Design, Modelling and Analysis of a Workflow Reconfiguration

M. Mazzara[1], F. Abouzaid[2], N. Dragoni[3] and A. Bhattacharyya[1]

[1] Newcastle University, Newcastle upon Tyne, UK
{Manuel.Mazzara, Anirban.Bhattacharyya}@ncl.ac.uk
[2] École Polytechnique de Montréal, Canada
m.abouzaid@polymtl.ca
[3] Technical University of Denmark (DTU), Copenhagen
ndra@imm.dtu.dk

Abstract. This paper describes a case study involving interference between application activities and reconfiguration activities in an office workflow. We state the requirements on a system implementing the workflow and its reconfiguration, and describe the system's design in BPMN. We then use an asynchronous π-calculus and $Web\pi_\infty$ to model the design and to verify whether or not it will meet the requirements. In the process, we evaluate the formalisms for their suitability for the modelling and analysis of dynamic reconfiguration of dependable systems.

1 Introduction

Competition drives technological development, and the development of dependable systems is no exception. Thus, modern dependable systems are required to be more flexible, available and dependable than their predecessors, and dynamic reconfiguration is one way of achieving these requirements.

A significant amount of research has been performed on hardware reconfiguration (see [5] and [9]), but little has been done for reconfiguration of services, especially regarding computational models, formalisms and methods appropriate to the service domain. Furthermore, much of the current research assumes that reconfiguration can be instantaneous, or that the environment can wait during reconfiguration for a service to become available (see [14] and [13]). These assumptions are unrealistic in the service domain. For example, instantaneous mode change in a distributed system is generally not possible, because the system usually has no well-defined global state at a specific instant (due to significant communication delays). Also, waiting for the reconfiguration to complete is not acceptable if (as a result) the environment becomes dangerously unstable or the service provider loses revenue by the environment aborting the service request.

These observations lead to the conclusion that further research is required on dynamic reconfiguration of dependable services, and especially on its formal foundations, modelling and verification. In particular, the problem of interference between old configuration activities, new configuration activities and reconfiguration activities that occurs due to overlapping modes needs to be addressed. In

a preliminary paper [16], we examined a number of well-known formalisms for their suitability for reconfigurable dependable systems. In this paper, we focus on one of the formalisms ($Web\pi_\infty$) and compare it to a π-calculus in order to perform a deeper analysis than was possible in [16]. We use a more complex case study involving the reconfiguration of an office workflow for order processing, define the requirements on a system implementing the workflow and its reconfiguration, and describe the design of a system in BPMN (see section 2). We then use an asynchronous π-calculus with summation (in section 3) and $Web\pi_\infty$ [18] (in section 4) to model the design and to verify whether or not the design will meet the reconfiguration requirements. We chose process algebras because they are designed to model interaction (including interference) between concurrent activities. An asynchronous π-calculus was selected because π-calculi are designed to model link reconfiguration, and asynchrony is suitable for modelling communication in distributed systems. $Web\pi_\infty$ was selected because it is designed to model composition of web services.

Thus, the contribution of this paper is to identify strengths and weaknesses of an asynchronous π-calculus with summation and $Web\pi_\infty$ for modelling dynamic reconfiguration and verifying requirements (discussed in section 5). This evaluation may be useful to system designers intending to use formalisms to design dynamically reconfigurable systems, and also to researchers intending to design better formalisms for the design of dynamically reconfigurable systems.

2 Office Workflow: Requirements and Design

This case study describes dynamic reconfiguration of an office workflow for order processing that is commonly found in large and medium-sized organizations [7]. These workflows typically handle large numbers of orders. Furthermore, the organizational environment of a workflow can change in structure, procedures, policies and legal obligations in a manner unforseen by the original designers of the workflow. Therefore, it is necessary to support the unplanned change of these workflows. Furthermore, the state of an order in the old configuration may not correspond to any state of the order in the new configuration. These factors, taken in combination, imply that instantaneous reconfiguration of a workflow is not always possible; neither is it practical to delay or abort large numbers of orders because the workflow is being reconfigured. The only other possibility is to allow overlapping modes for the workflow during its reconfiguration.

2.1 Requirements

A given organization handles its orders from existing customers using a number of activities arranged according to the following procedure:

1. `Order Receipt`: an order for a product is received from a customer. The order includes customer identity and product identity information.

2. `Evaluation`: the product identity is used to perform an inventory check on the availability of the product. The customer identity is used to perform a credit check on the customer using an external service. If both the checks are positive, the order is accepted for processing; otherwise the order is rejected.
3. `Rejection`: if the order is rejected, a notification of rejection is sent to the customer and the workflow terminates.
4. If the order is to be processed, the following two activities are performed concurrently:
   (a) `Billing`: the customer is billed for the total cost of the goods ordered plus shipping costs.
   (b) `Shipping`: the goods are shipped to the customer.
5. `Archiving`: the order is archived for future reference.
6. `Confirmation`: a notification of successful completion of the order is sent to the customer.

In addition, for any given order, `Order Receipt` must precede `Evaluation`, which must precede `Rejection` or `Billing` and `Shipping`.

