



# Relating Model-Based Adaptation and Implementation Platforms: A Case Study with WF/.NET 3.0

Javier Cubo, G.Salaün, C.Canal, E.Pimentel




P.Poizat



WCOP 2007  
Berlin, Germany  
July 31, 2007

- Motivations
- Contributions
- WF Overview
- WF & BPEL
- Case Study: On-Line Computer Sale
- Composition and Adaptation of WF Components
- Conclusion

- ❑ Resolve **mismatch problems** among components 
- ❑ Need to **automate the component adaptation**
- ❑ Many approaches dedicated to model-based adaptation focus on
  - ❑ the **behavioural** interoperability level
  - ❑ generating new components called **adaptors**
- ❑ Very few approaches relate their results with existing programming languages and platforms
  - ❑ **COM/DCOM** [Inverardi-JSS03] and **BPEL** [Brogi-ICSOC06]

## □ Proposal

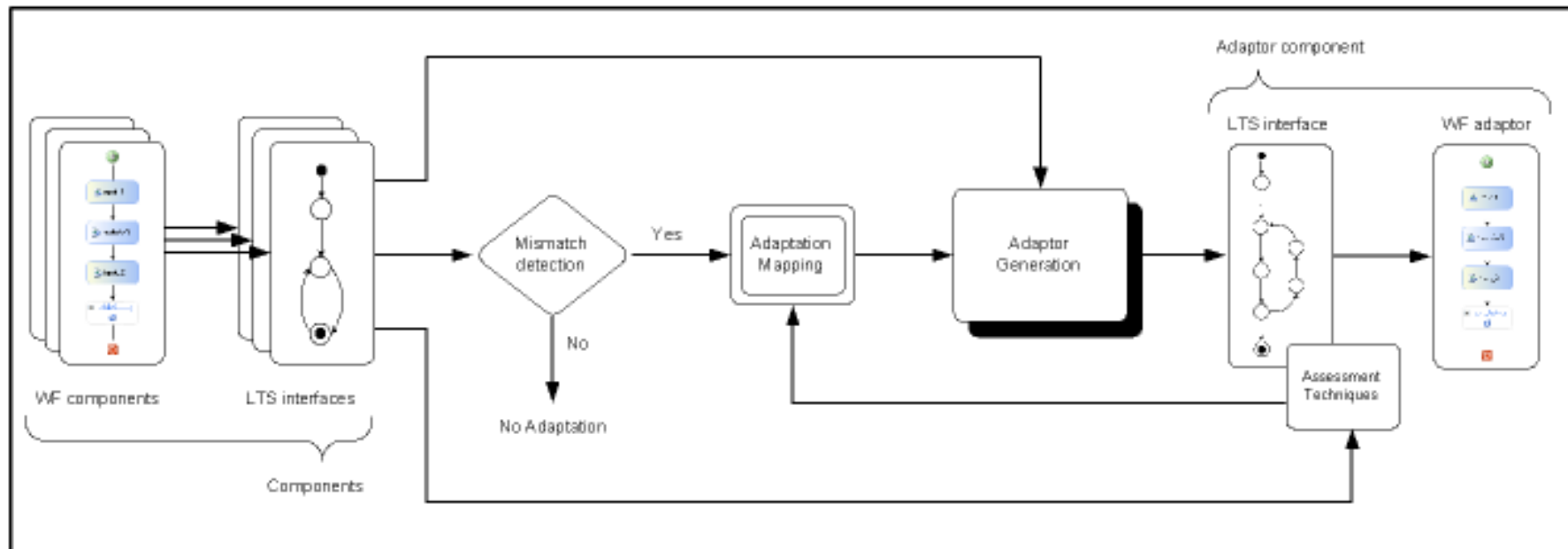
- ▣ relate **adaptor** generation proposals with existing **implementation platforms**

## □ Goals

- ▣ dealing with **behavioural mismatch**
- ▣ building a **workflow-based adaptor** through an automaton generated by the composition between different components
- ▣ experimenting with the **WF/.NET 3.0 platform comparing** the results with other languages or platforms, such as **BPEL**
- ▣ benefiting to the wide number of people that use the .NET Framework in private companies

