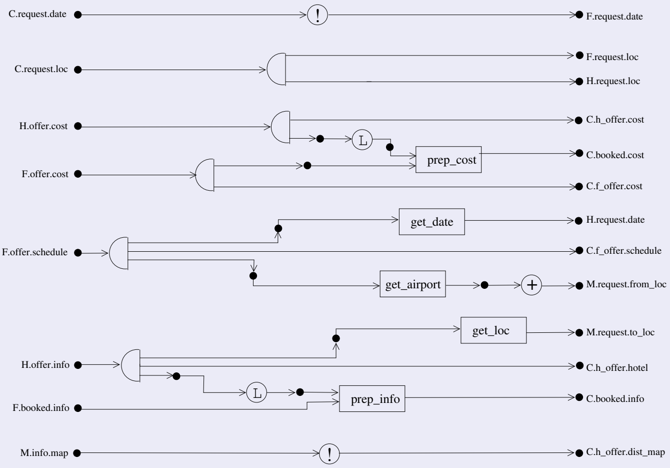


C.booked.cost must be obtained from **F.offer.cost** and form the **last H.offer.cost** received from the Hotel service (the one chosen by the Customer)



The Virtual Travel Agency Case Study

VTA Case Study: the data-net



Integration within the ASTRO Approach

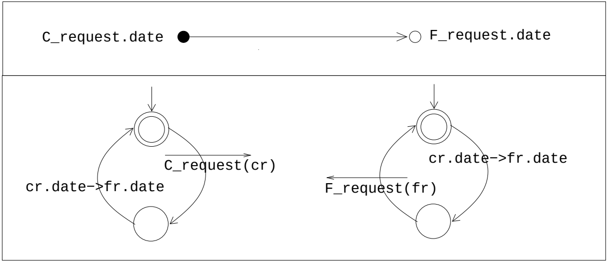
Data Net composition requirements can be encoded within the planning domain in an efficient compositional way.

Integration within the ASTRO Approach

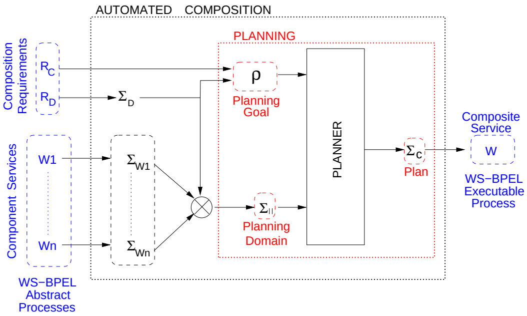
Data Net composition requirements can be encoded within the planning domain in an efficient compositional way.

A data-net defines constraints on the possible operations that the composite process can perform on messages.

- We encode each data-flow element in the data-net as a STS.
- The STS modeling the composition domain, is the synchronized product of all the STSs corresponding to data-flow elements and to component services.



Data Net Composition Approach



Data Requirements as STS

A data-net defines constraints on the possible operations that the composite process can perform on messages.

Data Requirements as STS

A data-net defines constraints on the possible operations that the composite process can perform on messages.

We assume that, in the new composite process, there exists a variable for each connection node in the data-net:

Data Requirements as STS

A data-net defines constraints on the possible operations that the composite process can perform on messages.

We assume that, in the new composite process, there exists a variable for each connection node in the data-net:

- variables associated to external connection nodes are those used by the new composite process to store received messages and to prepare the messages to be sent

Data Requirements as STS

A data-net defines constraints on the possible operations that the composite process can perform on messages.

We assume that, in the new composite process, there exists a variable for each connection node in the data-net:

- variables associated to external connection nodes are those used by the new composite process to store received messages and to prepare the messages to be sent
- variables associated to internal connection nodes are those used to manipulate messages by means of internal functions and assignments

Data Requirements as STS

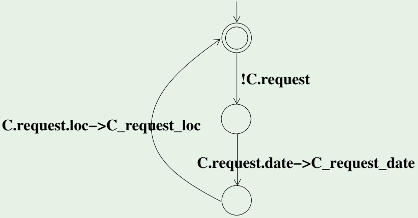
For each output operation of a component service in the data-net we define a STS which represents the sending of the message (as an output action) and the storing of all message parts (as internal actions)

Data Requirements as STS

For each output operation of a component service in the data-net we define a STS which represents the sending of the message (as an output action) and the storing of all message parts (as internal actions)

Example

For the output operation C.request with message parts date and loc we define the following STS:



Data Requirements as STS

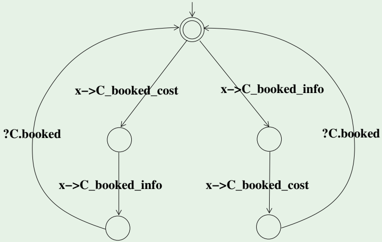
For each input operation of a component service in the data-net we define a STS which represents the storing of all message parts (as internal actions) and the reception of the message (as an input action).

Data Requirements as STS

For each input operation of a component service in the data-net we define a STS which represents the storing of all message parts (as internal actions) and the reception of the message (as an input action).

Example

For the input operation C.booked with message parts info and cost we define the following STS:



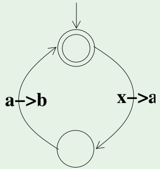
Data Requirements as STS

We define a STS for each data-flow element in the data-net:

Data Requirements as STS

We define a STS for each data-flow element in the data-net:

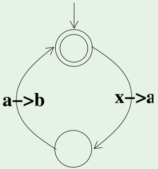
`id(a)(b)`



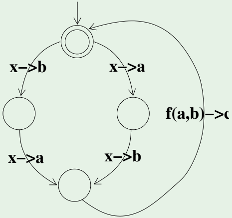
Data Requirements as STS

We define a STS for each data-flow element in the data-net:

`id(a)(b)`



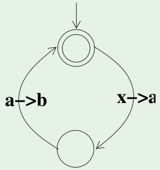
`oper[f](a,b)(c)`



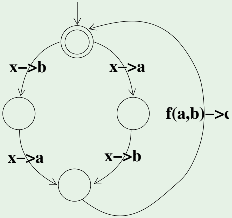
Data Requirements as STS

We define a STS for each data-flow element in the data-net:

id(a)(b)

