

# Automated Service Composition with Adaptive Planning

Sandrine Beauche<sup>1</sup>, **Pascal Poizat**<sup>2</sup>

<sup>1</sup>INRIA / ARLES Project-Team, France

<sup>2</sup>Univ. Évry, France and LRI, France  
pascal.poizat@lri.fr

<http://www.lri.fr/~poizat>

this work has been done while at IBISC and INRIA / ARLES Project-Team, France

ICSOC'2008

December 1–5, 2008

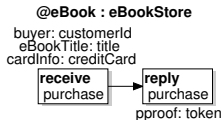
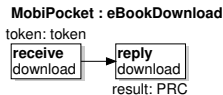
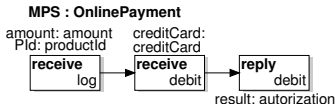
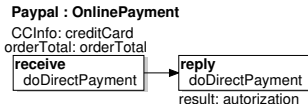
# Motivation

- user needs realized through the **automatic assembly of available resources**
- SOC is a **cornerstone** towards the realization of this vision (potentially heterogeneous) resources abstracted as services
- yet:
  - services developed by different third-parties incompatible protocols and data flows  
→ **horizontal mismatch**
  - higher description level for user requirements than services prevents composition  
→ **vertical mismatch**

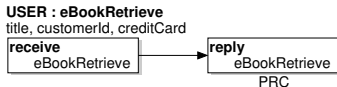
**composition includes** (some form of) **adaptation**

# Example (Services)

available services:



? – what if the user wants:



example inspired from the MPS case-study,  
[Marconi *et al.*, ICWS'07]

# Approach

- planning increasingly applied in SOC [Peer, Tech. Report, 2005]  
supports underspecified requirements and multiple compositions  
→ graphplan forward planning
- hierarchical planning [Lotem *et al.*, AAAI/IAAI'99]  
supports decomposition of abstract requirements into concrete tasks  
→ Capacity Semantic Structure (CSS)
- software adaptation [Canal *et al.*, IEEE TSE, 34(4), 2008]  
support for solving mismatch out using adaptation contracts or semantics  
→ Data Semantic Structure (DSS)

Service = a **capacity** and a conversation in YAWL with I/O

User Requirement = a service **without conversation**

the **objective** is to obtain this conversation

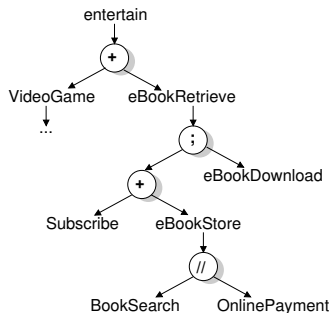
# Capacity Semantic Structure

## Capacity Semantic Structure

A Capacity Semantic Structure (CSS) is a couple  $(\mathcal{K}, \mathcal{T}_{\mathcal{K}})$  where:

- $\mathcal{K}$  is a set of concepts (capacities)
- $\mathcal{T}_{\mathcal{K}}$  is a tree where nodes can be either:
  - a capability node (in  $\mathcal{K}$ ), or
  - a control node  
sequence: ;  
choice: +  
parallelism: //

+ tree well-formedness



# Data Semantic Structure

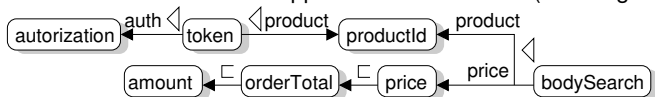
## Data Semantic Structure

A Data Semantic Structure (DSS) a tuple  $(\mathcal{D}, \triangleleft, \sqsubseteq)$   
where:

- $\mathcal{D}$  is a set of concepts (data type/semantics),
- $\triangleleft$  is a composition relation  
( $d_1 \triangleleft_x d_2$  means a  $d_1$  is composed of an  $x$  of type  $d_2$ ), and
- $\sqsubseteq$  is a simulation relation  
( $d_1 \sqsubseteq d_2$  means  $d_1$  can be used as a  $d_2$ ).