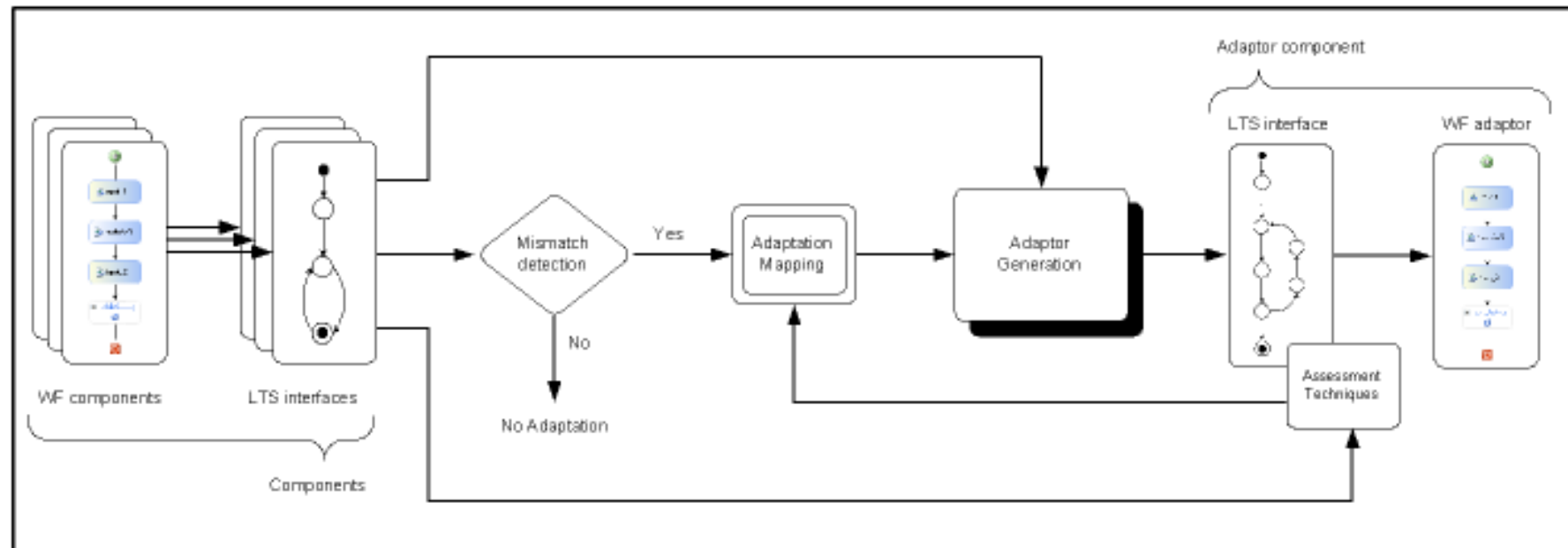
## Proposal

- relate **adaptor** generation proposals with existing **implementation platforms**



## Proposal

- relate **adaptor** generation proposals with existing **implementation platforms**



## Next

- extending**, **formalising** and **implementing** the current proposal



# WF (Windows Workflow Foundation)

- ❑ WF is the programming model, engine and tools for quickly **building workflow** enabled applications on Windows
- ❑ A **workflow** is a set of activities stored as a model that describe a real world process, and can be represented by means a graph, like a flowchart, a state diagram or based on rules
- ❑ WF consist of:
  - ▶ **Execution engine**
  - ▶ **Rules engine**
  - ▶ Number of **Activities**
  - ▶ Number of supporting **runtime services**
  - ▶ Designer allowing developers to design their **workflows graphical on VS2005** (Graphical debugger)
- ❑ The engine is designed building the workflow as **code** constructs, in a declarative fashion using **XAML**, or both
- ❑ The engine allows alteration of the executing **workflow at runtime**




























# WF (Windows Workflow Foundation)

- WF is the programming model, engine and **workflow** enabled applications on Windows
- A **workflow** is a set of activities stored in a file that represent a real world process, and can be represented as a flowchart, a state diagram or based on a text description
- WF consist of:
  - ▶ Execution engine
  - ▶ Rules engine
  - ▶ Number of **Activities**
  - ▶ Number of supporting **runtime services**
  - ▶ Designer allowing developers to design workflows in **VS2005** (Graphical debugger)
- The engine is designed building the workflow in a declarative fashion using **XAML**, or a text description
- The engine allows alteration of the execution



