

# Model Checking Business Processes

Ioannis G. Baltopoulos  
([ioannis.baltopoulos@cl.cam.ac.uk](mailto:ioannis.baltopoulos@cl.cam.ac.uk))



22nd British Colloquium for Theoretical Computer Science  
BCTCS 2006, Swansea University

- Introduction
  - Motivation
  
- Theoretical Background
  - Polyadic  $\pi$ -calculus
  - Modal  $\mu$ -calculus
  
- WS-BPEL
  - Language Concepts
  - Translations

# Motivation

- Web Services are the latest evolutionary step in Distributed Computing Technologies
- Core benefits are
  - loosely coupled
  - ease of integration
  - allow service reuse
- Alone, Web Services are not that interesting; combining them and creating workflows is much more interesting
- OASIS has proposed the Business Process Execution Language (BPEL) that allows the Web Service coordination for creating long lasting business processes.

# Project Goal

## Problem Statement

Create a system based on formal methods that allows the property checking of real-world business processes.

Useful for complex systems which are additionally

- **Concurrent:** Distributed systems give rise to concurrency
- **Quality-critical:** Applications where failure is not dangerous but economically expensive
- **Safety-critical:** Applications where failure may endanger human life
- **Security-critical:** Applications where failure means unauthorized access to sensitive information

# Aims & Contributions

- Provide a formal semantics in  $\pi$ -calculus for a subset of the BPEL language
- Implement an automated tool for property checking of business processes expressed in BPEL
- Use the tool on real world examples to identify problematic situations

# Overview

## $\pi$ -calculus

The  $\pi$ -calculus is a process algebra (a mathematical model) for describing and analysing systems consisting of agents/processes, whose configuration (interconnections) or neighbourhood is continually changing.

## $\mu$ -calculus

- The propositional modal  $\mu$ -calculus is a powerful language for expressing properties of transition systems by using least and greatest fixpoint operators.
- It is interesting because there exist efficient model checking algorithms for this formalism.

# Assumptions

We assume

- the existence of the infinite set  $\mathcal{N}$  of **Names**<sup>1</sup>; they have no structure. Lower case English letters are members of the set ( $x, y, z, \dots \in \mathcal{N}$ )
- finite sequences  $\tilde{a}$ . ( $\tilde{a} \stackrel{\text{def}}{=} u_1, u_2, \dots, u_n$ )
- the existence of a basic kind of entity, a *process* and an associated infinite set  $\mathcal{P}$  of **Processes**. Uppercase English letters are members of the set ( $P, Q, R, \dots \in \mathcal{P}$ )

Set Symbol	Name	Members in Set
$\mathcal{N}$	Names	$a, b, c, \dots$
$\overline{\mathcal{N}}$	Conjugate Names	$\bar{a}, \bar{b}, \bar{c}, \dots$

Table: Names

<sup>1</sup>Naming and Computers

$\pi$ -calculus syntax

$P ::= M$	(Normalised Process)
$P_1 \mid P_2$	(Parallel composition)
$!P$	(Replication)
$(\mathbf{new} \ x) \ P$	(Restriction)
$M ::= \emptyset$	(Nil Process)
$\pi.P$	(Prefixing)
$M_1 + M_2$	(Choice)
$\pi ::= \bar{a}\langle\tilde{x}\rangle$	(Output)
$a(\tilde{x})$	(input)
$\tau$	(Silent)
$[x = y]\pi$	(Match)

Table:  $\pi$ -calculus Language Syntax

# $\mu$ -calculus syntax

$\mu$ -calculus formulas ( $\Phi$ ) are built up from:

- *boolean connectives* ( $\wedge, \vee, \neg$ )
- *modal operators* ( $\langle \rangle, []$ )
- *maximum fixed points* ( $\mu, \nu$ )
- *variables* ( $Z$ )
- *actions* ( $K \subset \mathcal{A}$ )

According to the following syntax

$$\begin{aligned} \Phi ::= & \text{tt} \mid \text{ff} \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \\ & \mid [K]\Phi \mid \langle K \rangle \Phi \mid \nu Z. \Phi \mid \mu Z. \Phi \end{aligned}$$

Table:  $\mu$ -calculus Language Syntax

# Semantics I

Formulas in the  $\mu$ -calculus are interpreted relative to a transition system (Kripke Structure). Equipped with a Kripke Structure  $M = (S, S_0, R, L)$  and an environment  $\epsilon : Z \rightarrow 2^S$ , a formula  $\Phi$  is interpreted as a set of states in which  $\Phi$  is true, denoted  $\llbracket \Phi \rrbracket_{M, \epsilon}$ . The set  $\llbracket \Phi \rrbracket_{M, \epsilon}$  is defined recursively as follows:

