A Model-Based Approach to the Verification and Adaptation of WF/.NET Components

Javier Cubo, Gwen Salaün, Carlos Canal, Ernesto Pimentel

Pascal Poizat

Motivations (1/2)

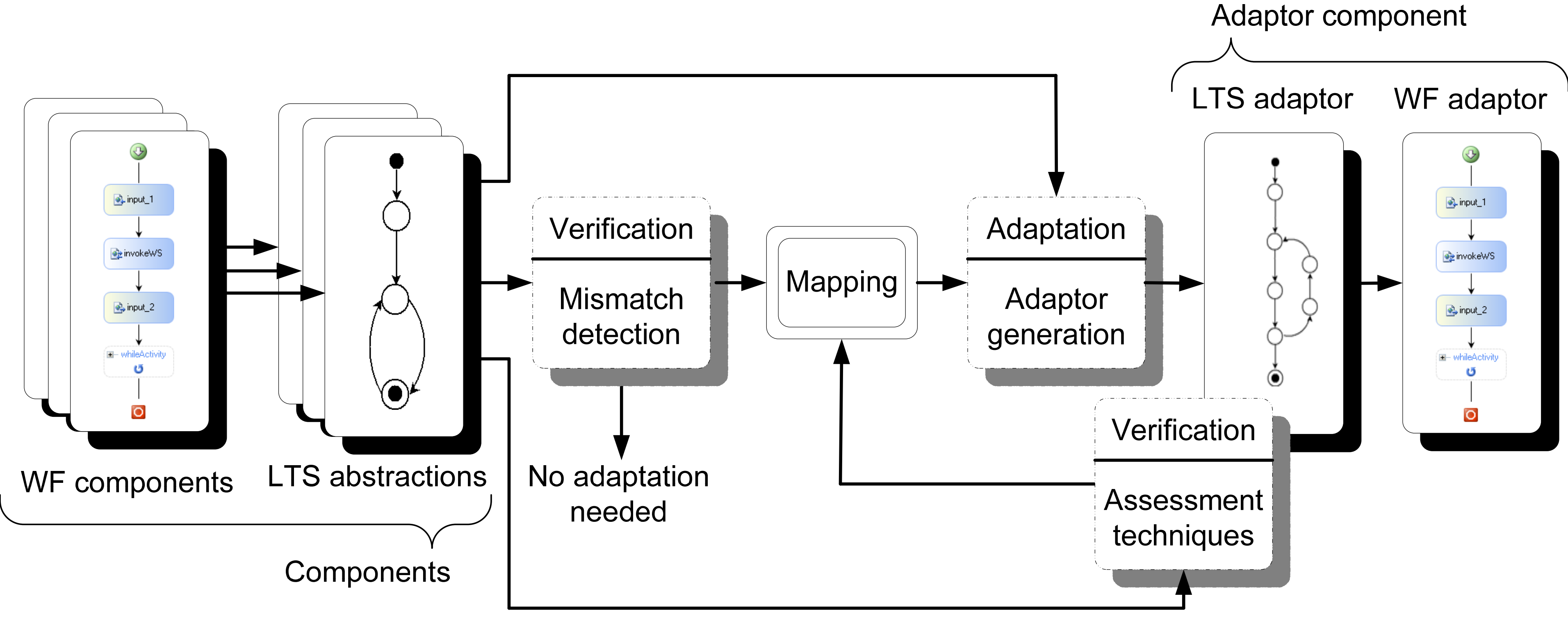
- Component-based systems are built by composition and reuse of existing components
- Several levels of interoperability in component description:
  - + Signatures, behaviours
  - - Semantic aspects, quality of services

- A component is seldom used directly as it is and needs some adaptations to solve mismatch problems
- Many approaches dedicated to model-based adaptation $\Rightarrow$ generation of adaptor components as automatically as possible

Motivations (2/2)

- Goal: relate adaptor generation methods with existing programming languages and platforms
- Very few approaches relate their results with existing programming languages and platforms:
  - COM/DCOM [InverardiTivoli-JSS03], BPEL [BrogiPopescu-ICSOC06], SCA components [MotahariNezhadEtAl-WWW07]
- Experiments with Windows Workflow Foundation (WF) in the .NET 3.0 Framework: alternative solution, widely used
- Key-points of our proposal:
  - Focus on behavioural descriptions of components, LTSs
  - Translation between WF and LTSs: extraction (wf2lts), and generation (lts2wf)
  - Adaptation but also verification of WF