- An **activity** is a step in a workflow (execution, reuse and composition) with properties, events, and methods

 An **activity** is a step in a workflow (execution composition) with properties, events, and n

-  **Windows Workflow**
-  Pointer
-  Sequence
-  Parallel
-  While
-  IfElse
-  Listen
-  EventDriven
-  Delay
-  ConditionedActivityGroup
-  Replicator
-  ExceptionHandler
-  Throw
-  Compensate
-  Code
-  InvokeWebService
-  InvokeWorkflow
-  InvokeMethod
-  EventSink
-  UpdateData
-  SelectData
-  WaitForData
-  WaitForQuery
-  WebServiceReceive
-  WebServiceResponse
-  Suspend
-  Terminate

- ❑ An **activity** is a step in a workflow (execution, reuse and composition) with properties, events, and methods
- ❑ **Code**: executes user code provided for execution within the workflow
- ❑ **Terminate**: finalises the execution of a workflow
- ❑ **InvokeWebService**: calls a WS and receives the requested service
- ❑ **WebServiceInput**: data reception wrt invoked WS
- ❑ **WebServiceOutput**: data sending wrt invoked WS
- ❑ **Sequence**: executes a group of activities in order
- ❑ **IfElse**: corresponds to an if-then-else cond expression
- ❑ **Listen**: defines a set of EventDriven that wait for a specific event; one of the event is fired when the msg is received
- ❑ **While**: defines a set of activities that are fired as many times as its cond is true