After some time, managers notice that lack of synchronisation between the Billing and Shipping activities is causing delays between the receipt of bills and the receipt of goods that are unacceptable to customers. Therefore, the managers decide to change the order processing procedure, so that `Billing` is performed before `Shipping` (instead of performing the two activities concurrently). During the transition interval from one procedure to the other, the following requirements must be met:

1. The result of the `Evaluation` activity for any given order should not be affected by the change in procedure.
2. All accepted orders must be billed and shipped exactly once, then archived, then confirmed.
3. All orders accepted after the change in procedure must be processed according to the new procedure.

2.2 Design

We designed the system implementing the office workflow using the Business Process Modeling Notation (BPMN) [4]. We chose BPMN because it is a widely used graphical tool for designing business processes. In fact, BPMN is a standard for business process modelling, and is maintained by the Object Management Group (see http://www.omg.org/).

The system is designed as a collection of eight pools: Office Workflow, Order Generator, Credit Check, Inventory Check, Reconf. Region, Bill&Ship1, Bill&Ship2 and Archive. The different pools represent different functional entities, and each pool can be implemented as a separate concurrent task (see Figure 1). Office Workflow coordinates the entire workflow: it receives a request from a customer, and makes a synchronous call to Order Generator to create an order. It then calls Credit Check (with the order) to check the creditworthiness of the customer, and tests the returned value using an Exclusive Data-Based

Gateway. If the test is positive, Office Workflow calls Inventory Check (with the order) to check the availability of the ordered item, and tests the returned value. If either of the two tests is negative, the customer is notified of the rejected order and the workflow terminates. If both tests are positive, Office Workflow calls Reconf. Region, which acts as a switch between configuration 1 and configuration 2 of the workflow, and thereby handles the reconfiguration of the workflow.

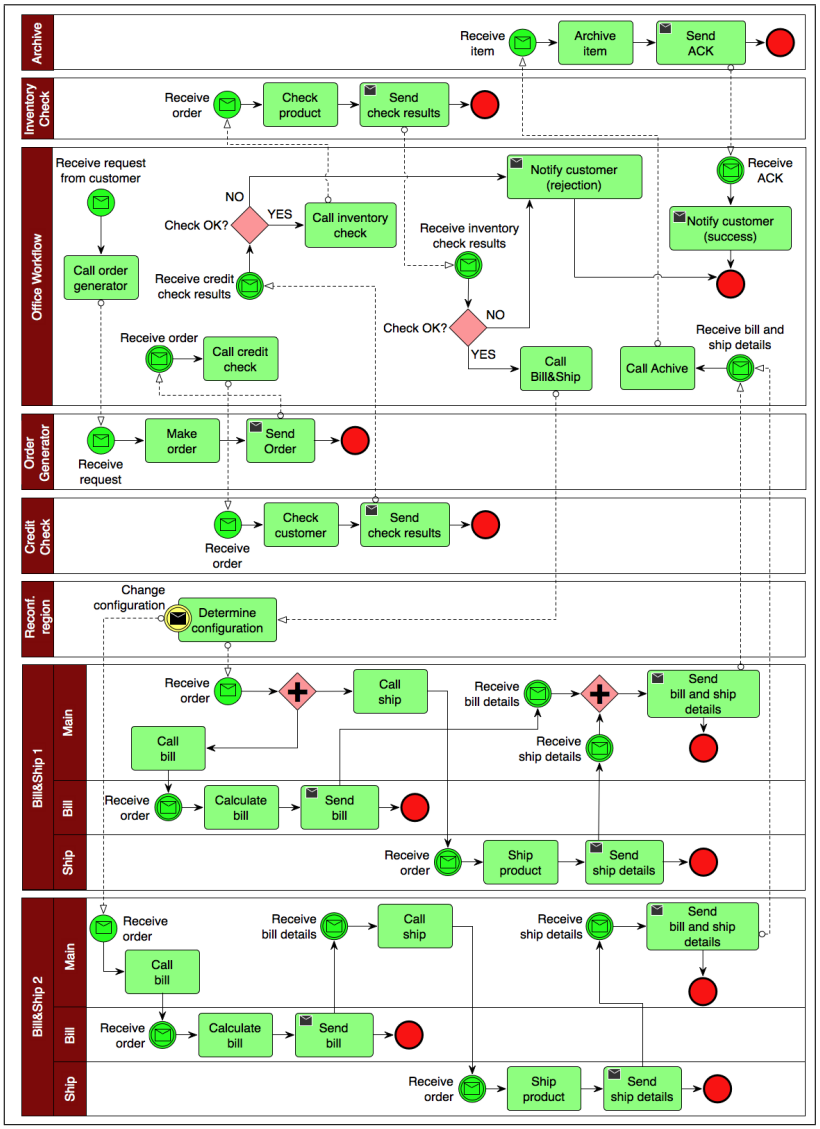

Fig. 1. Office workflow - BPMN diagram of the reconfiguration

Reconf. Region calls Bill&Ship1 by default: it makes an asynchronous call to the Main pool within Bill&Ship1, which uses a Parallel Gateway to call Bill and Ship concurrently and merge their respective results, and then returns these results to Office Workflow. The Office Workflow then calls Archive to store the order, then notifies the customer of the successful completion of the order, and then terminates the workflow. However, if Reconf. Region receives a change configuration message, it calls the Main pool within Bill&Ship2 instead, which makes sequential a call to Bill and then to Ship, and then returns the results to Office Workflow.