Overview of our Approach

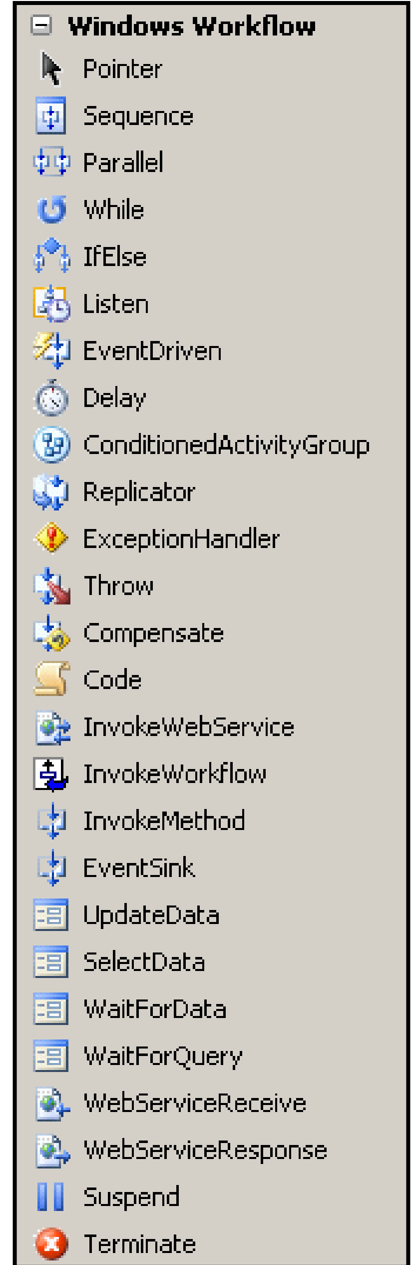


Outline of the Talk

- WF Workflow Notation
- Extracting LTSs from Abstract WF Workflows
- Adaptation and Verification
- Generating Abstract WF Workflows from LTSs
- Concluding Remarks and Future Work

WF Overview

- WF is the programming platform for building workflow-based applications on Windows
- A workflow is composed of a set of activities:
  - Code: executes user code written in Visual Basic or C#
  - Terminate: finalises the execution of a workflow
  - InvokeWebService: calls a WS, receives the requested service back
  - WebServiceInput: receives data from a WS
  - WebServiceOutput: sends data to a WS
  - Sequence: executes a set of activities in a precise order
  - IfElse: executes one branch depending on the condition
  - Listen: defines several branches, one of which is fired when the corresponding message is received
  - While: defines a looping activity guarded by a condition



Abstract WF Workflow

A ::= Code
| Terminate
| InvokeWebService(O_1,…,O_n,I)
| WebServiceInput(I_1,…,I_n)
| WebServiceOutput(O)
| Sequence(A_1,A_2)
| IfElse((C_1, A_1),…,(C_n, A_n),A_{n+1})
| Listen(E_1,…,E_n)
| While(C,A)

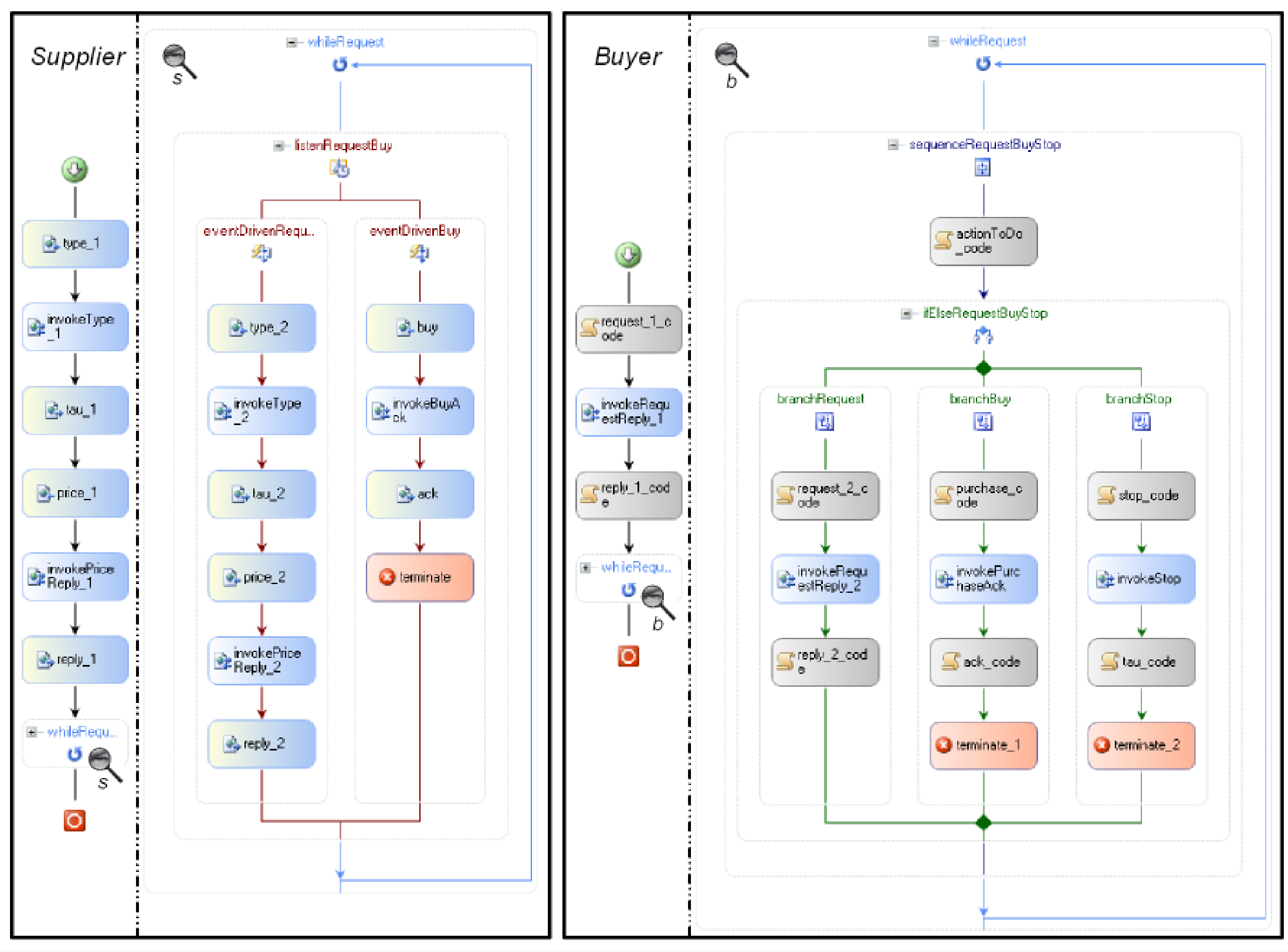
E ::= EventDriven(WebServiceInput(I),A)