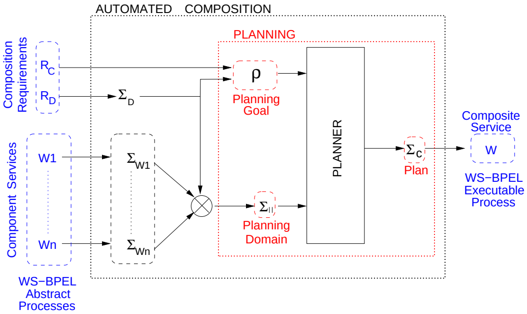


oper[f](a,b)(c)

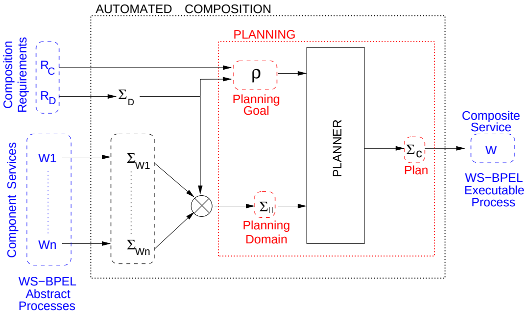


The STS $\Sigma_{\mathcal{D}}$, modeling the data-net, is the synchronized product of all the STSs corresponding to external connection nodes and data-flow elements.

Data Net Composition Approach



Data Net Composition Approach



Theorem: Correctness of the Data Net approach

Each execution of the new composite service W, when interacting with the components, satisfies the data flow requirements in the data net R_D.

Outline

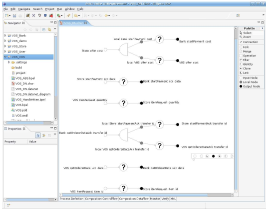
Implementation: the ASTRO WS-Synth Tool

The presented WS composition framework has been implemented as an Eclipse Plugin within the **ASTRO toolset** and is distributed under LGPL license.

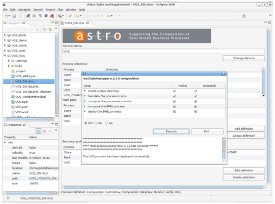
astro — Supporting *the* Composition *of* Distributed Business Processes

www.astroproject.org

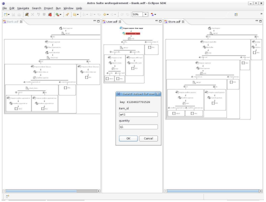
REQUIREMENTS SPECIFICATION



AUTOMATED SYNTHESIS



DEPLOYMENT and RUN



Outline

The Amazon-MPS Case Study

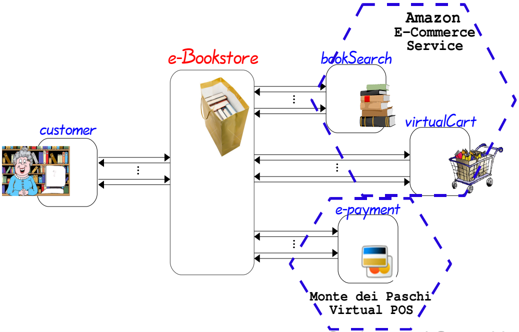
- Work in collaboration with Monte Paschi di Siena (MPS) - one of the most important banks in Italy

The Amazon-MPS Case Study

- Work in collaboration with Monte Paschi di Siena (MPS) - one of the most important banks in Italy
- Great opportunity to evaluate the feasibility and efficiency of the ASTRO approach on a **real composition scenario** that entails a high level of complexity.

The Amazon-MPS Case Study

- Work in collaboration with Monte Paschi di Siena (MPS) - one of the most important banks in Italy
- Great opportunity to evaluate the feasibility and efficiency of the ASTRO approach on a **real composition scenario** that entails a high level of complexity.



Amazon E-Commerce Service (ECS)

ECS aim

Exposes Amazon product information and e-commerce functionalities:

- searching for Amazon products (books, movies, music, restaurant, etc.)
- handling shopping carts
- inspecting customer contents (reviews, wish lists, listmania lists, etc..)
- inspecting vendor contents (customer feedbacks, etc..)

Amazon E-Commerce Service (ECS)

ECS aim

Exposes Amazon product information and e-commerce functionalities:

- searching for Amazon products (books, movies, music, restaurant, etc.)
- handling shopping carts
- inspecting customer contents (reviews, wish lists, listmania lists, etc..)
- inspecting vendor contents (customer feedbacks, etc..)

ECS specification

- WSDL document defining available operations, messages and their data structure
- several documents describing informally (natural language, flow charts, etc.):
  $\Rightarrow$ business workflows
  $\Rightarrow$ failures and non-nominal cases
  $\Rightarrow$ structure of each specific purpose message (movie-Search vs book-Search)

Amazon E-Commerce Service (ECS)

ECS aim

Exposes Amazon product information and e-commerce functionalities:

- searching for Amazon products (books, movies, music, restaurant, etc.)
- handling shopping carts
- inspecting customer contents (reviews, wish lists, listmania lists, etc..)
- inspecting vendor contents (customer feedbacks, etc..)