Notice that for the sake of simplicity, we assume neither Bill nor Ship produces a negative result. Furthermore, the Bill and Ship pools are identical in both configurations, which suggests their code is replicated (rather than shared) in the two configurations. Finally, we assume the reconfiguration is planned rather than unplanned.

3 Asynchronous π-Calculus

The asynchronous π-calculus ([10], [3]) is a subset of Milner's π-calculus [20], and it is known to be more suitable for distributed implementation. It is considered a rich paradigm for asynchronous communication, although it is not as expressive as Milner's π-calculus in representing mixed-choice constructs, such as $\overline{a}.P + b.P'$ (see [22]).

We recall the (monadic) asynchronous π-calculus. Let $\mathcal{N}$ be a set of names (e.g. $a, b, c, ...$) and $\mathcal{V}$ be a set of variables (e.g. $x, y, z, ...$). The set of the asynchronous π-calculus processes is generated by the following grammar:

$$P ::= \bar{x}z \mid G \mid P|P \mid [a = b]P \mid (\nu x)P \mid A(x_1, ..., x_n)$$

where guards G are defined as follows:

$$G ::= 0 \mid x(y).P \mid \tau.P \mid G + G$$

Intuitively, an output $\bar{x}z$ represents a message z tagged with a name x indicating that it can be received (or consumed) by an input process $x(y).P$ which behaves as $P\{z/y\}$ upon receiving z. Furthermore, $x(y).P$ binds the name y in P and the restriction $(\nu x)P$ declares a name x private to P and thus binds x. Outputs are non-blocking.

The parallel composition $P|Q$ means P and Q running in parallel. $G + G$ is the non-deterministic choice that is restricted to τ and input prefixes.

$[a = b]P$ behaves like P if a and b are identical.

$A(y_1, ..., y_n)$ is an *identifier* (also *call*, or *invocation*) of arity n. It represents the instantiation of a defined agent. We assume that every such identifier has a unique, possibly recursive, definition $A(x_1, ..., x_n) \stackrel{def}{=} P$ where the x_is are pairwise distinct, and the intuition is that $A(y_1, ..., y_n)$ behaves like P with each y_i replacing x_i.

Furthermore, for each $A(x_1, ..., x_n) \stackrel{def}{=} P$ we require: $fn(P) \subseteq \{x_1, ..., x_n\}$, where $fn(P)$ stands for the set of free names in P, and $bn(P)$ for the set of bound names in P. The *input prefix* and the ν *operator* bind the names. For example, in a process $x(y).P$, the name y is bound. In $(\nu x)P$, x is considered to be bound. Every other occurrences of a name like x in $x(y).P$ and x, y in $\bar{x}\langle y\rangle.P$ are free.

Due to lack of space we omit to give details on structural congruence and operational semantics for the asynchronous π-calculus. They can be found in [1] for the version of the calculus we use in this paper.

The Model in Asynchronous π-Calculus The model in asynchronous π-calculus needs to keep the synchronization between actions in sequence coherent with the workflow definition. So sequence is implemented by using parallel composition with prefix and postfix on the same channel. Channel names are not restricted since the full system is not described here and has to be put in parallel with the detailed implementation of the environment process described (that will be omitted here).

The entire model is expressed in asynchronous π-calculus as follows:

Entire Model

Let $params =$
$\{customer, item, Archive, ArchiveReply, Bill, BillReply, BillShip, Confirm,$
$CreditCheck, CreditOk, CreditReject, InventoryCheck,$
$InventoryOk, InventoryReject, OrderGenerator,$
$OrderGeneratorReply, OrderReceipt, Reject, Ship, ShipReply, reco, recn\}$

We can define the $Workflow$ process as follows:

$Workflow(params) \triangleq$
$(\nu\, order)\ (OrderReceipt(customer, item).\overline{OrderGenerator}\, customer, item$
$| OrderGeneratorReply(order).\overline{CreditCheck}\, customer$
$| (creditOk().\overline{InventoryCheck}\, item + CreditReject().\overline{Reject}\, order)$
$| (InventoryOk().\overline{BillShip}\ + InventoryReject().\overline{Reject}\, order)$
$| rec_o().BillShip().(\overline{Bill}\, customer, item, order \mid \overline{Ship}\, customer, item, order)$
$| BillReply(order).ShipReply(order).\overline{Archive}\, order$
$+rec_n().BillShip().(\overline{Bill}\, customer, item, order$
$| BillReply(order).\overline{Ship}\, customer, item, order)\ | ShipReply(order).\overline{Archive}\, order$
$| ArchiveReply(order).\overline{Confirm}\, order)\ | Workflow(params)$

In the model, the old region is identified as follows:

$rec_o().BillShip().(\overline{Bill}\, customer, item, order \mid \overline{Ship}\, customer, item, order)$
$| BillReply(order).ShipReply(order).\overline{Archive}\, order$

And the new region is:

$rec_n().BillShip().(\overline{Bill}\, customer, item, order$
$| BillReply(order).\overline{Ship}\, customer, item, order)\ | ShipReply(order).\overline{Archive}\, order$

In the asynchronous π-calculus, two outputs cannot be in sequence. In order to impose ordering between $\overline{Bill}$ and $\overline{Ship}$, in the new region, it is necessary to put a guard on *Ship*, which requires enlarging the boundary of the old region to include the processes in the environment of the workflow that synchronize with $\overline{Bill}$ and $\overline{Ship}$. We did not model these processes because they are outside the system being designed, but the limitations of the asynchronous π-calculus imply that we must be able to access the logic of external services for which we know only the interfaces. For a more detailed description of this problem, please see [12].