## WF and BPEL

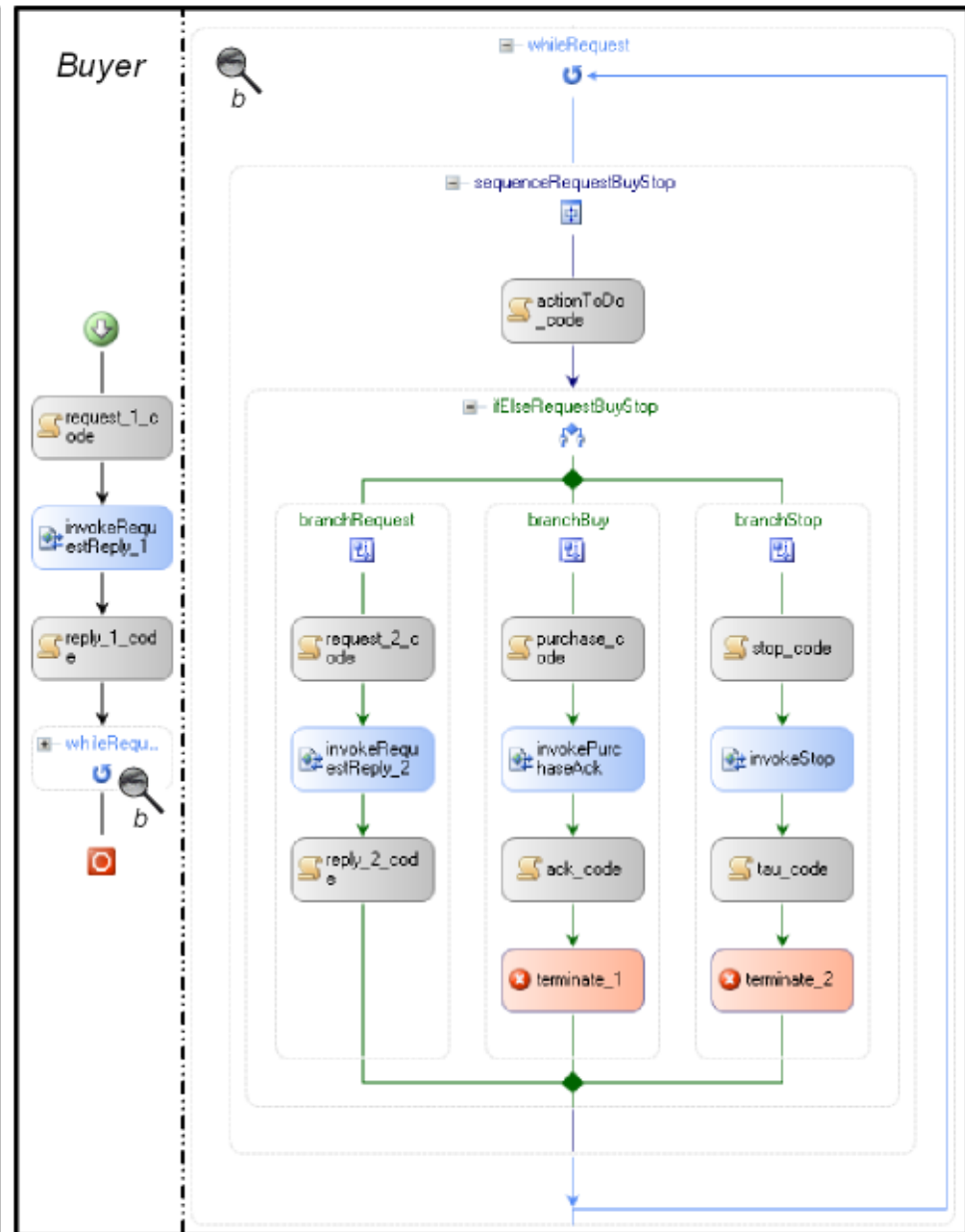
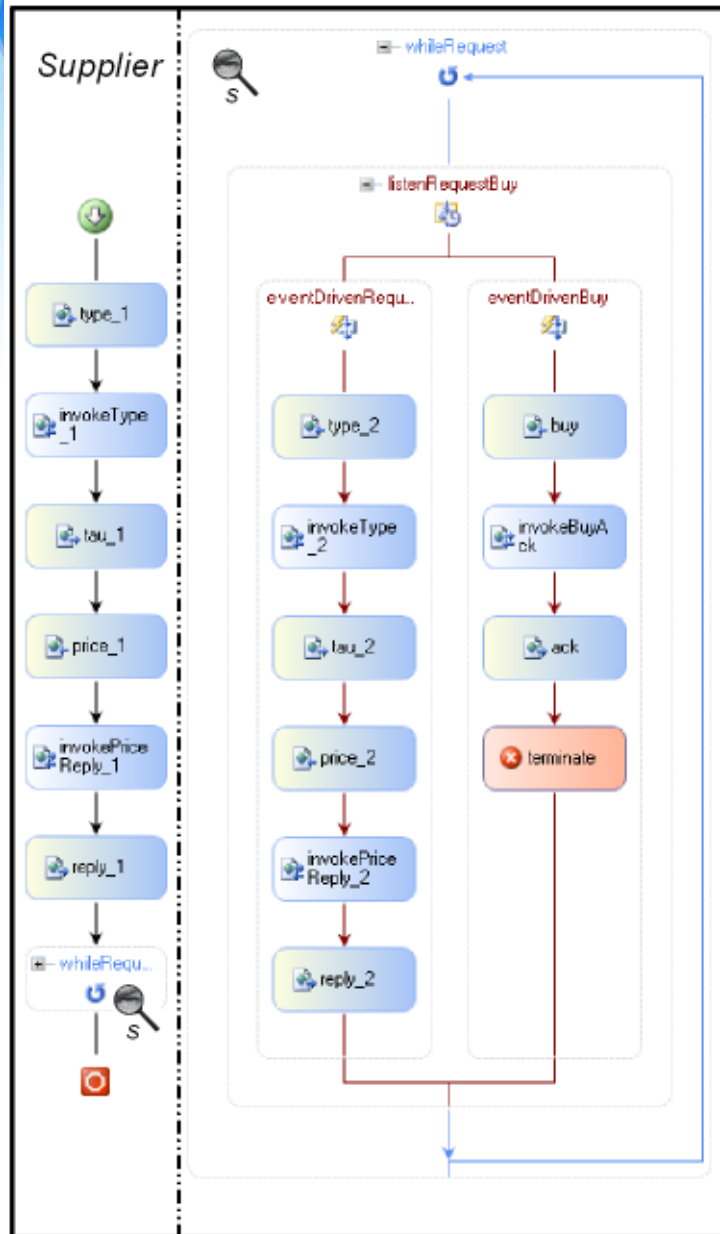
- ▶ allow to design **Web services**; WF can also be used to implement any kind of software **components**
- ▶ their respective platforms (.NET Framework 3.0 on Visual Studio 2005 – Microsoft and Java App Server in Netbeans Enterprise) make
  - the implementation easier thanks to their **workflow-based graphical** support
  - the **automated** generation of most of the underlying **code** (XML+C# and XML+Java)