Output parameters

Input parameters

Boolean condition

Running Example: On-line PC Sale



Ex.: Supplier Abstract Workflow

```
While
     ( true
      Listen
           ( EventDriven
                 ( WebServiceInput(type),
                   ...
                 ),
            EventDriven
                 ( WebServiceInput(buy),
                   Sequence
                        ( InvokeWebService(buy,ack),
                          Sequence
                               ( WebServiceOutput(ack),
                                 Terminate
                               )
                        )
                 ) ...
```

Outline of the Talk

- WF Workflow Notation
- Extracting LTSs from Abstract WF Workflows
- Adaptation and Verification
- Generating Abstract WF Workflows from LTSs
- Concluding Remarks and Future Work

Overview of awf2lts (1/3)

Code → τ

Terminate →

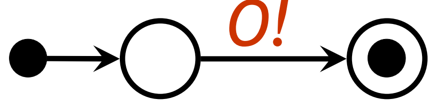
InvokeWebService(O_1,…,O_n,I)

→ $O_1!$ … $O_n!$ $I?$

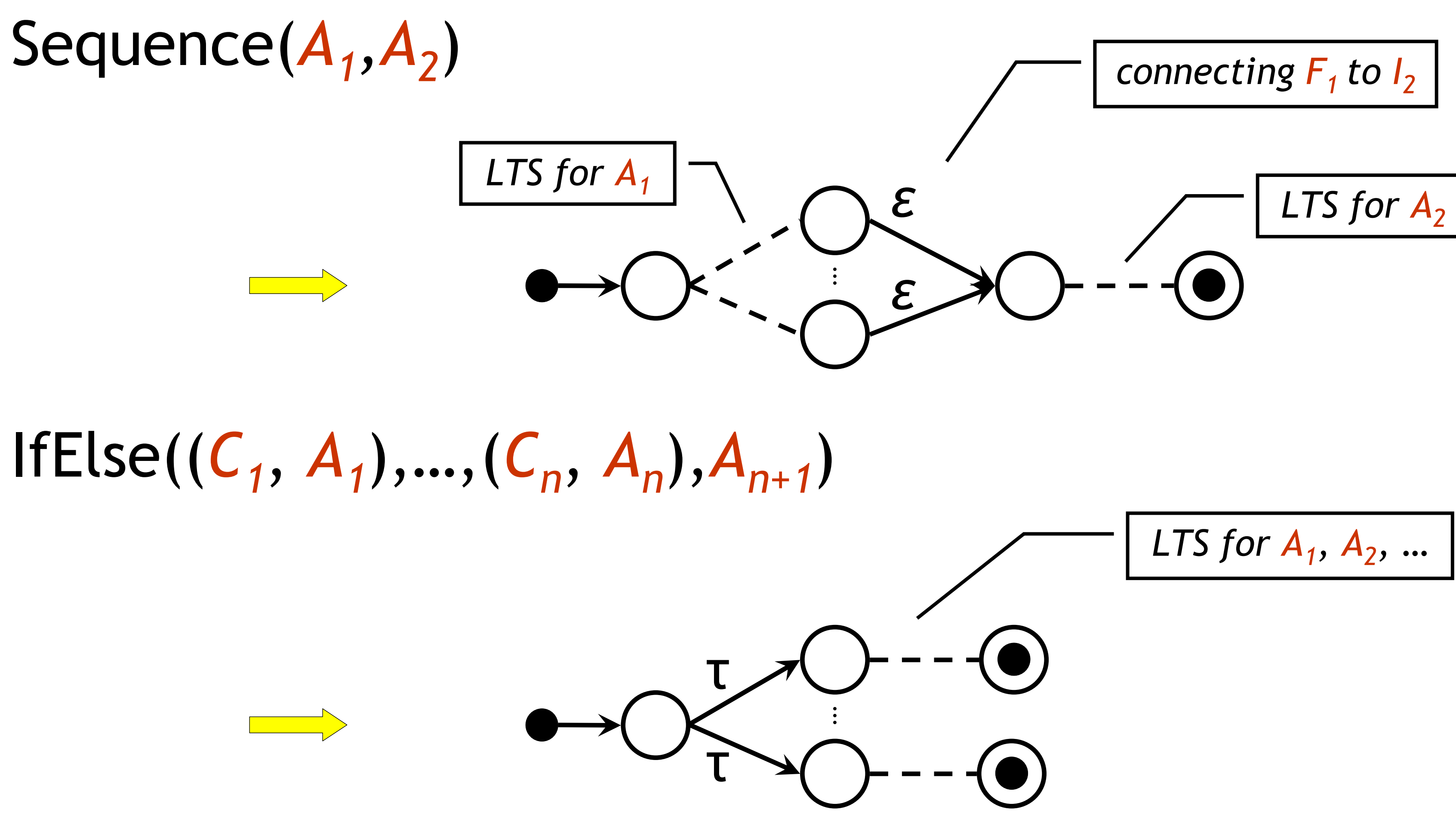
WebServiceInput(I_1,…,I_n)