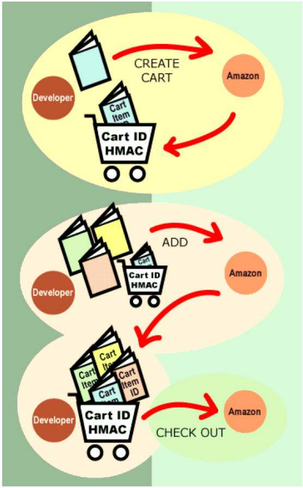
ECS specification

- WSDL document defining available operations, messages and their data structure
- several documents describing informally (natural language, flow charts, etc.):
  ⇒ business workflows
  ⇒ failures and non-nominal cases
  ⇒ structure of each specific purpose message (movie-Search vs book-Search)

⇒ **Need for an explicit and formal specification of each business workflow**

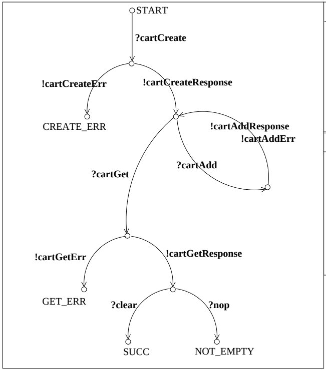
(Amazon Book-Search and Amazon Virtual-Cart)

Amazon Virtual-Cart Service: Flow "Specification"



Amazon Virtual-Cart Service

Amazon Virtual Cart

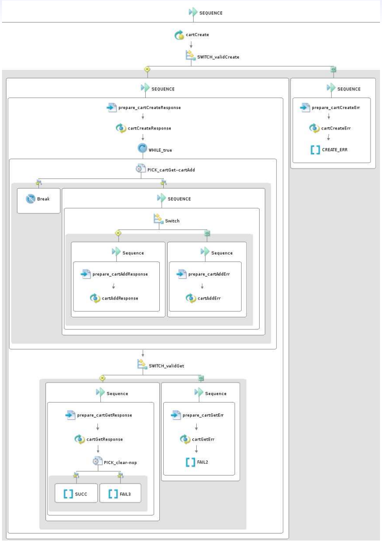


INPUT

ACTION	MSG PART
cartCreate	body
cartAdd	body

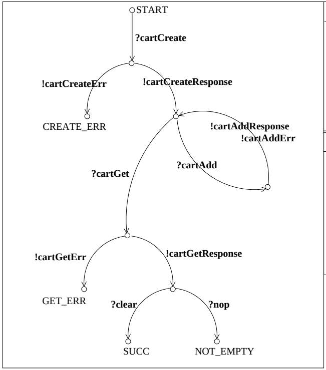
OUTPUT

ACTION	MSG PART
cartCreateResponse	body
cartCreateErr	error
cartAddResponse	body
cartAddErr	error
cartGetResponse	body
cartGetErr	error



Amazon Virtual-Cart Service

Amazon Virtual Cart

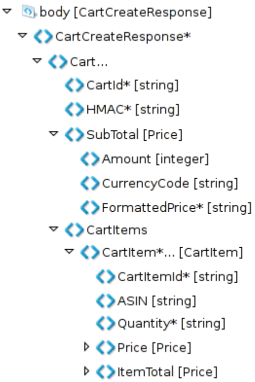


INPUT

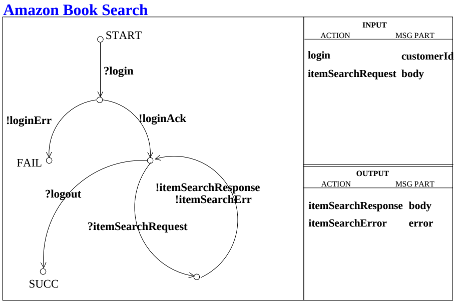
ACTION	MSG PART
cartCreate	body
cartAdd	body

OUTPUT

ACTION	MSG PART
cartCreateResponse	body
cartCreateErr	error
cartAddResponse	body
cartAddErr	error
cartGetResponse	body
cartGetErr	error



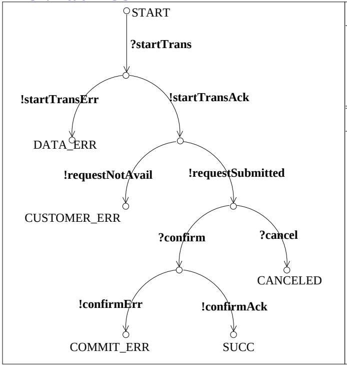
Amazon Book-Search Service



MPS Virtual Point of Sale (POS) Service

Models a real on-line payment service offered by Monte Paschi di Siena

MPS Virtual POS



INPUT

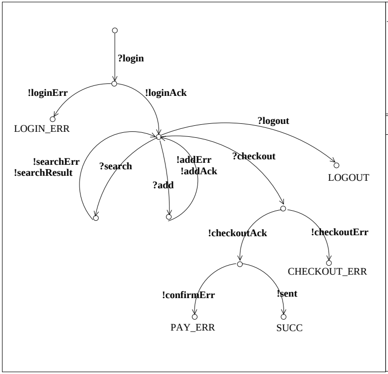
ACTION	MSG PART
startTrans	amount shopID shopEmail

OUTPUT

ACTION	MSG PART
startTransAck	paymentURL
startTransErr	errorDesc
requestSubmitted	transID
requestNotAvail	errorDesc
confirmAck	transDate transTime transAuthorization
confirmErr	errorDesc

e-Bookstore service customer interface

e−Bookstore Client



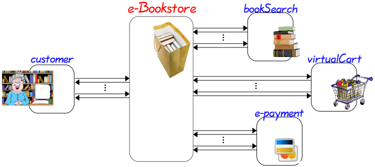
INPUT

ACTION	MSG PART
login	customerId
search	keyword author publisher title
add	ASIN quantity

OUTPUT

ACTION	MSG PART
searchResult	numItems ASIN detailPageURL author title publisher ISBN price
searchErr	error
addErr	error
addAck	subTotal
checkoutErr	error
checkoutAck	subTotal paymentURL
sent	transDate transTime transAuthorization
confirmErr	error