▶ We focus on WF as an interesting **alternative to BPEL** that has not been studied yet



## Case Study: On-Line Computer Sale

- ❑ System to sell computer material (PCs, laptops, PDAs)
- ❑ Two components, *Supplier* and *Buyer*, implemented using WF/.NET
- ❑ WF workflows for the *Supplier* and *Buyer* components



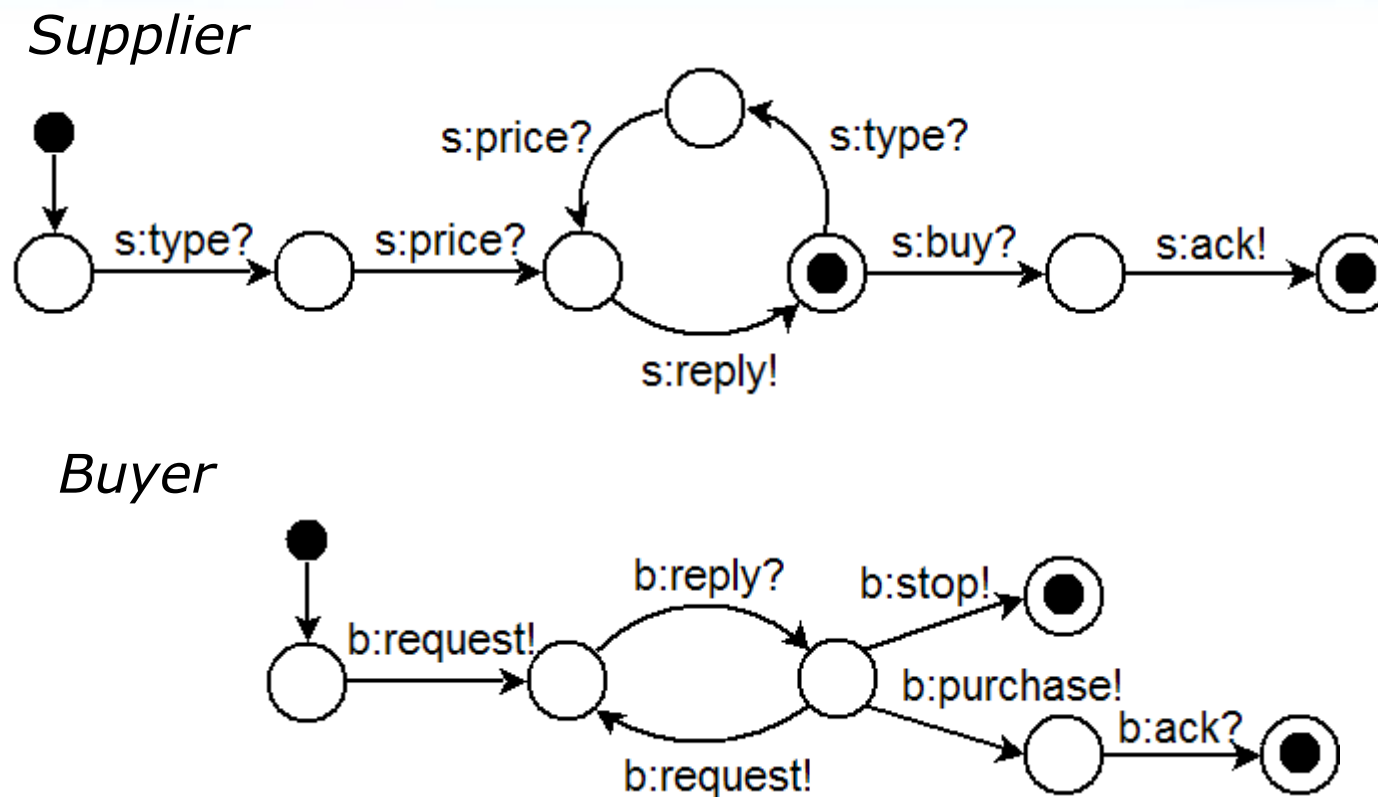
# Composition and Adaptation of WF Components