→ $I_1?$ … $I_n?$

WebServiceOutput(O) →

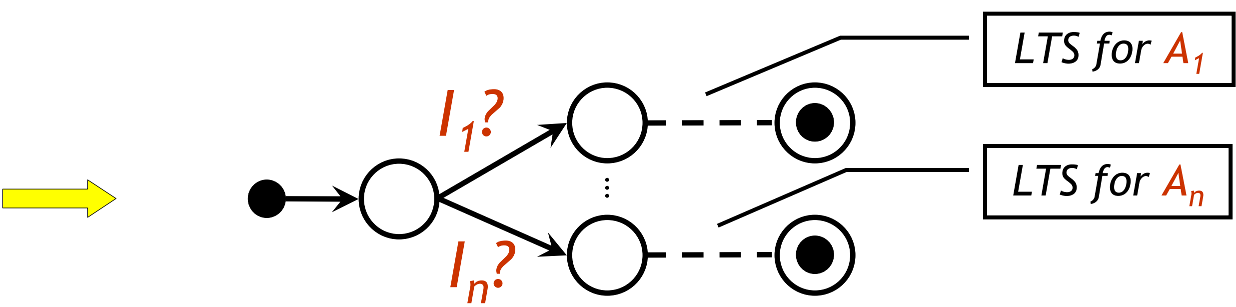


Overview of awf2lts (2/3)

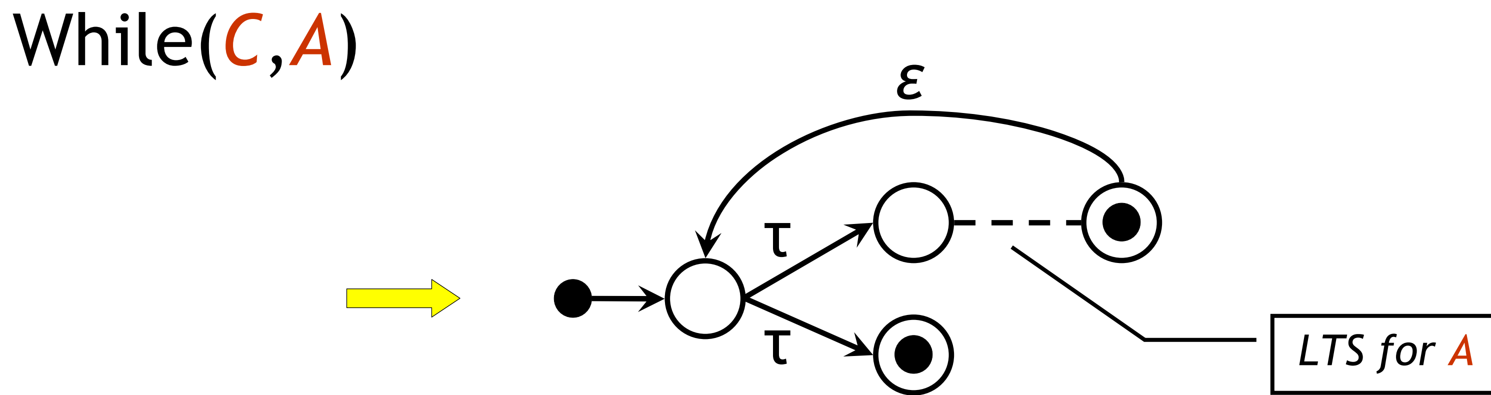


Overview of awf2lts (3/3)

Listen(EventDriven(WebServiceInput(I_1),A_1), …
EventDriven(WebServiceInput(I_n),A_n))