The entire model represents a specific instance of the workflow that spawn concurrently another instance with fresh customer and item which here are assumed to be fresh names but in reality will be user entered (but it is not relevant to our purposes). We have to assume the existence of a "higher level" process (at the level of the BPEL engine) that activates the entire workflow and bounds the names that are free in the above π-calculus process. In this model channels $creditOK, creditReject, InventoryOK$ and $InventoryReject$ are used to receive the result of the credit check and inventory check, respectively. The old/new region is externally triggered using specific channels rec_o and rec_n chosen according to the value x received on channel *region*:

$$(\nu\, x)Workflow(param) \;\mid\; region(x).([x = new]\overline{rec_n} \;\mid\; [x = old]\overline{rec_o}\,)$$

In section 4 we show a more efficient solution using Webπ_∞.

Analysis in π-logic Logics have long been used to reason about complex systems, because they provide abstract specifications that can be used to describe system properties of concurrent and distributed systems. Verification frameworks can support checking of functional properties of such systems by abstracting away from the computational contexts in which they are operating.

In the context of π-calculi, one can use the π-logic with the HAL Toolkit model-checker [8]. The π-logic has been introduced in [8] to specify the behavior of systems in a formal and unambiguous manner by expressing temporal properties of π-processes.

Syntax of the π-logic The logic integrates modalities defined by Milner ([21]) with $EF\phi$ and $EF\{\chi\}\phi$ modalities on possible future. The π-logic syntax is:

$$\phi ::= true \mid \sim\phi \mid \phi \wedge \phi' \mid EX\{\mu\}\phi \mid EF\phi \mid EF\{\chi\}\phi$$

where μ is a π-calculus action and χ could be μ, $\sim\mu$, or $\bigvee_{i\in I}\mu_i$ and where I is a finite set.

Semantics of π-formulae is given below:

- $P \models true$ for any process P;
- $P \models \sim\phi$ iff $P \not\models \phi$;
- $P \models \phi \wedge \phi'$ iff $P \models \phi$ and $P \models \phi'$;

- $P \models EX\{\mu\}\phi$ iff there exists P' such as $P \xrightarrow{\mu} P'$ and $P' \models \phi$ (**strong next**);
- $P \models EF\phi$ iff there exists $P_0, ..., P_n$ and $\mu_1, ..., \mu_n$, with $n \geq 0$, such as $P = P_0 \xrightarrow{\mu_1} P_1 ... \xrightarrow{\mu_n} P_n$ and $P_n \models \phi$. The meaning of $EF\phi$ is that ϕ must be true sometimes in a possible future.
- $P \models EF\{\chi\}\phi$ if and only if there exists $P_0, ..., P_n$ and $\nu_1, ..., \nu_n$, with $n \geq 0$, such that $P = P_0 \xrightarrow{\nu_1} P_1 ... \xrightarrow{\nu_n} P_n$ and $P_n \models \phi$ with:
  - $\chi = \mu$ for all $1 \leq j \leq n,\ \nu_j = \mu$ *or* $\nu_j = \tau$;
  - $\chi = \sim \mu$ for all $1 \leq j \leq n,\ \nu_j \neq \mu$ *or* $\nu_j = \tau$;
  - $\chi = \bigvee_{i \in I} \mu_i$: for all $1 \leq j \leq n, \nu_j = \mu_i$ for some $i \in I$ *or* $\nu_j = \tau$.

  The meaning of $EF\{\chi\}\phi$ is that the truth of ϕ must be preceded by the occurrence of a sequence of actions χ.

Some useful dual operators are defined as usual:
$false$, $\phi \vee \phi$, $AX\{\mu\}\phi$ ($\sim EX\{\mu\} \sim \phi$), $< \mu > \phi$ (weak next), $[\mu]\phi$ (Dual of weak next), $AG\phi$ ($AG\{\chi\}$) (always).

Properties of the dynamic reconfiguration model
We need to verify that during the reconfiguration interval the requirements given in section 2.1 hold. For this purpose, we need to express the requirements formally, if possible, using the π-logic.

The result of the `Evaluation` activity for any given order should not be affected by the change in procedure. The following formula means whatever the chosen path (old or new region), an order will be billed, shipped and archived or refused:

$AG\{EF\{OrderReceipt()\}true\}$
$AG\{(EF\{\overline{Bill}\, customer, item, order\}true \wedge EF\{\overline{Ship}\, customer, item, order\}true \wedge$
$EF\{\overline{Archive}\, order\}true) \vee EF\{\overline{Reject}\,\}true\}$

All accepted orders must be billed and shipped exactly once, then archived, then confirmed. The following formula means that after an order is billed and shipped, it is archived and confirmed, and cannot be billed nor shipped again:

$AG\{EF\{BillShip()\}true\}$
$AG\{EF\{\overline{Bill}\, customer, item, order\}true \wedge EF\{\overline{Ship}\, customer, item, order\}true \wedge$
$EF\{\overline{Archive}\, order\}true\} \wedge EF\{\overline{Confirm}\, order\}true\}$
$AG\{\{\overline{Bill}\, customer, item, order\}false \wedge \{\overline{Ship}\, customer, item, order\}false\}$

All orders accepted after the change in procedure must be processed according to the new procedure We can express in the π-logic the following requirement: "after a reception on the channel rec_n, no other reception on channel rec_0 will be accepted". This meets the desired requirement since it is obvious from the model that, if a signal is received on channel rec_n, the order will be processed according to the new procedure.