## ☐ Extraction of the Behavioural Interfaces

- ☐ **While** and **IfElse** cond are abstrated, since LTS does not support the description of data expressions
- ☐ **WebServiceInput** / **WebServiceOutput** identified with **tau** are traslated as **tau** transitions in the LTS
- ☐ Initial and final states in LTS come from workflow (including **Terminate** in final states)
- ☐ **Messages** in *Supplier* and *Buyer* come from the output / input parameters in theirs invoke activities
- ☐ **InvokeWebService** are made abstract, because are interactions with external components
- ☐ All **tau** transitions in both LTSs (C#) and **WebServiceOutput** have been removed to favour readability

# Composition and Adaptation of WF Components

## Extraction of the Behavioural Interfaces

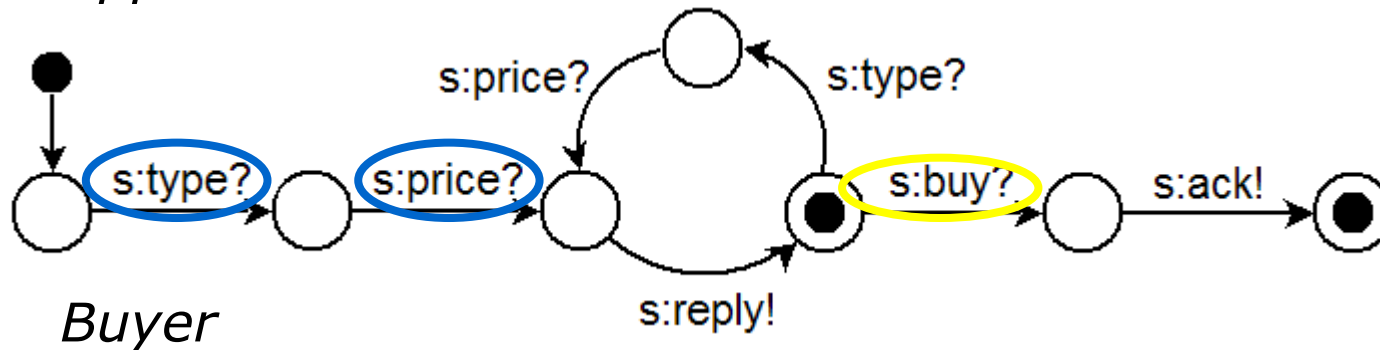


# Composition and Adaptation of WF Components

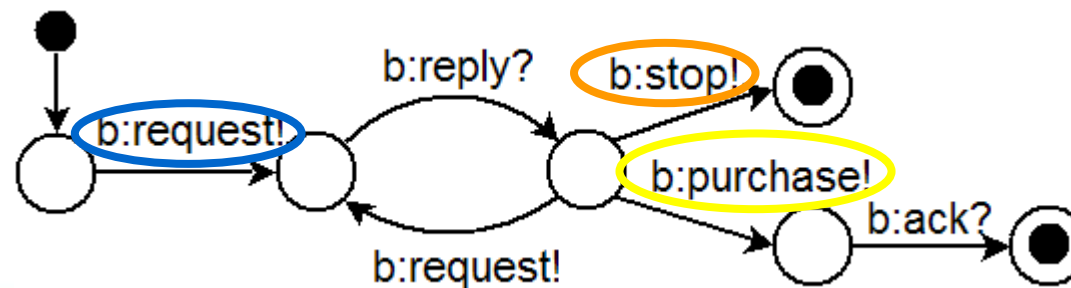
## ❑ Mismatch Cases

- ▶ **name mismatch:** purchase! / buy?
- ▶ **mismatching number of msgs:** request! / type? price?
- ▶ **independent evolution:** stop!

*Supplier*



*Buyer*



## ❑ Mismatch Cases

- ❏ **name mismatch**: purchase! / buy?
- ❏ **mismatching number of msgs**: request! / type? price?
- ❏ **independent evolution**: stop!