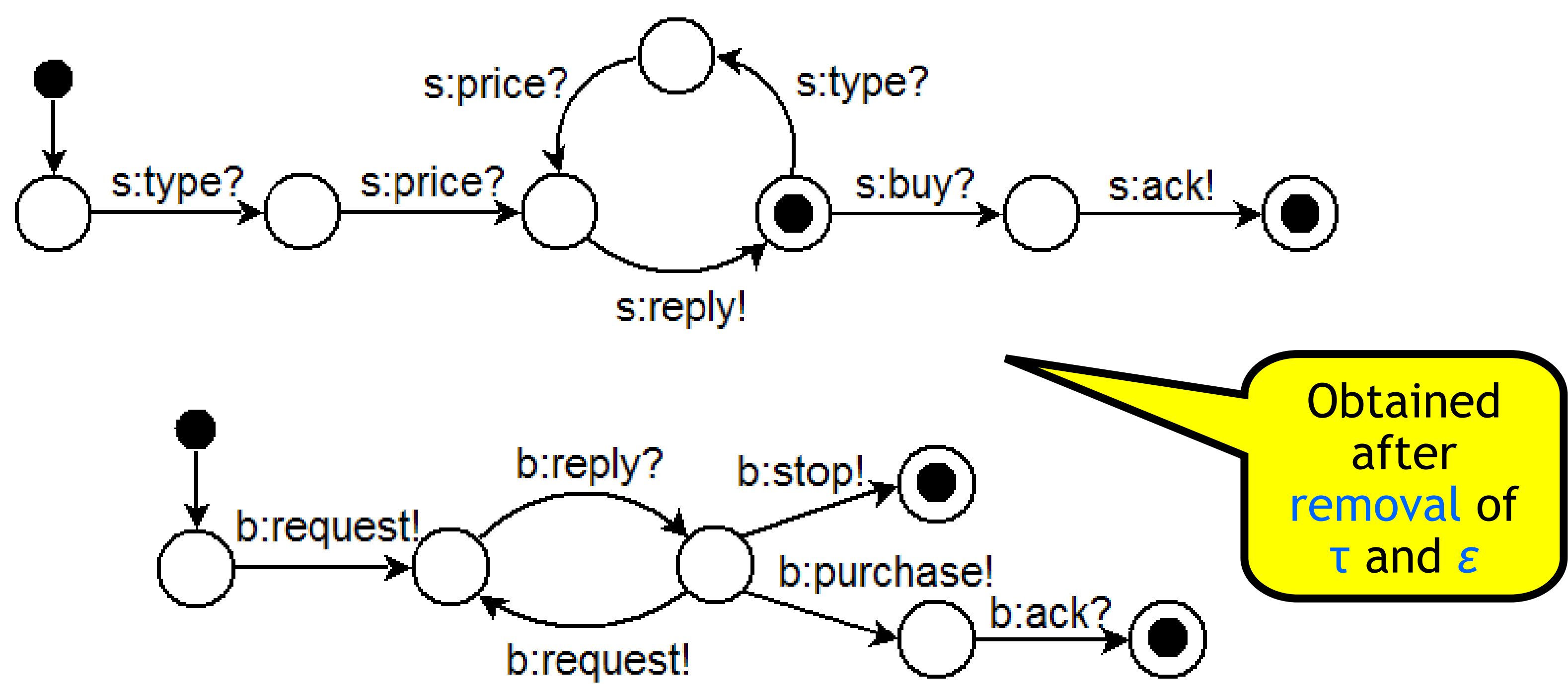


While(C,A)



Ex.: Supplier and Buyer LTS interfaces

- Observable messages are coming from invoke activities, and input/output messages
- Internal activities are dismissed, *eg*. database access

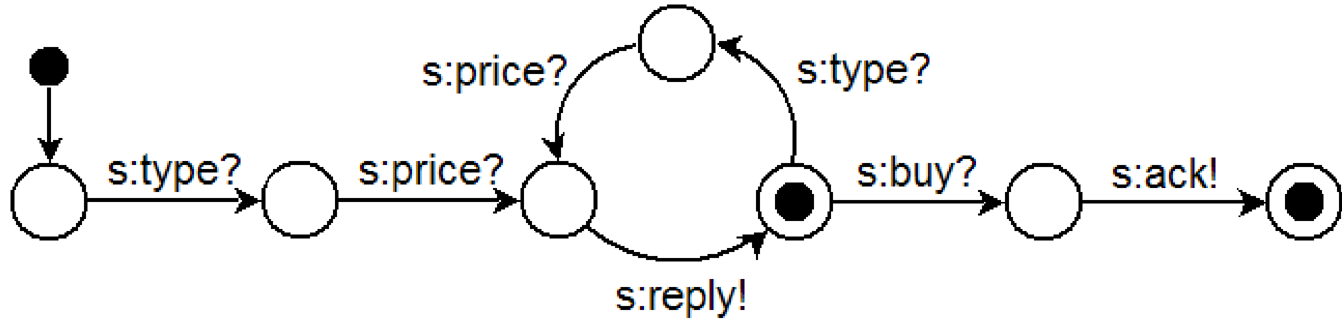


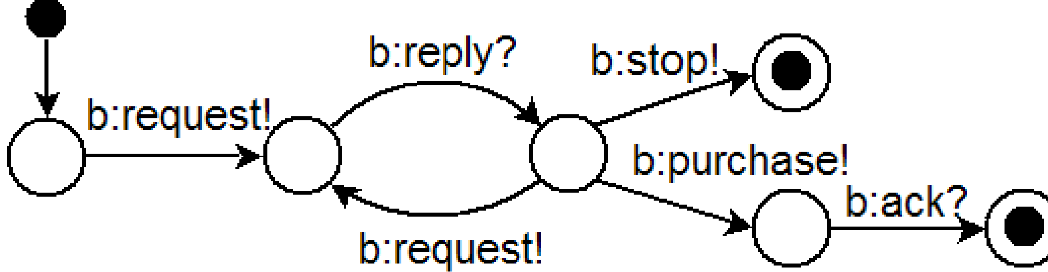
Outline of the Talk

- WF Workflow Notation
- Extracting LTSs from Abstract WF Workflows
- Adaptation and Verification
- Generating Abstract WF Workflows from LTSs
- Concluding Remarks and Future Work

Running Example: Mismatch Cases

- Name mismatch: *purchase!* vs *buy?*
- Mismatching number of messages: *request!* vs *type?* and *price?*
- Independent evolution: *stop!*





Adaptation Mapping

- Vectors to define correspondence between messages
- Adaptation mapping for the running example:

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

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

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

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

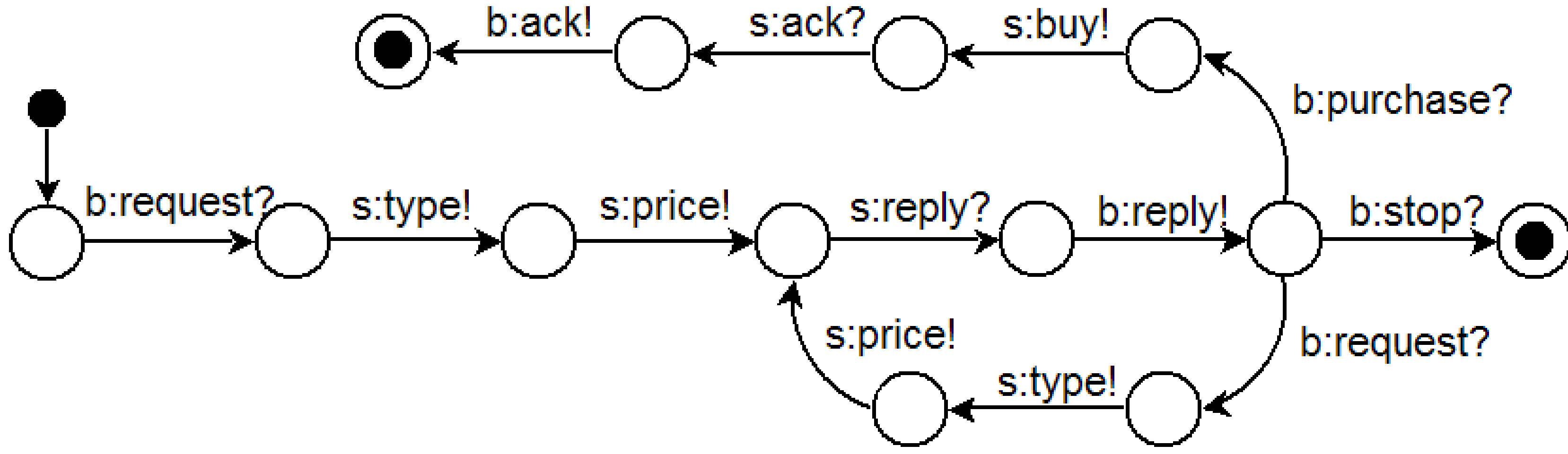
$$V_{buy} = \langle b{:}purchase!,\ s{:}buy? \rangle$$

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

  - V_{buy} solves the name mismatch
  - V_{req} and V_{price} solve the mismatching number of messages
  - V_{stop} makes explicit the independent evolution

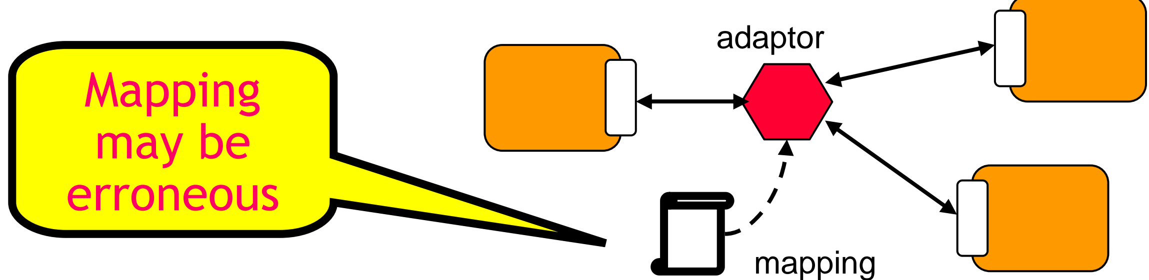
Adaptor Protocol Generation

- Given component LTSs and a mapping: generate the adaptor protocol automatically using existing tools, *eg*. Compositor+Scrutator [MateescuEtAl-ASE07]
- Running example: resulting adaptor LTS



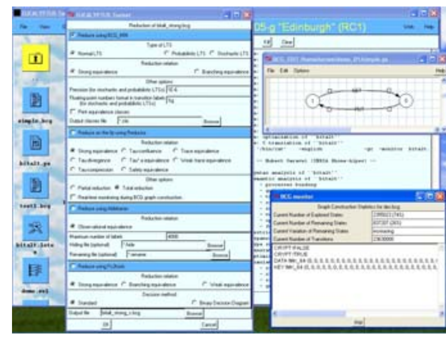
Assessment Techniques

- Goal: verification of the whole system, LTSs+adaptor



- Model-checking tools and techniques, here CADP
  - computation of the whole system LTSs+adaptor using EXP.OPEN
  - writing of temporal properties (liveness, safety) in mu-calculus
  - automatic verification using Evaluator

```
[true* . "b_request"]
  mu X. (<true> true and
    [not ("b_stop" or "b_purchase" or "b_request")]X)
```



Outline of the Talk

- WF Workflow Notation
- Extracting LTSs from Abstract WF Workflows
- Adaptation and Verification
- Generating Abstract WF Workflows from LTSs
- Concluding Remarks and Future Work

Overview of lts2awf (1/2)

- Initial state in both adaptor LTS and abstract WF
- Final states encoded as Terminate activities
- Next, two successive steps to extract WF activities:
  - First, extraction of WF message activities