$AG\{\{rec_n()\}true\ AG\{rec_0()\}false\}$

However, since the choice between the old procedure and the new one is non-deterministic, this formula will not be true, although it is an essential requirement for the model. This result illustrates the difficulty of the asynchronous π-calculus to model the dynamic reconfiguration properly. A first attempt to answer this problem is presented in the next section.

4 Webπ_∞

Webπ_∞ is a conservative extension of the π-calculus developed for modelling and analysis of Web services and Service Oriented Architectures. The basic theory has been developed in [18] and [15], whilst its applicability has been shown in other work: [12] gives a BPEL semantics in term of Webπ_∞, [6] clarifies some aspects of the Recovery Framework of BPEL, and [17] exploits a web transaction case study (a toy example has also been discussed in [16]).

Syntax and Semantics The syntax of `web`π_∞ *processes* relies on a countable set of *names*, ranged over by $x, y, z, u, \cdots$. Tuples of names are written $\widetilde{u}$. We intend $i \in I$ with I a finite non-empty set of indexes.

$$P ::= 0 \mid \overline{x}\,\widetilde{u} \mid \sum_{i \in I} x_i(\widetilde{u_i}).P_i \mid (x)P \mid P|P \mid\ !x(\widetilde{u}).P \mid \langle\!|P\ ;\ P|\!\rangle_x$$

It is worth noting that the syntax of `web`π_∞ simply augments the asynchronous π-calculus with a workunit process. A workunit $\langle\!|P\ ;\ Q|\!\rangle_x$ behaves as the *body* P until an abort $\overline{x}$ is received, and then it behaves as the *event handler* Q.

We give the semantics of `web`π_∞ in two steps, following the approach of Milner [19], separating the laws that govern the static relations between processes from the laws that rule their interactions. The static relations between processes are governed by the *structural congruence* $\equiv$, the least congruence satisfying the Abelian monoid laws for parallel and summation (associativity, commutativity and $\mathbf{0}$ as identity) and closed with respect to α-renaming and the axioms shown in table 1.

The scope laws are standard while novelties regard workunit and floating laws. The law $\langle\!|\mathbf{0}\ ;\ Q|\!\rangle_x \equiv \mathbf{0}$ defines committed workunit, namely workunit with $\mathbf{0}$ as body. These ones, being committed, are equivalent to $\mathbf{0}$ and, therefore, cannot fail anymore. The law $\langle\!|\langle\!|P\ ;\ Q|\!\rangle_y \,|\, R\ ;\ R'|\!\rangle_x \equiv \langle\!|P\ ;\ Q|\!\rangle_y \,|\, \langle\!|R\ ;\ R'|\!\rangle_x$ moves

Scope laws	$(u)\mathbf{0} \equiv \mathbf{0}, \qquad (u)(v)P \equiv (v)(u)P$
	$P \mid (u)Q \equiv (u)(P \mid Q)\ , \quad \mathit{if}\ u \notin \mathrm{fn}(P)$
	$\langle\!(z)P\ ;\ Q\rangle\!\rangle_x \equiv (z)\langle\!P\ ;\ Q\rangle\!\rangle_x\ , \quad \mathit{if}\ z \notin \{x\} \cup \mathrm{fn}(Q)$
Workunit laws	$\langle\!\mathbf{0}\ ;\ Q\rangle\!\rangle_x \equiv \mathbf{0}$
	$\langle\!\langle P\ ;\ Q\rangle\!\rangle_y \mid R\ ;\ R'\rangle\!\rangle_x \equiv \langle\!P\ ;\ Q\rangle\!\rangle_y \mid \langle\!R\ ;\ R'\rangle\!\rangle_x$
	$\langle\!(z)P\ ;\ Q\rangle\!\rangle_x \equiv (z)\langle\!P\ ;\ Q\rangle\!\rangle_x\ , \quad \mathit{if}\ z \notin \{x\} \cup \mathrm{fn}(Q)$
Floating law	$\langle\!\overline{z}\,\widetilde{u} \mid P\ ;\ Q\rangle\!\rangle_x \equiv \overline{z}\,\widetilde{u} \mid \langle\!P\ ;\ Q\rangle\!\rangle_x$

Table 1. $\mathtt{web}\pi_\infty$ Structural Congruence

workunit outside parents, thus flattening the nesting. Notwithstanding this flattening, parent workunits may still affect the children by means of names. The law $\langle\!\overline{z}\,\widetilde{u} \mid P\ ;\ Q\rangle\!\rangle_x \equiv \overline{z}\,\widetilde{u} \mid \langle\!P\ ;\ Q\rangle\!\rangle_x$ floats messages outside workunit boundaries. By this law, messages are particles that independently move towards their inputs. The intended semantics is the following: if a process emits a message, this message traverses the surrounding workunit boundaries until it reaches the corresponding input. In case an outer workunit fails, recoveries for this message may be detailed inside the handler processes.