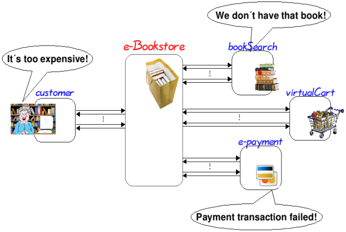
Control Flow Requirements



e-Bookstore goal

SELL BOOKS

Control Flow Requirements



e-Bookstore goal

do whatever is possible to

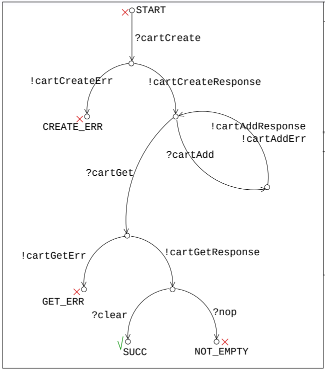
SELL BOOKS

if something goes wrong guarantee

NO SINGLE COMMITMENTS

Control Flow Requirements

Amazon Virtual Cart



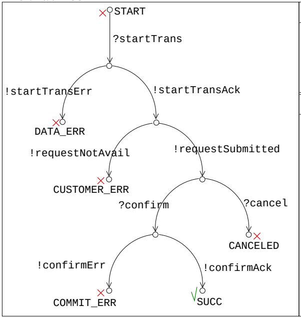
INPUT

ACTION	MSG PART
cartCreate	body
cartAdd	body

OUTPUT

ACTION	MSG PART
cartCreateResponse	body
cartCreateErr	error
cartAddResponse	body
cartAddErr	error
cartGetResponse	body
cartGetErr	error

MPS Virtual POS



INPUT

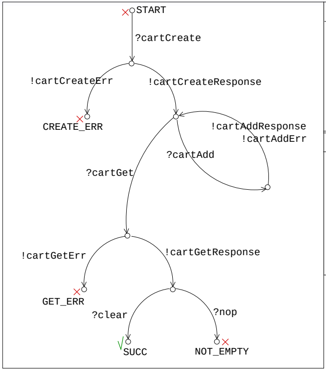
ACTION	MSG PART
startTrans	amount shopID shopEmail

OUTPUT

ACTION	MSG PART
startTransAck	paymentURL
startTransErr	errorDesc
requestSubmitted	transID
requestNotAvail	errorDesc
confirmAck	transDate transTime transAuthorization
confirmErr	errorDesc

Control Flow Requirements

Amazon Virtual Cart



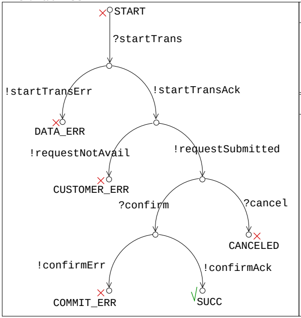
INPUT

ACTION	MSG PART
cartCreate	body
cartAdd	body

OUTPUT

ACTION	MSG PART
cartCreateResponse	body
cartCreateErr	error
cartAddResponse	body
cartAddErr	error
cartGetResponse	body
cartGetErr	error

MPS Virtual POS



INPUT

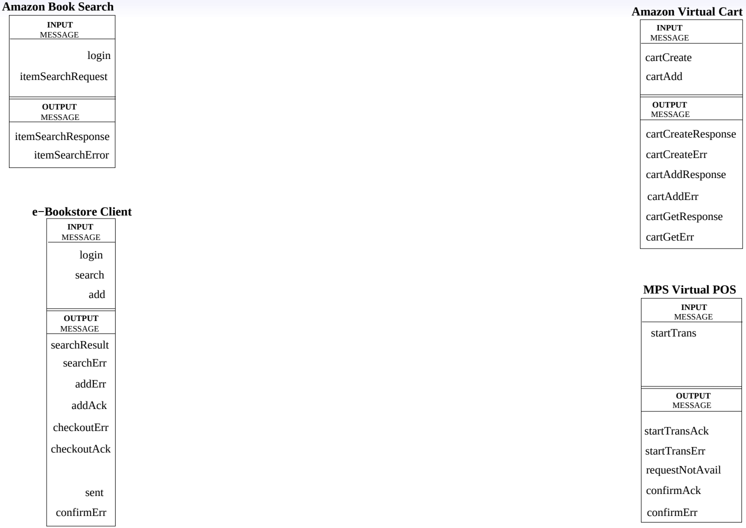
ACTION	MSG PART
startTrans	amount shopID shopEmail

OUTPUT

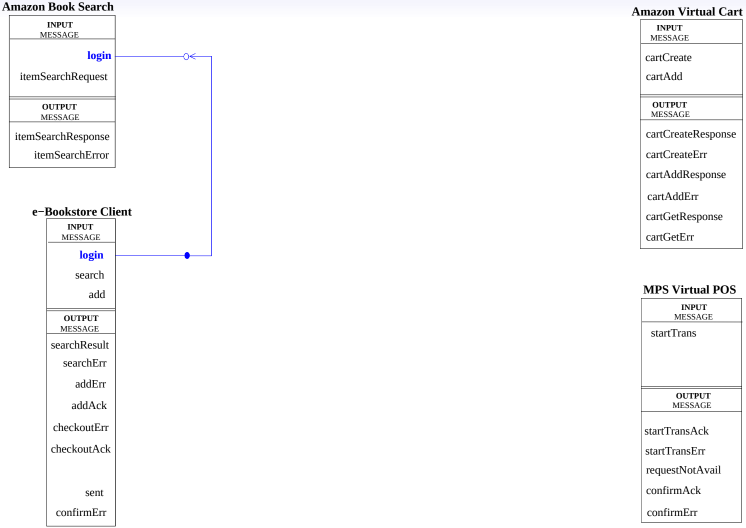
ACTION	MSG PART
startTransAck	paymentURL
startTransErr	errorDesc
requestSubmitted	transID
requestNotAvail	errorDesc
confirmAck	transDate transTime transAuthorization
confirmErr	errorDesc