## ❑ Adaptation Mapping

$$V_{req} = \langle b:request!, s:type? \rangle$$

$$V_{price} = \langle b:\varepsilon, s:price? \rangle$$

$$V_{reply} = \langle b:reply!, s:reply? \rangle$$

$$V_{stop} = \langle b:stop!, s:\varepsilon \rangle$$

$$V_{buy} = \langle b:purchase!, s:buy? \rangle$$

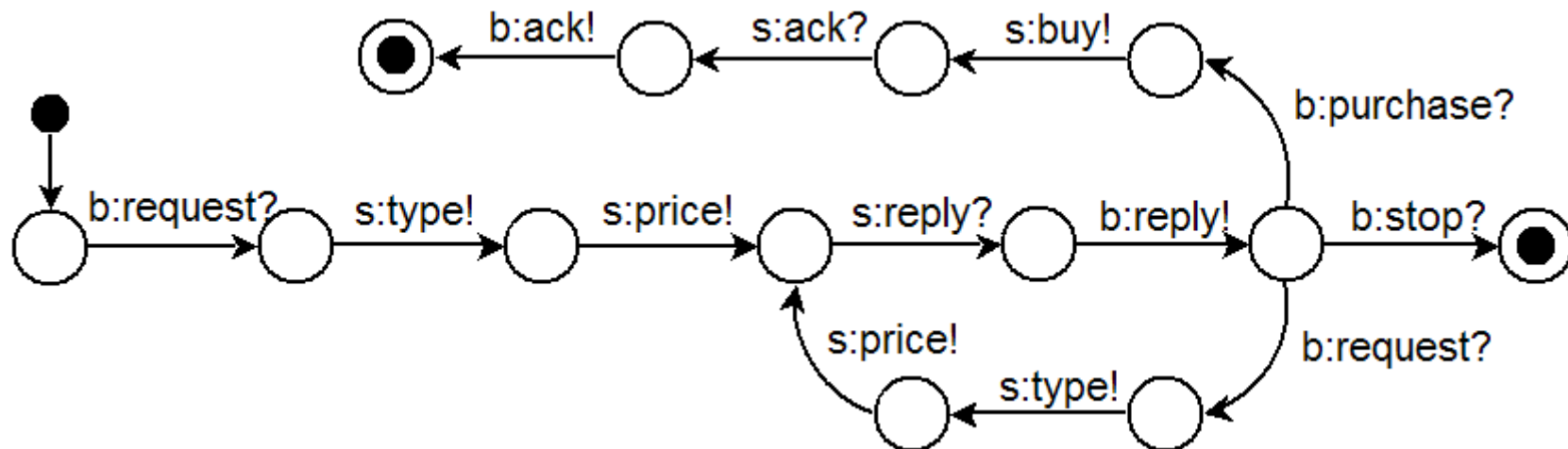
$$V_{ack} = \langle b:ack?, s:ack! \rangle$$

- ❏  $V_{buy}$  solve the name mismatch
- ❏  $V_{req}$  and  $V_{price}$  solve the mismatching number of msgs
- ❏  $V_{stop}$  resolve the independent evolution

# Composition and Adaptation of WF Components

## Generation of the Adaptor Protocol

- Given a set of component LTSs and a mapping: **generate the adaptor protocol automatically** (CanalEtAI-FMOODS06)
- Adaptor** LTS