The dynamic behavior of processes is instead defined by the reduction relation $\rightarrow$ which is the least relation satisfying the axioms and rules shown in table 2 and closed with respect to $\equiv$, $(x)\_$, $\_ \mid \_$, and $\langle\!\_\ ;\ Q\rangle\!\rangle_z$. In the table we use the shortcut: $\langle\!P\ ;\ Q\rangle\!\rangle \stackrel{\text{def}}{=} (z)\langle\!P\ ;\ Q\rangle\!\rangle_z$ where $z \notin \mathrm{fn}(P) \cup \mathrm{fn}(Q)$

COM	$\overline{x_i}\,\widetilde{v} \mid \sum_{i\in I} x_i(\widetilde{u_i}).P_i \ \rightarrow\ P_i\{\widetilde{v}/\widetilde{u_i}\}$
REP	$\overline{x}\,\widetilde{v} \mid !x(\widetilde{u}).P \ \rightarrow\ P\{\widetilde{v}/\widetilde{u}\} \mid !x(\widetilde{u}).P$
FAIL	$\overline{x} \mid \langle\!\prod_{i\in I}\sum_{s\in S} x_{is}(\widetilde{u_{is}}).P_{is} \mid \prod_{j\in J}!x_j(\widetilde{u_j}).P_j\ ;\ Q\rangle\!\rangle_x \ \rightarrow\ \langle\!Q\ ;\ \mathbf{0}\rangle\!\rangle$
	$\mathit{where}\ J \neq \emptyset \vee (I \neq \emptyset \wedge S \neq \emptyset)$

Table 2. $\mathtt{web}\pi_\infty$ Reduction Semantics

Rules (COM) and (REP) are standard in process calculi and model input-output interaction and lazy replication. Rule (FAIL) models workunit failures: when a unit abort (a message on a unit name) is emitted, the corresponding body is terminated and the handler activated. On the contrary, aborts are not possible if the transaction is already terminated (namely every thread in the body has completed its own work), for this reason we close the workunit restricting its name.

The model in Webπ_∞ For the modelling purposes of this work, the idea of workunit and event handler turn out to be particularly useful. Webπ_∞ uses

the mechanism of workunit to bound the identified regions, and event raising is exploited to operate the non immediate change (reconfiguration). The model can be expressed as follows (as a shortcut we will use here process invocation):

$$
\begin{array}{l}
Workflow(customer, item) \triangleq \\
(\nu\, order)\ OrderReceipt(customer, item).\overline{OrderGenerator}\, customer, item \\
\mid OrderGeneratorReply(order).\overline{CreditCheck}\, customer \\
\mid (CreditCheckReply_t(order).\overline{InventoryCheck}\, item \\
+ CreditCheckReply_f(order).\overline{Reject}\, order) \\
\mid (InventoryCheckReply_t(order).\overline{BillShip} \\
+ InventoryCheckReply_f(order).\overline{Reject}\, order) \\
\mid \langle\!| BillShip().(\overline{Bill}\, customer, item, order \mid \overline{Ship}\, customer, item, order \\
\mid (\nu\, customer)(\nu\, item)\, Workflow(customer, item)) \\
;\ (\nu\, customer)(\nu\, item)\, Workflow_n(customer, item) |\!\rangle_{rec} \\
\mid BillReply(order).ShipReply(order).\overline{Archive}\, order \\
\mid ArchiveReply(order).\overline{Confirm}\, order
\end{array}
$$

Webπ_∞ shows here a subtle feature which is important for modelling reconfigurable systems. Since the floating laws of structural congruence allow the asynchronous outputs in a workunit to freely escape, once the region to reconfigure has been entered and the $\overline{BillShip}$ has been triggered, $\overline{Bill}\, customer, item, order$ and $\overline{Ship}\, customer, item, order$ will not be killed by any incoming rec signal. This means that, once the region has been entered by an order, that order will go through without being interrupted by reconfiguration events and the old order will be processed according to the old procedure, not the new one. Future orders will find instead only the new procedure $Workflow_n$ waiting for orders:

$$
\begin{array}{l}
Workflow_n(customer, item) \triangleq \\
(\nu\, order)\ OrderReceipt(customer, item).\overline{OrderGenerator}\, customer, item \\
\mid OrderGeneratorReply(order).\overline{CreditCheck}\, customer \\
\mid (CreditCheckReply_t(order).\overline{InventoryCheck}\, item + \\
CreditCheckReply_f(order).\overline{Reject}\, order) \\
\mid (InventoryCheckReply_t(order).\overline{BillShip}\ + \\
InventoryCheckReply_f(order).\overline{Reject}\, order) \\
\mid BillShip().(\overline{Bill}\, customer, item, order \mid BillReply(order).\overline{Ship}\, customer, item, order) \\
\mid ShipReply(order).\overline{Archive}\, order\ \mid ArchiveReply(order).\overline{Confirm}\, order \\
\mid (\nu\, customer)(\nu\, item)\, Workflow_n(customer, item)
\end{array}
$$

As in the π-calculus model, we have to assume the existence of a top level process activating the entire workflow and bounding all the names appearing free in the above π-calculus process. The change in procedure will be activated when the channel t is triggered.

$$(\nu\, customer)(\nu\, item)(\nu\, rec)\, Workflow(customer, item) \mid t().\overline{rec}$$

This process is also responsible for triggering the reconfiguration.