	eBS	ABS	AVC	VPOS
Primary	✓	✓	✓	✓
Secondary	×	✓/×	×	×

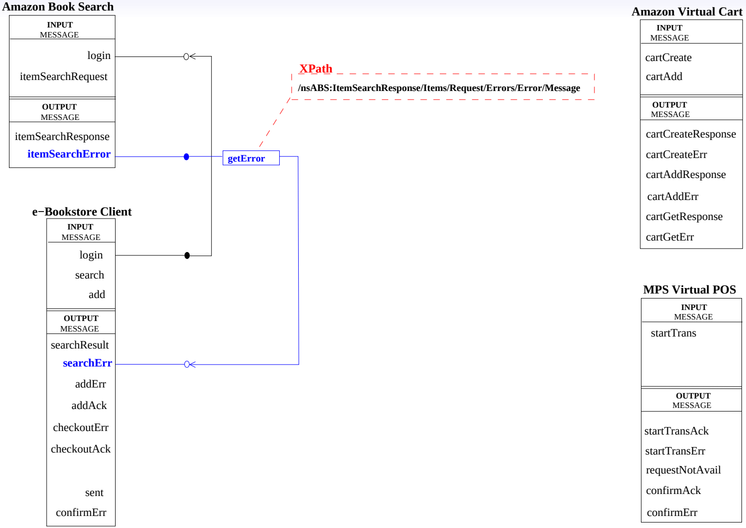
e-Bookstore Data Flow Requirements



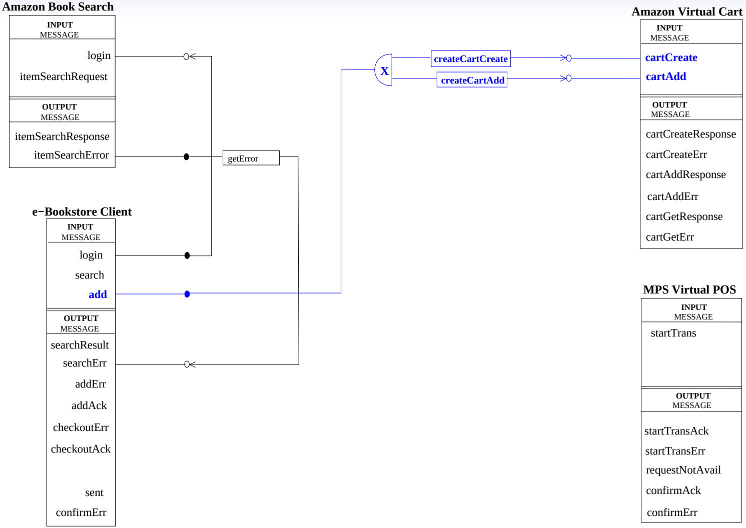
e-Bookstore Data Flow Requirements



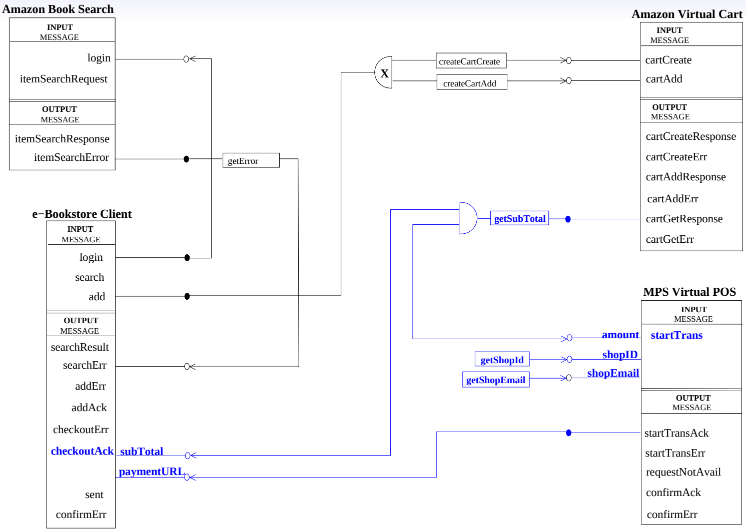
e-Bookstore Data Flow Requirements



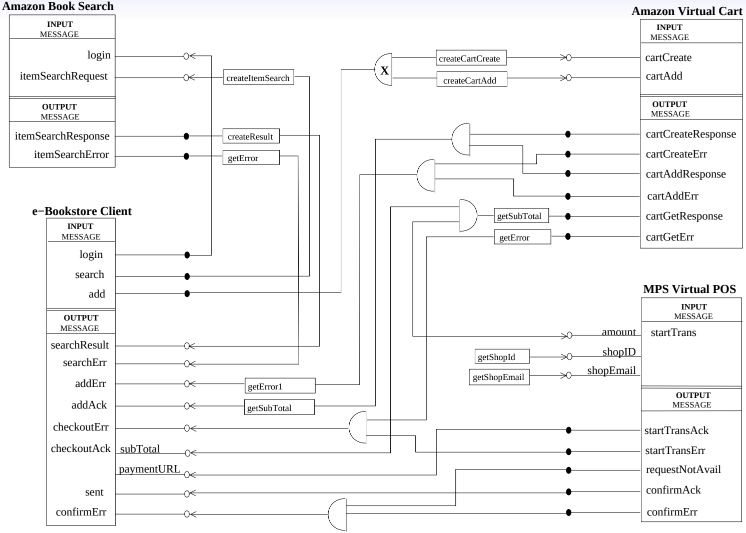
e-Bookstore Data Flow Requirements



e-Bookstore Data Flow Requirements



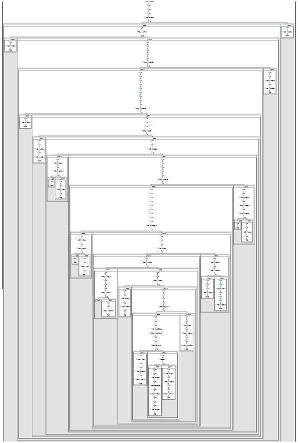
e-Bookstore Data Flow Requirements



Evaluation: The Amazon-MPS Case Study

⇒ **Efficiency of the automated composition techniques**

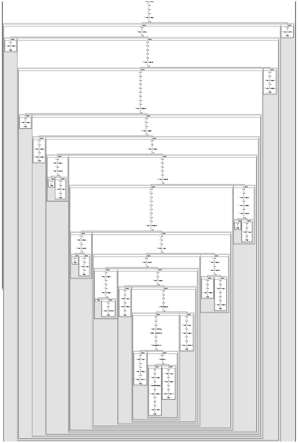
- composition techniques can scale up to real world scenarios
  - composite service: 200 WS-BPEL basic activities
  - requirements specification 2 hours
  - composition time: 200 sec.