- $\llbracket p \rrbracket_{M, \epsilon} = \{s \mid p \in L(s)\}$
- $\llbracket Q \rrbracket_{M, \epsilon} = \epsilon(Q)$
- $\llbracket \neg \Phi \rrbracket_{M, \epsilon} = S \setminus \llbracket \Phi \rrbracket_{M, \epsilon}$
- $\llbracket \Phi_1 \wedge \Phi_2 \rrbracket_{M, \epsilon} = \llbracket \Phi_1 \rrbracket_{M, \epsilon} \cap \llbracket \Phi_2 \rrbracket_{M, \epsilon}$
- $\llbracket \Phi_1 \vee \Phi_2 \rrbracket_{M, \epsilon} = \llbracket \Phi_1 \rrbracket_{M, \epsilon} \cup \llbracket \Phi_2 \rrbracket_{M, \epsilon}$

## Semantics II

- $\llbracket \langle a \rangle \Phi \rrbracket_{M, \epsilon} = \left\{ s \mid \exists t \in S. s \xrightarrow{a} t \text{ and } t \in \llbracket \Phi \rrbracket_{M, \epsilon} \right\}$

or intuitively,

*“It is possible to make an  $a$ -transition to a state where  $\Phi$  holds.”*

- $\llbracket [a] \Phi \rrbracket_{M, \epsilon} = \left\{ s \mid \forall t \in S. s \xrightarrow{a} t \text{ implies } t \in \llbracket \Phi \rrbracket_{M, \epsilon} \right\}$

or intuitively,

*“ $\Phi$  holds in all states reachable (in one step) by making an  $a$ -transition.”*

- $\llbracket \mu Z. \Phi \rrbracket_{M, \epsilon}$  is the least fixed point solution

*“The smallest set of tasks such that  $Z = \Phi$ ”*

- $\llbracket \nu Z. \Phi \rrbracket_{M, \epsilon}$  is the greatest fixed point solution

*“The largest set of tasks such that  $Z = \Phi$ ”*

# Satisfiability and Validity

Glueing the two theories together!

For any modal formula  $\Phi$  and a set of processes  $\mathcal{P}$  we say that,

## Satisfiability

A process  $P \in \mathcal{P}$  has, or **satisfies**, the property  $\Phi$ , if some process satisfies it and we write

$$P \models \Phi \quad \text{iff} \quad \exists P \in \mathcal{P}. \Phi \text{ holds}$$

## Validity

A formula  $\Phi$  is valid if all processes  $P \in \mathcal{P}$  satisfy it.

# Interesting Properties

For the purposes of business processes, we will check the following properties

## Deadlock

A process  $P \in \mathcal{P}$  is deadlocked is expressed as

$$P \models [\cdot]\mathbf{ff}$$

## Cancellation

A process can be cancelled, expressed as

$$P \models \mu \text{cancel}.\langle - \rangle \mathbf{tt} \wedge [\text{cancel}]\mathbf{ff}$$

Or any other properties that can be expressed in the  $\mu$ -calculus (an order gets eventually shipped, a bank account will eventually be credited)

# Introduction

- The Business Process Execution Language (BPEL) is an **imperative, XML-based** language, used for specifying business process behaviour based on Web Services.
- Currently in version 2.0 part of the OASIS specifications undergoing standardisation.

# Processes and Partners

## Process

**Business Processes** can be described in two ways; *Executable* business processes model actual behaviour of a participant in a business interaction. *Abstract* business processes are partially specified processes that are not intended to be executed; they have descriptive role.

## Partners, Partner Links, Partner Types

A **Partner** represents both a *consumer* of a service provided by the business process and a *provider* of a service to the business process. **Partner Links** model *interactions* between the process and partner services and **Partner Link Types** characterises the *conversational relationship* between two services by defining specific “roles”.

# Variables & Correlation Sets











## Variables

Variables provide the means for holding messages that constitute the state of a business process.








## Correlation Set

A named group of properties that, taken together, serve to define a way of identifying an application-level conversation within a business protocol instance.