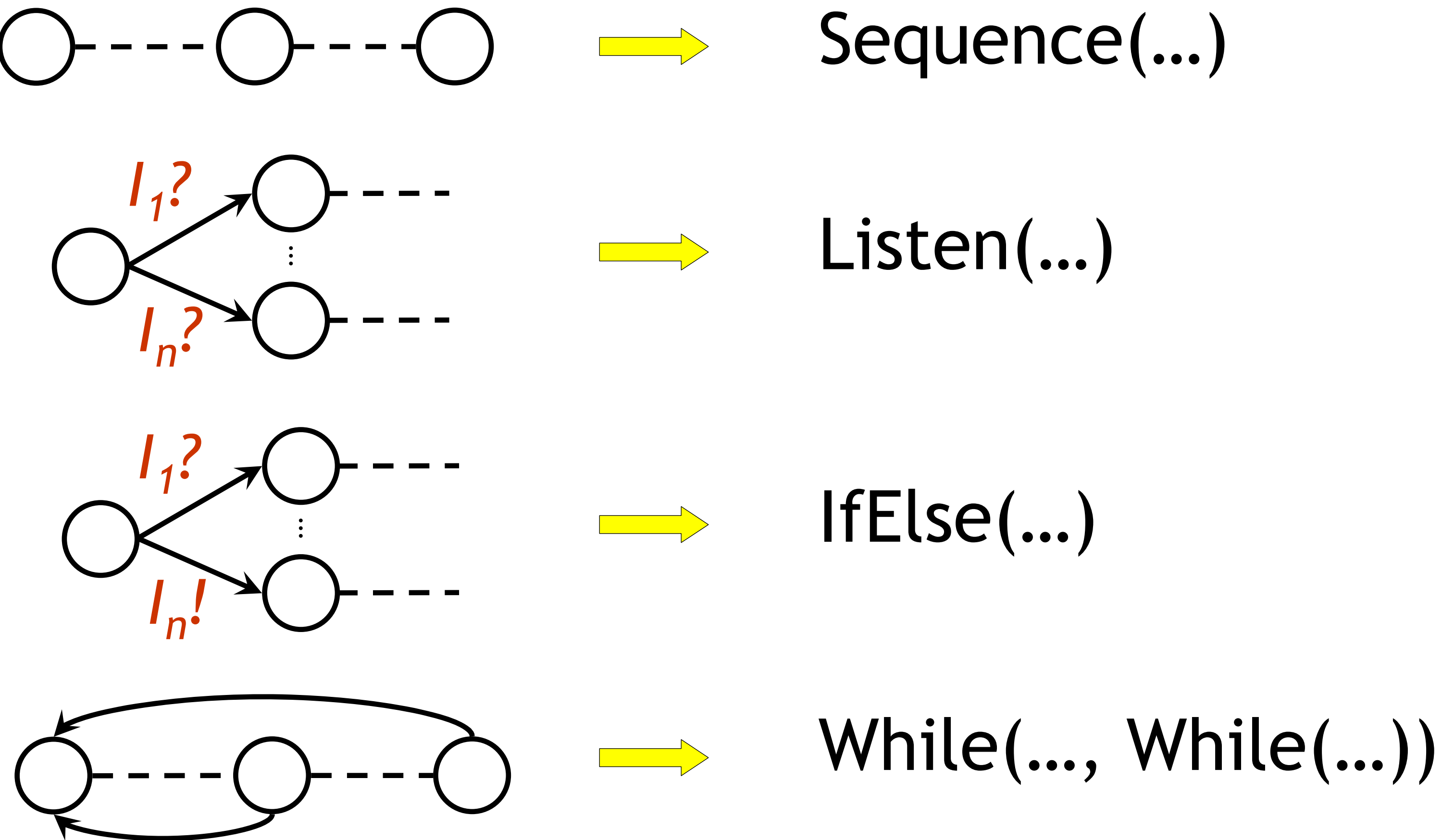
$\bigcirc \xrightarrow{O_1!} \bigcirc - - - \bigcirc \xrightarrow{O_n!} \bigcirc \xrightarrow{I?} \bigcirc$ ⇒ InvokeWS(O_1,…,O_n,I)

$\bigcirc \xrightarrow{I_1?} \bigcirc - - - \bigcirc \xrightarrow{I_n?} \bigcirc$ ⇒ WSInput(I_1,…,I_n)

$\bigcirc \xrightarrow{O!} \bigcirc$ ⇒ WSOutput(O)