Evaluation: The Amazon-MPS Case Study

⇒ **Efficiency of the automated composition techniques**

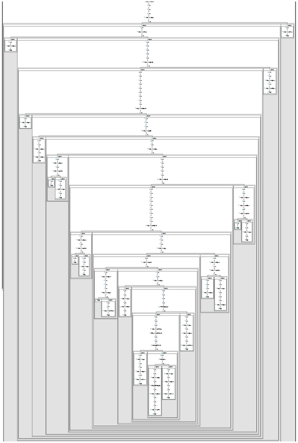
- composition techniques can scale up to real world scenarios
  - composite service: 200 WS-BPEL basic activities
  - requirements specification 2 hours
  - composition time: 200 sec.
- hand writing e-Bookstore code :
  - composite service: 1 missing scenario
  - composition time: more than 20 hours



Evaluation: The Amazon-MPS Case Study

⇒ Efficiency of the automated composition techniques

- composition techniques can scale up to real world scenarios
  - composite service: 200 WS-BPEL basic activities
  - requirements specification 2 hours
  - composition time: 200 sec.
- hand writing e-Bookstore code :
  - composite service: 1 missing scenario
  - composition time: more than 20 hours
- the synthesised code is readable and implements the same strategy



Outline

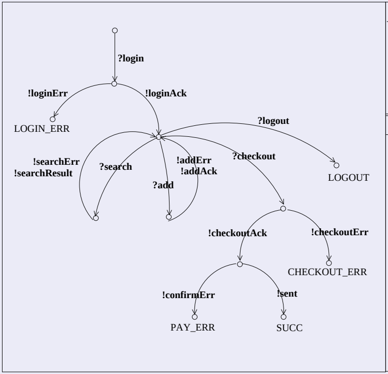
Amazon Case Study: Problem

⇒ **Do we really need to specify the customer interaction protocol?**

Amazon Case Study: Problem

⇒ Do we really need to specify the customer interaction protocol?

e−Bookstore Client



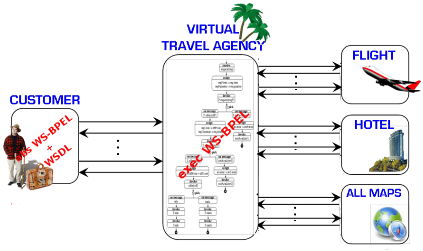
INPUT ACTION	MSG PART
login	customerId
search	keyword author publisher title
add	ASIN quantity

OUTPUT ACTION	MSG PART
searchResult	numItems ASIN detailPageURL author title publisher ISBN price
searchErr	error
addErr	error
addAck	subTotal
checkoutErr	error
checkoutAck	subTotal paymentURL
sent	transDate transTime transAuthorization
confirmErr	error

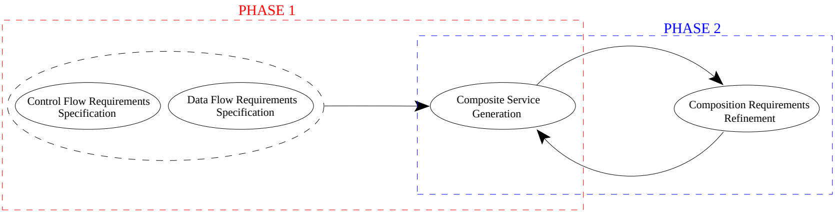
The Problem

⇒ **We do not want to specify the customer interaction protocol!**

- Automatically obtain both the customer interaction protocol and the composite process
- Define a semi-automated iterative development process that reduces as much as possible the effort for the composition task.



The Proposed Iterative Approach



- **Phase 1.** obtain a **preliminary version** of the composite process starting from initial composition requirements.
- **Phase 2.** on the basis of the automated composition outcomes **refine** both the composition requirements and the customer interface and automatically **re-compose**.

Phase 1: Control Flow requirements specification

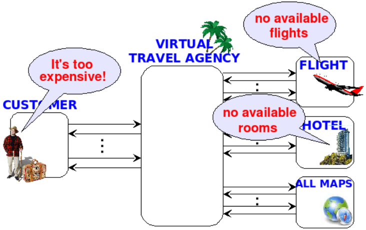
⇒ **Specification of control flow requirements (manual).**

⇒ **Translation in the internal formal language (automated).**

Phase 1: Control Flow requirements specification

⇒ **Specification of control flow requirements (manual).**

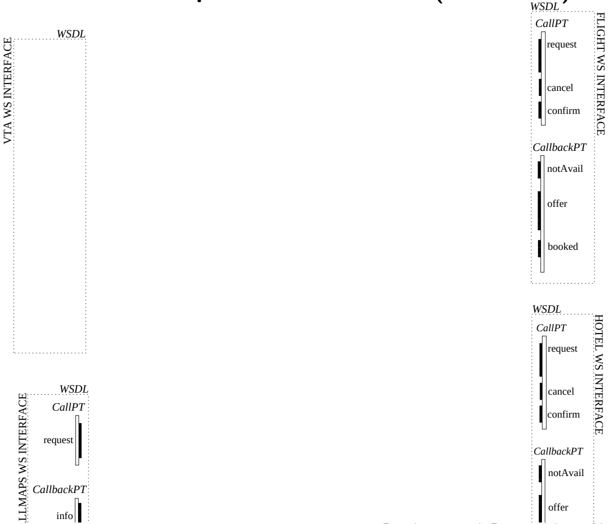
⇒ **Translation in the internal formal language (automated).**