# Basic Activities

-  **receive** Do a blocking wait for a matching message to arrive
-  **reply** Send a message in reply to a formerly received message
-  **invoke** Invoke a one-way or request-responses operation
-  **assign** Update the values of variables or partner links with new data
-  **empty** No-op instruction for a business process
-  **throw** Generate a fault from inside the business process
-  **rethrow** Forward a fault from inside a fault handler
-  **exit** Immediately terminate execution of a business process instance
-  **wait** Wait for a given time period or until a certain time has passed
-  **compensate** Invoke compensation on an inner scope that has already completed

# Structured Activities

-  **flow** Contained activities are executed in parallel, partially ordered through control links
-  **sequence** Contained activities are performed sequentially in lexical order
-  **while** Contained activity is repeated while a predicate holds
-  **repeatUntil** Contained activity is repeated until a predicate holds
-  **pick** Block and wait for a suitable message to arrive (or time out)
-  **if then else** Select exactly one branch of activity from a set of choices
-  **scope** Associate contained activity with its own local variables, fault handlers, compensation handler, and event handlers

# Scopes & Handlers

## Scope

The behaviour context for each activity is provided by a **scope**. Scopes can provide local variables, links, correlation sets, subordinate activities and handlers.

## Handlers

- **Event handlers:** Allow a process to act on message events or timer events.
- **Fault handlers:** Undo partial or unsuccessful work of a scope by dealing with exceptional situations (internal faults).
- **Compensation handler:** Allow the undoing of persisted effects of already completed activities.
- **Termination handler:** Permits the dealing with forced scope termination (external faults).

# Abstract Syntax

```
 $\langle \text{bpelProcess} \rangle ::= \text{'process' name } \langle \text{parterLinkDef} \rangle? \langle \text{variableDef} \rangle? \\ \langle \text{faultHandlersDef} \rangle? \langle \text{activityDef} \rangle$   
 $\langle \text{parterLinkDef} \rangle ::= \text{'pLink' name } pLinkType$   
 $\langle \text{variableDef} \rangle ::= \text{'var' name } messageType$   
 $\langle \text{activityDef} \rangle ::= \langle \text{basicActivityDef} \rangle \\ | \langle \text{structuredActivityDef} \rangle$   
 $\langle \text{basicActivityDef} \rangle ::= \langle \text{receiveOperation} \rangle \\ | \langle \text{replyOperation} \rangle \\ | \langle \text{invokeOperation} \rangle \\ | \langle \text{assignOperation} \rangle \\ | \langle \text{emptyOperation} \rangle \\ | \langle \text{exitOperation} \rangle \\ | \langle \text{waitOperation} \rangle$ 
```

# Abstract Syntax

```
 $\langle receiveOperation \rangle ::= 'in' \langle msg \rangle$   
 $\langle replyOperation \rangle ::= 'out' \langle msg \rangle$   
 $\langle invokeOperation \rangle ::= 'out' \langle msg \rangle ('in' \langle msg \rangle)?$   
 $\langle assignOperation \rangle ::= 'assign' name value$   
 $\langle emptyOperation \rangle ::= 'empty'$   
 $\langle exitOperation \rangle ::= 'exit'$   
 $\langle waitOperation \rangle ::= 'wait' \langle timeExpr \rangle$   
 $\langle timeExpr \rangle ::= 'for' \alpha$   
                  | 'until'  $\alpha$   
 $\langle msg \rangle ::= pLink/pType/operation/msgname$ 
```

# Abstract Syntax

$$\langle structuredActivityDef \rangle ::= \langle flowOperation \rangle$$

- |  $\langle ifThenElseOperation \rangle$
- |  $\langle sequenceOperation \rangle$
- |  $\langle whileOperation \rangle$
- |  $\langle repeatOperation \rangle$
- |  $\langle pickOperation \rangle$
- |  $\langle onMessage \rangle$
- |  $\langle onAlarm \rangle$

$$\langle flowOperation \rangle ::= \text{'flow'} \langle activityDef \rangle^*$$
$$\langle ifThenElseOperation \rangle ::= \text{'if'} \beta \text{'then'} \langle activityDef \rangle (\text{'else'} \langle activityDef \rangle)^?$$
$$\langle sequenceOperation \rangle ::= \text{'seq'} \langle activityDef \rangle^*$$
$$\langle whileOperation \rangle ::= \text{'while'} \beta \langle activityDef \rangle$$
$$\langle repeatOperation \rangle ::= \text{'repeat'} \langle activityDef \rangle \text{'until'} \beta$$
$$\langle pickOperation \rangle ::= \text{'pick'} \langle onMessage \rangle^? \langle onAlarm \rangle^*$$

## Translation framework

The translation is performed through an inductively defined relation on the phrases of the abstract BPEL syntax

$$Tr : \mathbf{BPELStmt} \rightarrow \pi \mathbf{Expr}$$

Each non-trivial syntactic entity is represented as a parametrised agent of the following form

$$\llbracket S \rrbracket \stackrel{\text{def}}{=} (h, r, \tilde{g}, \tilde{p}, in, \tilde{n}).P_S \text{ where}$$

$h$  is a channel to signal process has finished

$r$  is used to return the result to some process

$\tilde{g}$  is used to get the values of variables that occur in  $S$

$\tilde{p}$  is used to put values to variables

# Trivial translations

Some BPEL constructs correspond trivially to  $\pi$ -calculus expressions.