Analysis in Webπ_∞ Analysis in Webπ_∞ is intended as equational reasoning. At the moment, one severe weakness of Webπ_∞ is its lack of tool support, i.e. au-

tomatic system verification. However, it is clearly possible to encode Webπ_∞ into the π-calculus, being the only technical complication the encoding of the workunit and its asynchronous interrupt. Once the compilation into the π-calculus has been done, we can proceed using HAL. From one side, Webπ_∞ simplifies the modelling of dependable systems expressing with its workunit the recovery behavior. On the other side, it makes the verification more difficult. Luckily, there is an optimal solution using Webπ_∞ as *modelling language* and the π-calculus as *intermediate language*, i.e. a *verification bytecode*. We can then offer a practical modelling suite to the designer and still use the tool support for the π-calculus. At the moment our research has not gone so far, so we will just discuss the three requirements here. We will analyse the requirements in terms of equational reasoning (see [18] and [15]). The case study of this paper is interesting at showing both the modelling power of Webπ_∞ and the weaknesses of its reasoning system.

The result of the `Evaluation` activity for any given order should not be affected by the change in procedure. The acceptability of an order (`Evaluation` activity) is computed *outside* the region to be reconfigured, and there is no interaction between `Evaluation` and the region. That means that the `Evaluation` in the old procedure $workflow$ is exactly the same as in the new procedure $workflow_n$, i.e. the checks are performed in the same exact order. We can formally express it, in term of equational reasoning, stating that the `Evaluation` activity in the old procedure $workflow$ is bisimilar to the `Evaluation` activity in the new procedure $workflow_n$ which is trivially true.

All accepted orders must be billed and shipped exactly once, then archived, then confirmed. The presence of a workunit does not affect how the order itself is processed. The workflow of actions described by the requirement can be formally expressed as follows:

$(\nu\, x)(\nu\, y)\ (\overline{Bill}\, customer, item, order \mid \overline{Ship}\, customer, item, order$
$\mid BillReply(order).\overline{x} \mid ShipReply(order).\overline{y}\ \mid x().y().\overline{Archive}\, order$
$\mid ArchiveReply(order).\overline{Confirm}\, order)$

In plain words this process describes billing and shipping happening in any order but both before archiving and confirming. The channels x and y are there precisely to work as a joint for billing and shipping. If we want to express the requirements in term of equational reasoning, we can require that both the old and the new regions have to be bisimilar with the above process. However, this is too strict since the above process allows a set of traces which is a superset of both the set of traces of the old configuration and the new one. In this case similarity could be considered instead of bisimilarity.

All orders accepted after the change in procedure must be processed according to the new procedure To show this requirements has been implemented in the model semantic reasoning is not necessary, structural congruence

is sufficient. The change in procedure is here modelled by triggering the rec channel and spawning the workunit handler. The handler then activates a new instance of the workflow based on the new procedure scheme which has been called $workflow_n$. The floating laws of structural congruence of Webπ_∞ (definition 1) allow the asynchronous outputs in a workunit to freely escape the workunit itself. Thus, once the region to reconfigure has been already entered and the $\overline{BillShip}$ has been triggered, $\overline{Bill}\,customer, item, order$ and $\overline{Ship}\,customer, item, order$ will not be killed by any incoming rec signal. Thus, once the region has been entered by an order, that order will be not interrupted by reconfiguration events so that old order will be processed according to the old procedure and not the new one.

5 Discussion

In this section, we discuss three issues which arose during design and modelling: how the modelling influenced our design, how the π-calculus and Webπ_∞ compare with respect to modelling, and correctness criteria for verification of the workflow reconfiguration.

Modelling and Design Different formalisms have different biases on design because of their different perspectives. In one of the alternative designs we considered, the Bill and Ship pools were outside the reconfiguration region, so that their code was shared between the two configurations. Thus, the boundary of the reconfiguration region was different. We chose the design in section 2.2 because it is easier to model. It is the job of a formalist to model what the system designers produce, and ask them to change the design if it cannot be modelled or is unverifiable. Our experience with asynchronous π-calculi and Webπ_∞ suggested that extending the boundary of the reconfiguration region to include billing and shipping was a practical choice. This is because in the asynchronous π-calculus (and consequently in Webπ_∞), two outputs cannot be in sequence. So, in order to impose ordering between `Bill` and `Ship`, we had to enlarge the boundary of the reconfiguration region to include the processes in the environment of the workflow that synchronize with them. The negative side of this solution is that we have been forced to include in the region parts of the system that were not intended to be changed. Here the asynchronous π-calculus shows its weakness in terms of reconfiguring processes dynamically.

Comparison of π-calculus and Webπ_∞ This paper has shown the Webπ_∞ workunit as being able to offer a more efficient solution to the problem of modelling the case study. By means of the Webπ_∞ floating laws, interference between application activities and reconfiguration activities can also be better handled. However, at the moment, one weakness of Webπ_∞ is its lack of tool support, whereas the π-calculus is supported by verification tools (e.g. TyPiCal [11] and HAL [8]). Therefore, Webπ_∞ has to be intended as a a front end for modelling

with the the π-calculus as the *verification bytecode*. As mentioned above, neither the asynchronous π-calculus nor Webπ_∞ can have two outputs in sequence, and this leads to the specific design choice.