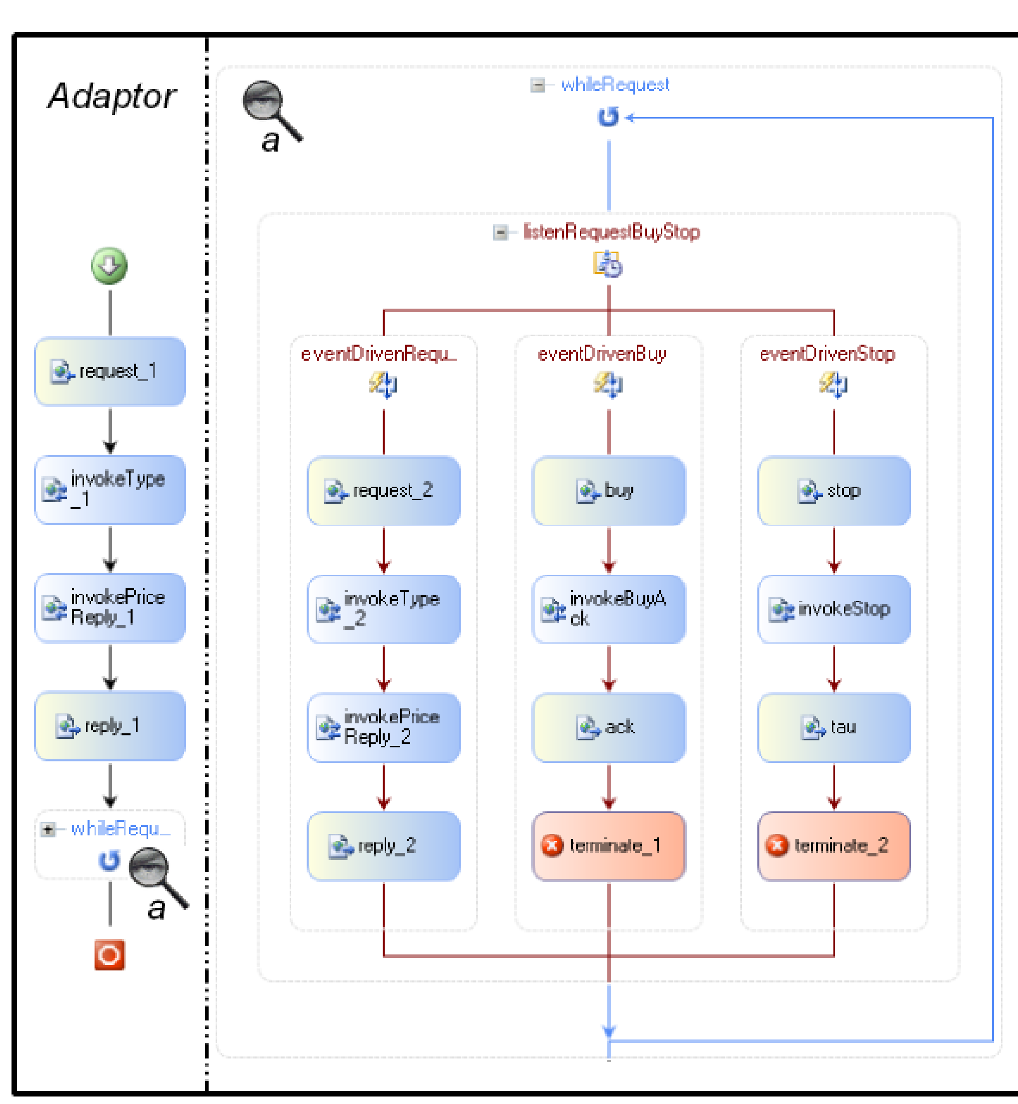
Overview of lts2awf (2/2)

- Second, extraction of WF structuring activities



Running Example: Adaptor Workflow

- lts2awf → abstract WF
- Intervention of the designer to:
  - Concretise conditions
    - ⇒ IfElse and While
  - Add C# pieces of code
  - Specify component invocations (addresses)
- Obtaining of the concrete workflow

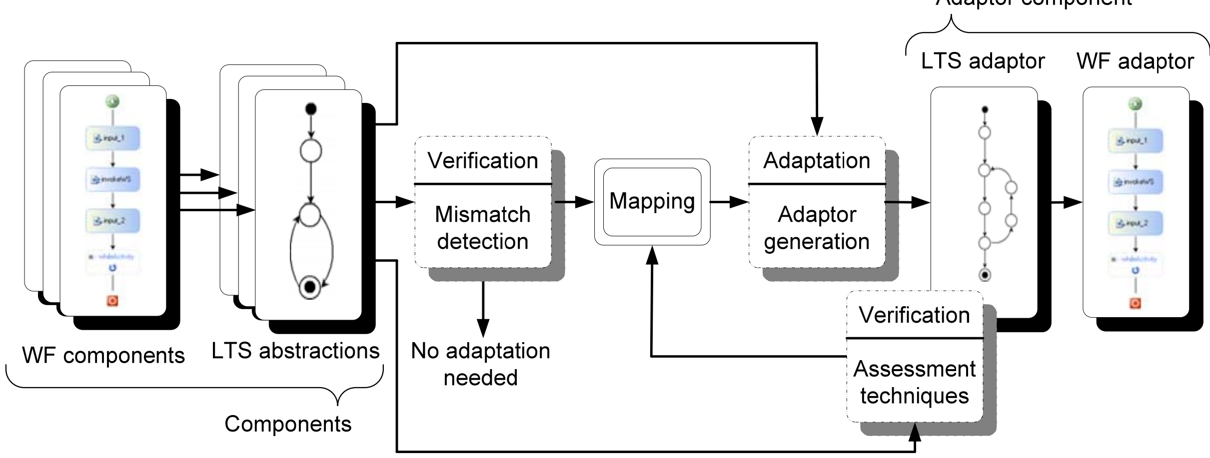


Outline of the Talk

- WF Workflow Notation
- Extracting LTSs from Abstract WF Workflows
- Adaptation and Verification
- Generating Abstract WF Workflows from LTSs
- Concluding Remarks and Future Work

Concluding Remarks

- A first and early proposal to apply verification and adaptation techniques to WF components



- Illustration of our proposal on a simple example
- Promising because shows that software adaptation can really help while reusing software entities
- However, still a lot to do …

Future Work

- Short-term perspectives:
  - Formalisation of function lts2awf
  - Implementing our translations (AWF<->LTSs) in a prototype tool
  - Implementing translators between WF workflows (XML+C#) and abstract workflows in both directions
  - Experimenting the proposal on a set of case studies

- Long-term perspectives
  - More WF activities: Parallel, message parameter, data, etc
  - Extending the LTS model with respect to these new activities
  - Verification and adaptation techniques for this new model
  - Extension of the prototype tool to deal with these enhancements