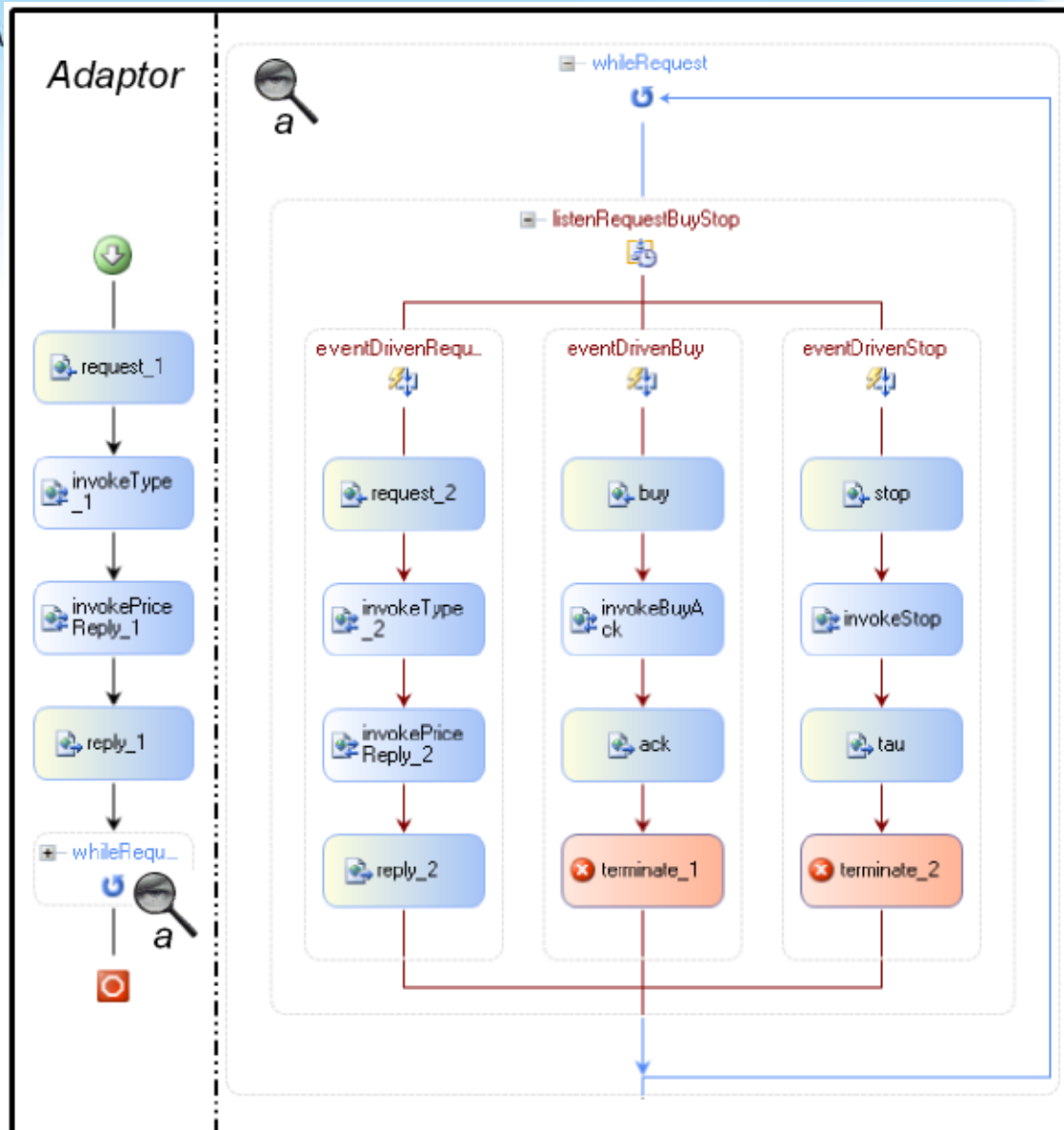


# Composition and Adaptation of WF Components

## Implementation of the *WF Adaptor*

- From the adaptor LTS a corresponding WF component is obtained using the reversed process sketched previously
- Every emission followed by a replay is encoded as an **InvokeWebService**
- Other input/output events are translated using **WebServiceInput** / **WebSeviceOutput**
- WF workflow for the *Adaptor* component

# Composition and Adaptation of WF Components



- ❑ A simple yet realistic example how existing **model-based adaptation** approaches can be related to **implementation platforms** such as WF/.NET
- ❑ We have had to face and work out specificities of the WF platform such as the use of **tau WebServiceOutput**, or of several **InvokeWebService** in one session
- ❑ Promising, since it shows that **software adaptation can be of real use**, and helps the developer in building software applications by reusing components or services
- ❑ Perspectives and ongoing works (CuboEtAI-FACS07):
  - ▶ **automating** the **LTS extraction from WF** workflows
  - ▶ **generating WF** workflows from the adaptor **LTS**
  - ▶ **automating** the **mismatch detection**, and generating the list of mismatch from a set of component LTSs
  - ▶ beyond mismatch detection, tackling **verification** of WF components
  - ▶ supporting techniques to help the designer to write the **mapping** out, and to generate **automatically** part of it
  - ▶ experimenting on the implementation of the adaptors using **BPEL** and the Netbeans Enterprise to compare with WF

# Thank you for your attention!

- ☐ Javier Cubo, G.Salaün, C.Canal, E. Pimentel, P. Poizat
  - ☐ University of Málaga
  - ☐ {cubo,salaun,canal,ernesto}@lcc.uma.es
  - ☐ INRIA/ARLES Project-Team, and IBISC FRE 2873 CNRS - Université d'Évry, France
  - ☐ pascal.poizat@inria.fr