	Flight	Hotel	VTA
Primary	✓	✓	✓
Secondary	×	×	×

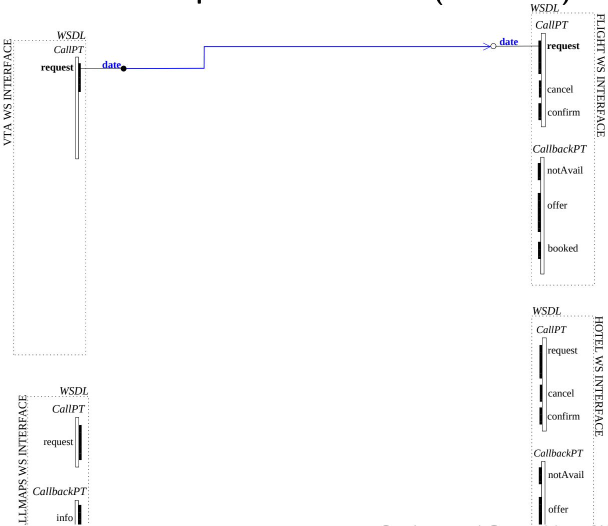
Phase 1: Data Flow requirements specification

⇒ **Specification of data flow requirements (manual).**
⇒ **Specification of the composite service WSDL (automated).**



Phase 1: Data Flow requirements specification

⇒ **Specification of data flow requirements (manual).**
⇒ **Specification of the composite service WSDL (automated).**



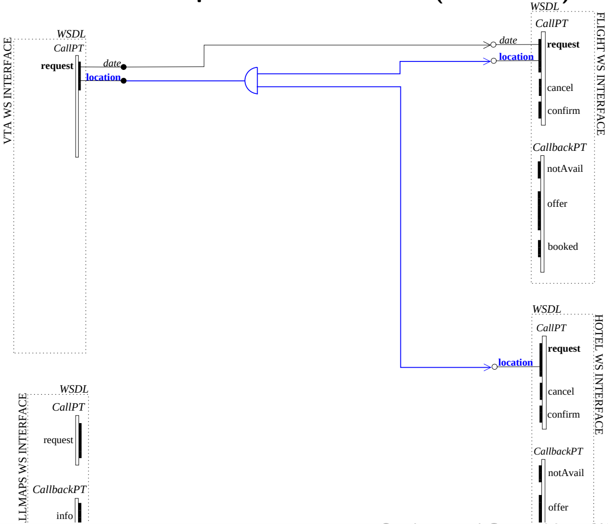
Phase 1: Data Flow requirements specification

VTA WSDL interface

⟨ message name="requestMsg"⟩
  ⟨ part name="date" type="nsFlight:dateType" /⟩
⟨/ message⟩
⟨ portType name="CallPT"⟩
  ⟨ operation name="request"⟩
    ⟨ input message="requestMsg" /⟩
  ⟨/ operation⟩
⟨/ portType⟩

Phase 1: Data Flow requirements specification

⇒ **Specification of data flow requirements (manual).**
⇒ **Specification of the composite service WSDL (automated).**



Phase 1: Data Flow requirements specification

VTA WSDL interface

⟨ message name="requestMsg" ⟩
  ⟨ part name="date" type="nsFlight:dateType" /⟩
  ⟨ part name="location" type="nsFlight:dateType" /⟩
⟨/ message⟩
⟨ portType name="CallPT" ⟩
  ⟨ operation name="request" ⟩
    ⟨ input message="requestMsg" /⟩
  ⟨/ operation⟩
⟨/ portType⟩

Phase 1: Composite service generation

⇒ **Generation of composite service exec WS-BPEL (automated)**

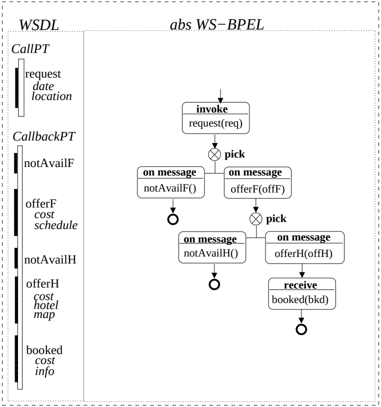
⇒ **Generation of client WS-BPEL (automated)**

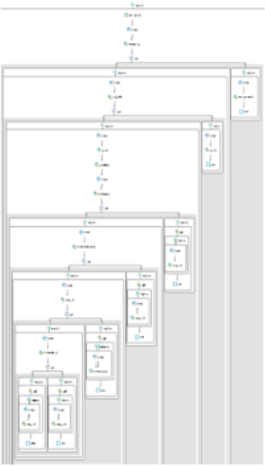
Phase 1: Composite service generation

⇒ **Generation of composite service exec WS-BPEL (automated)**

⇒ **Generation of client WS-BPEL (automated)**

VTA Customer WS protocol

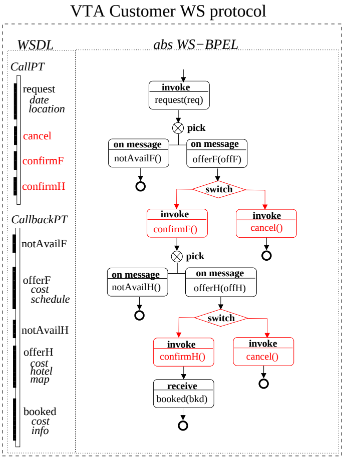




Phase 2: Requirements refinement and re-composition.

⇒ **Refinement of the customer protocol (manual)**

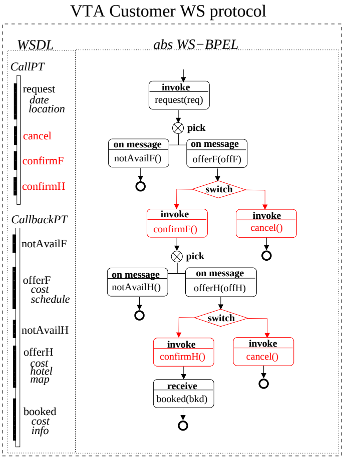
⇒ **Re-composition of the composite process (automatic)**

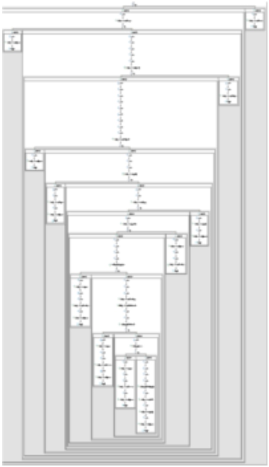


Phase 2: Requirements refinement and re-composition.