Correctness Criteria The standard notion of correctness used in process algebras is congruence based on bisimulation. However, our requirements are not all expressible as congruences between processes. The first and third requirements can be expressed as congruences, and so bisimulation can be used in the reasoning. The second requirement cannot be expressed as a congruence because the old and new configurations are not behaviourally congruent. So, we have used reasoning based on simulation instead. Thus, we found that congruence as it has been used in section 4 is not always applicable for verifying the correctness of our models. Therefore, in section 3 we have investigated model checking.

The discussion leads us to the following:

1. It is easier to model workflow reconfiguration in Webπ_∞ than in the asynchronous π-calculus. However, modelling would be even easier in a synchronous version of Webπ_∞.
2. Model checking is more widely applicable than equational reasoning based on congruences for verifying workflow reconfiguration.

These two conclusions seem to have wider applicability than just reconfiguration of workflows; but this needs to be verified.

Future Work We intend to proceed with a deeper analysis of alternative designs for this case study, and evaluate other formalisms, such as VDM [2] and Petri nets [23]. We are also working on a BPEL implementation of the system. We also need larger industrial case studies to help us to design and evaluate formalisms for the modelling and analysis of dynamic reconfiguration.

Acknowledgments

This work is partly funded by the EPSRC under the terms of a graduate studentship. The paper has been improved by conversations with John Fitzgerald, Cliff Jones, Alexander Romanovsky, Jeremy Bryans, Gudmund Grov, Mario Bravetti, Massimo Strano, Michele Mazzucco, Paolo Missier and Mu Zhou. We also want to thank members of the Reconfiguration Interest Group (in particular, Kamarul Abdul Basit, Carl Gamble and Richard Payne), the Dependability Group (at Newcastle University) and the EU FP7 DEPLOY Project (Industrial deployment of system engineering methods providing high dependability and productivity).

References

1. R. M. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous π-calculus. *Theoretical Computer Science*, 195(2):291 – 324, 1998.

2. D. Bjorner and C. B. Jones, editors. *The Vienna Development Method: The Meta-Language*, volume 61 of *Lecture Notes in Computer Science*. Springer, 1978.
3. G. Boudol. Asynchrony and the π-calculus. rapport de recherche 1702. Technical report, INRIA, Sophia-Antipolis, 1992.
4. BPMN. Bpmn - business process modeling notation. 'http://www.bpmn.org/.
5. A. Carter. Using dynamically reconfigurable hardware in real-time communications systems: Literature survey. Technical report, Computer Laboratory, University of Cambridge, November 2001.
6. N. Dragoni and M. Mazzara. A formal semantics for the ws-bpel recovery framework - the pi-calculus way. In *WS-FM'09, Springer Verlag*, 2009.
7. C. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In *Proceedings of the Conference on Organizational Computing Systems (COOCS 1995)*. ACM, 1995.
8. G. L. Ferrari, S. Gnesi, U. Montanari, and M. Pistore. A model-checking verification environment for mobile processes. *ACM Transactions on Software Engineering and Methodology*, 12(4):440–473, 2003.
9. P. Garcia, K. Compton, M. Schulte, E. Blem, and W. Fu. An overview of reconfigurable hardware in embedded systems. *EURASIP J. Embedded Syst.*, 2006, January 2006.
10. K. Honda and M. Tokoro. An object calculus for asynchronous communication. In P. America, editor, *European Conference on Object-Oriented Programming (ECOOP)*, page 133147. Lecture Notes in Computer Science 512, 1991.
11. N. Kobayashi. Typical: Type-based static analyzer for the pi-calculus. http://www.kb.ecei.tohoku.ac.jp/ koba/typical/.
12. R. Lucchi and M. Mazzara. A pi-calculus based semantics for ws-bpel. *Journal of Logic and Algebraic Programming*, 70(1):96–118, 2007.
13. J. Magee, N. Dulay, and J. Kramer. Structuring parallel and distributed programs. *Software Engineering Journal (Special Issue)*, 8(2):73–82, 1993.
14. J. Magee, J. Kramer, and M. Sloman. Constructing distributed systems in conic. *IEEE Transactions on Software Engineering*, 15(6):663–675, 1989.
15. M. Mazzara. *Towards Abstractions for Web Services Composition*. PhD thesis, Department of Computer Science, University of Bologna, 2006.
16. M. Mazzara and A. Bhattacharyya. On modelling and analysis of dynamic reconfiguration of dependable real-time systems. In *DEPEND, International Conference on Dependability*, 2010.
17. M. Mazzara and S. Govoni. A case study of web services orchestration. In *COORDINATION*, pages 1–16, 2005.
18. M. Mazzara and I. Lanese. Towards a unifying theory for web services composition. In *WS-FM*, pages 257–272, 2006.
19. R. Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(2):119–141, 1992.
20. R. Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, 1999.
21. R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. *Theoretical Computer Science*, 1993.
22. C. Palamidessi. Comparing the expressive power of the synchronous and the asynchronous pi-calculus. In *Mathematical Structures in Computer Science*, pages 256–265. ACM, 1997.
23. C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Fakultt Matematik und Physik, Technische Universitaet Darmstadt, 1962.