- $Tr(msg) = pLink/pType/operation/msgname$
- $Tr('in'\langle msg \rangle) = \overline{Tr(msg)}(\tilde{x})$
- $Tr('out'\langle msg \rangle) = Tr(msg)\langle \tilde{x} \rangle$
- $Tr('empty') = \tau$
- $Tr('exit') = cancel.\emptyset$

# More complicated translations

Sequence, Flow, Pick

## A sequence of N activities

$$\llbracket \text{'seq'} S_1 S_2 \dots S_n \rrbracket \stackrel{\text{def}}{=} (h, \dots). (\text{new } h_1 \dots h_{n-1}) \\ (\llbracket S_1 \rrbracket \langle h_1, \dots \rangle \mid h_1(z_1). \llbracket S_2 \rrbracket \langle h_2, \dots \rangle \mid \dots \mid h_{n-1}(z). \llbracket S_n \rrbracket \langle h, \dots \rangle)$$

## N parallel activities

$$\llbracket \text{'flow'} S_1 S_2 \dots S_n \rrbracket \stackrel{\text{def}}{=} (h, \dots). \\ (\llbracket S_1 \rrbracket \langle h, \dots \rangle \mid \llbracket S_2 \rrbracket \langle h, \dots \rangle \mid \dots \mid \llbracket S_n \rrbracket \langle h, \dots \rangle)$$

## Selecting a single activity

$$\llbracket \text{'pick'} S_1 S_2 \dots S_n \rrbracket \stackrel{\text{def}}{=} (h, \dots). (msg_1(z_1). \llbracket S_1 \rrbracket \langle h, \dots \rangle + \\ msg_2(z_2). \llbracket S_2 \rrbracket \langle h, \dots \rangle + \dots + \tau. \llbracket S_n \rrbracket \langle h, \dots \rangle)$$

# More complicated translations

If, Switch, While

## If-then-else

$$\llbracket \text{'if'} \beta \text{'then'} S_1 \text{'else'} S_2 \rrbracket \stackrel{\text{def}}{=} (h, \dots).(\mathbf{new} t, f)\bar{b}(t, f).(t().\llbracket S_1 \rrbracket \langle h, \dots \rangle + f().\llbracket S_2 \rrbracket \langle h, \dots \rangle)$$

## Switching based on conditions

$$\llbracket \text{'switch'} S_1 S_2 \dots S_n \rrbracket \stackrel{\text{def}}{=} (h, \dots). (\tau.\llbracket S_1 \rrbracket \langle h, \dots \rangle + \tau.\llbracket S_2 \rrbracket \langle h, \dots \rangle + \dots + \tau.\llbracket S_n \rrbracket \langle h, \dots \rangle)$$

## While

$$\llbracket \text{'while'} \beta S \rrbracket \stackrel{\text{def}}{=} (h, \dots).(\mathbf{new} h_1, h_2, g)(\bar{g} !g(z).(\llbracket E \rrbracket \langle h_1, \dots \rangle | h_1(v).IF(v \rightarrow \llbracket S \rrbracket \langle h_2, \dots \rangle, \neg v \rightarrow \bar{h})|h_2(z).\bar{g}))$$

# Why all this?

All this theory has allowed us to automate the whole procedure of checking properties of real world, executing and deployed business processes. How?

- 1 We point the tool to the BPEL description for the process.
- 2 An automatic transformation takes place producing  $\pi$ -calculus specifications of the running business process. These are in a format the Mobility Workbench (MWB) can understand.
- 3 We load a file with predefined (and interesting) properties and we can model check the  $\pi$ -calculus specifications against the properties.
- 4 Feeding all this to MWB (or any model-checker) will give us a YES/NO answer!

# Concluding Remarks

In this talk

- we presented the theoretical foundations for enabling the model checking of business processes
- gave the outline of a formalisation of the BPEL language semantics in the  $\pi$ -calculus

Future Directions

- Find existing real world uses of BPEL to run the model checker on
- Finish the implementation and integration of the system with the Mobility Workbench (MWB)
- Experiment with alternative model checkers (abc, concurrency workbench)

# Acknowledgements

- My supervisor, Dr David Greaves, for his continuous support and feedback on this work,
- Professor Robin Milner for his initial suggestions and directions,
- Dr. Andy Gordon for providing valuable guidance on model checking and potential real world applications

**Thank you! Any questions?**