⇒ **Refinement of the customer protocol (manual)**

⇒ **Re-composition of the composite process (automatic)**





Iterative Approach: Concluding Remarks

- Very good in terms of specification effort
  - From 2 hours to 1 1/2 hours
  - Removed the less conceptual part of the specification
- Incremental approach
  - Helps solving the "Synthesis not found" problem
  - Identifies the requirement that makes the synthesis impossible

Outline

The ASTRO Automated Composition Approach

Current Composition Flavour

- **Centralized**: syntesize a ready to run new executable process.
- **Process-level**: components are complex and stateful workflows.
- **Design-time**: We have already selected the set of services we want to compose (no discovery, no selection)
- **Requirements**: both control and data flow requirements.

The ASTRO Automated Composition Approach

Current Composition Flavour

- **Centralized**: syntesize a ready to run new executable process.
- **Process-level**: components are complex and stateful workflows.
- **Design-time**: We have already selected the set of services we want to compose (no discovery, no selection)
- **Requirements**: both control and data flow requirements.
- **Aspect-oriented**:
    - decouple the different aspects of the component services;
    - express composition requirements separately for these aspects.

The ASTRO Automated Composition Approach

Current Composition Flavour

- **Centralized**: syntesize a ready to run new executable process.
- **Process-level**: components are complex and stateful workflows.
- **Design-time**: We have already selected the set of services we want to compose (no discovery, no selection)
- **Requirements**: both control and data flow requirements.
- **Aspect-oriented**:
  - decouple the different aspects of the component services;
  - express composition requirements separately for these aspects.
- **Iterative approach**:
  - composition as an iterative semi-automatic process.

ASTRO: The Team

ASTRO exists thanks to:

- Annapaola Marconi
- Dmitry Shaparau
- Fabio Barbon
- Gabriele Zacco
- Gigi Lucchese
- Heorhi Raik
- Marco Pistore
- Michele Trainotti
- Paolo Traverso
- Pietro Pilolli
- Raman Kazhamiakin
- Piergiorgio Bertoli
- ...

ASTRO: The Team

ASTRO exists thanks to:

- Annapaola Marconi
- Dmitry Shaparau
- Fabio Barbon
- Gabriele Zacco
- Gigi Lucchese
- Heorhi Raik
- Marco Pistore
- Michele Trainotti
- Paolo Traverso
- Pietro Pilolli
- Raman Kazhamiakin
- Piergiorgio Bertoli
- ...
- **Plus all our partners and collaborators...**

ASTRO: Some Publications

- **Control Flow Requirements for Automated Service Composition.** H. Raik, R. Kazhamiakin, M. Pistore, P. Bertoli, M. Paolucci and M. Wagner. (IEEE ICWS 09)
- **An Iterative Approach for the Process-level Composition of Web Services.** A. Marconi, M. Pistore and P. Traverso. (SEEFM 07)
- **Automated Web Service Composition at Work: the Amazon/MPS Case Study.** A. Marconi, M. Pistore, P.Poccianti and P. Traverso. (IEEE ICWS 07)
- **Implicit vs. Explicit Data-Flow Requirements in Web Service Composition Goals.** A. Marconi, M. Pistore and P. Traverso. (ICSOC 06)
- **Specifying Data-Flow Requirements for the Automated Composition of Web Services.** A. Marconi, M. Pistore and P. Traverso. (IEEE SEFM 06)
- **A Minimalist Approach to Semantic Annotations for Web Processes Compositions.** M. Pistore, L. Spalazzi and P. Traverso. (ESWC 06)
- **Automated Composition of Web Services by Planning at the Knowledge Level.** M. Pistore, A. Marconi, P. Traverso and P. Bertoli. (IJCAI 05)
- **Automated Synthesis of Composite BPEL4WS Web Services.** M. Pistore, P. Traverso, P. Bertoli and A.Marconi. (IEEE ICWS 05)
- **Automated Composition of Semantic Web Services into Executable Processes.** P. Traverso and M. Pistore. (ISWC 04)

ASTRO: Exploitation

The presented approach is

- adopted by **SAP AG** as starting point for an internal project proposal on business process integration / web service composition.
- currently being adopted in a **technology transfer project** (**VERSO21** company of OPERA21 group) that aims at exploiting Web service composition techniques in an **industrial setting**, in order to improve the effectiveness of customization of enterprise applications.
- tested, in collaboration with **Monte dei Paschi di Siena** (MPS), on an **on-line shopping service** integrating Amazon E-Commerce Services and MPS e-payment services.
- adopted as reference to define the **composition patterns** for the EC service delivery platform **NEXOF**.

On Adopting Formal Methods for Web Service Composition

Lessons learnt:

- **Look to the real world**:
  - choose a real language (BPEL)
  - look for real case studies (Amazon)

On Adopting Formal Methods for Web Service Composition

Lessons learnt:

- **Look to the real world**:
  - choose a real language (BPEL)
  - look for real case studies (Amazon)
- **Choose the right hammer (formal method) for the nail at hand (problem)**. In ASTRO we used all the following hammers:
  - automated task planning
  - automata synthesis
  - process algrebras
  - temporal logics
  - belief logics
  - ...

On Adopting Formal Methods for Web Service Composition

Lessons learnt:

- **Look to the real world**:
  - choose a real language (BPEL)
  - look for real case studies (Amazon)
- **Choose the right hammer (formal method) for the nail at hand (problem)**. In ASTRO we used all the following hammers:
  - automated task planning
  - automata synthesis
  - process algrebras
  - temporal logics
  - belief logics
  - ...
- **Integrate the promising results into a demo platform**:
  - very expensive in terms of resources, but necessary for exploitation

Synthesys and Composition of Web Services.

Marco Pistore
FBK-irst, Trento, Italy

June 3, 2009 - SFM-09:WS - Bertinoro