no circular composition

transformation function added to support further mismatch (see long version)



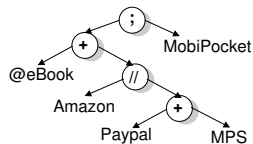
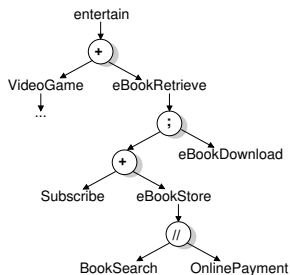
# Discovery

## Objectives:

- pre-selection of services based on their capacity
- tagging the CSS (I-CSS)

## Steps:

- 1 required capacity  
→ root
- 2 abstract capacities  
→ choice
- 3 capacity nodes  
→ services
- 4 tree cleaning



# Vertical Adaptation

Graphplan building:

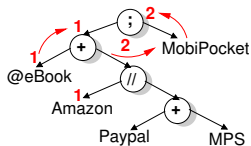
- alternate data and service call layers, arcs are I/O dependencies
- no-op actions transfer data

before service added

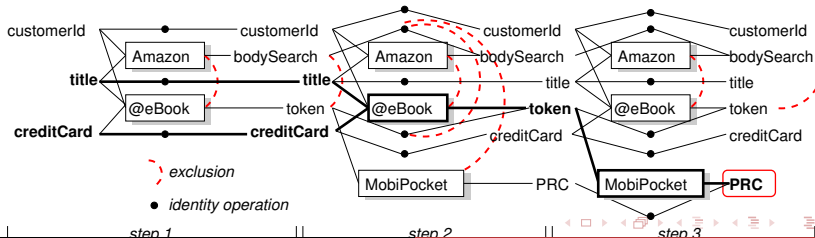
available inputs  
I-CSS respected

after service added

outputs produced  
I-CSS tagged exclusion relations



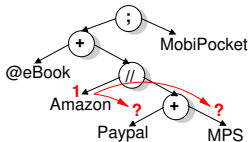
- stop when maximum solution length (I-CSS) reached



# Horizontal Adaptation

In-between data layers:

- check if services should be applicable (but are not)
- try to produce required inputs using data adaptation
- can be seen as planning too:



**decomposition**

$\text{decomp}(d, D)$

if  $D = \{d_i \mid d \triangleleft d_i\}$

**composition**

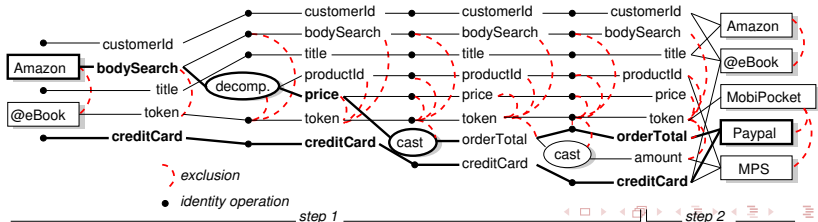
$\text{comp}(D, d)$

if  $D = \{d_i \mid d \triangleleft d_i\}$

**cast**

$\text{cast}(d_1, d_2)$

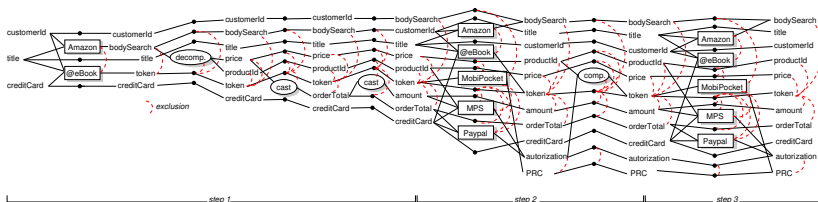
if  $d_1 \sqsubset d_2$



# Getting the Plans

Plans are obtained backtracking in the graphplan

- from the required data output (PRC)
- following the I-CSS

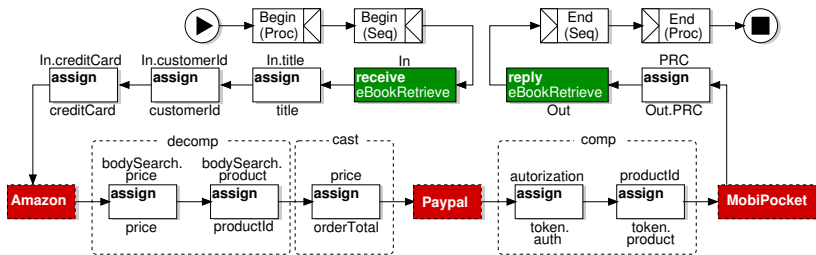


yields

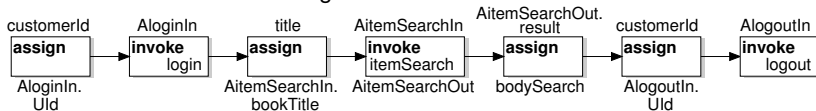
- @eBook;MobiPocket
- Amazon;decomp(bodySearch, {productId,price });cast(price,orderTotal);Paypal;  
comp({ authorization,productId },token);MobiPocket
- Amazon;decomp(bodySearch, { productId,price });cast(price,orderTotal);  
cast(orderTotal,amount);MPS;comp({ authorization,productId },token);MobiPocket.

# Implementing the Plans

Plans are transformed into **YAWL orchestrators**

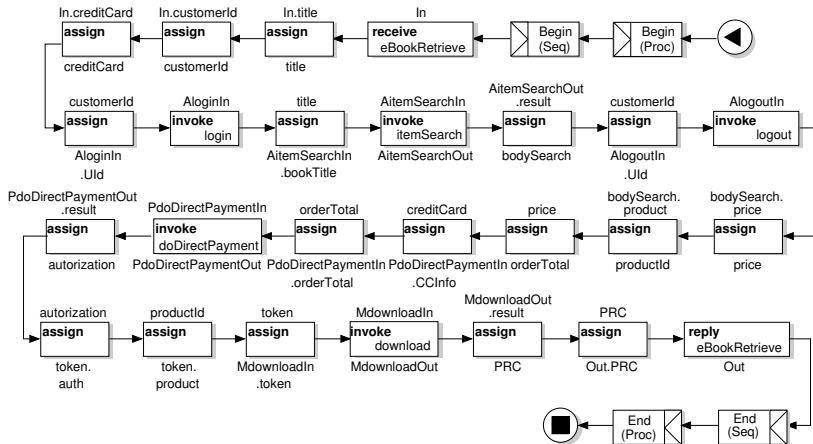


Service conversations are integrated in them



# Implementing the Plans

Plans are transformed into **YAWL orchestrators**



Perspective: YAWL2BPEL for translation into BPEL

[Brogi and Popescu, ICSSOC'06]

# Concluding Remarks

## Results

- support for deployment time automatic service orchestration
- adaptation for vertical and horizontal mismatch
- multiple solutions can be generated
- GraphAdaptor tool

## Perspectives

expressiveness

conversations over capacities following  
[Pistore *et al.*, ICWS'06], [Ben Mokhtar *et al.*, JSS 80(12), 2007]  
multi-sets for service I/O  
richer service and requirement models  
to ensure safety by construction

run-time adaptation

application to run-time service replacement

# Concluding Remarks

## Results

- support for deployment time automatic service orchestration
- adaptation for vertical and horizontal mismatch
- multiple solutions can be generated
- GraphAdaptor tool

## Perspectives

### expressiveness

conversations over capacities following  
[Pistore *et al.*, ICWS'06], [Ben Mokhtar *et al.*, JSS 80(12), 2007]  
multi-sets for service I/O  
richer service and requirement models  
to ensure safety by construction

### run-time adaptation

application to run-time service replacement

thank you for your attention