Sindh Univ. Res. Jour. (Sci. Ser.) Vol: 40 (2) 109-118 (2008)

Sindh Univ. Res. Jour.

SURJ

πBPEL SERVICES COLLABORATION

A. G. Memon, and M. S. Chandio

Institute of Mathematics and Computer Science, University of Sindh, Jamshoro, Pakistan
ghafoor@usindh.edu.pk, saleem@usindh.edu.pk

(Received 07th February 2009 and Revised 14th February 2009)

Abstract

Today collaboration services technologies for interoperability; dynamic discovery and collaboration of complex service have become popular. This may be used in different mechanism such as P2P, Grid-to-Grid, network-to-network and Internet. Complex services ensure the ability exchange of different applications running on heterogeneous platforms over the Internet. Nowadays these types of services are gaining attention in the world. Industry standards for Web Service composition, such as πBPEL technology, provide the notation and additional control mechanisms for the execution of business processes in Web service collaborations. The function of πBPEL technology consists of locations, containing dynamic documents and π-calculus processes which communicate with each other. Here, we have used process equivalence to analyze various communication patterns used in different query systems by servers, and suggest a novel model connecting mobility. The πBPEL, which is a simple and conventional addition of the π-calculus, we introduce it as a theoretical and foundational model for collaborative services management. We proposed πBPEL model and implemented in formalization of location and relocation with cloning framework for multiple web collaborative services. It validates a range of access types to resources with mixture of relocation, similar execution, and πBPEL declaration. Our model simplifies some semantical and pragmatical aspects, in particular regarding temporization, gaining a better understanding of the basic issues.

Keywords: πBPEL, π-calculus, Services-oriented, Collaboration Services.

1. Introduction and related work

The rise of web service has spurred automation of cooperation among organizations. Organizations process interoperability is an important topic for business currently. Loosely coupled organizations services are gaining importance in many business domains. The distinct characteristics of inter-organizational process, such as heterogeneity and autonomy of the participating applications have lead to the development of several new agent-based technologies supporting inter-organizational cooperation. The organizations shall contribute with their core competence and thereby generate a "best-of-breed" construct in respect of required expertise. The propagated benefits are higher flexibility and quicker provision of customized services. Besides, bespoke customer solutions, dissolution of conventional organizational structures by concentrating on core competences, and the rapid development of technologies to collaborate and communicate between remote locations are further trends that render, this concept conceivable and promising. It also speaks of different languages for exemplifying a process that collaboration open communications and composes a new one. The aim of collaboration communication is to ease and to automate business process collaborations across organizations boundaries. In paper (Memon 2005), we presented encoding and translation for organization-to-organization communication for share services which is based upon the π-calculus. Now this paper is extensive the work which based upon BPEL. It defines information of control and data flow among jobs, such as invocation to a communication mechanism.

The main reason here for using the π-calculus for formalization is collaboration services composition language, like XLANG, BPEL and WS-CDL claim to be based on it, and they should therefore allow rigorous mathematical treatment. However, no interesting relation with process algebras has really been proved for any of neither them, nor an effective tool for analysis and reasoning, either theoretical or software based, has been released. Therefore, we see a gap that needs

CamScanner

to be filled and we want to address the problem of composing services starting directly from the π-calculus. In this paper, we mainly refer to BPEL, the most likely candidate to become a standard among workflow-based composition languages. Other languages have been introduced, among them WS-CDL (Kavantzas, 98) which claims to be in some relation with the fusion calculus (Parrow, 98). Researchers discuss formal semantics of compensable activities in this context. The calculus in (Butler, 2004) is inspired by BPBeans (Chessel, 2001) the πt-calculus (Bocchi, 2003) focuses on BizTalk; (Bruni, 2003) deals with short-lived transactions in BizTalk; (Bruni, 2005) also presents the formal semantics for a hierarchy of transactional calculi with increasing expressiveness. This paper focuses on formal definition which framework based on Mobile Ambients (Bugliesi, 2004, Cardelli, 2000) which is similar to π-calculus (Parrow, 2001). We studied the interoperability of Client/Server and explored the approach to supporting high interoperability of Client/Server interactions. We proposed πBPEL model and implemented in formalization of migration and cloning framework for composite web collaboration services (Yin, 2000). A πBPEL is a protocol that has the ability to nomadic from one position to different positions (Memon 2005, Ishikawa, 2004). It validates a range of access types to resources with combination of relocation, similar execution, and

- 0 is inaction; it is a process that can do nothing.
- | denotes parallel composition,
- in the composition $P \mid Q$, the two components can proceed independently and interact via shared names or processes.
- $(vb)P$ is called the restriction and means that the scope of name b is restricted to P.
- $!P$ *replicated input on channel*

πBPEL announcement. This paper aims to make process calculus that is well suited express the conventions, uniform organizational resources and other shared idiosyncrasies that hold large organizational communication and collaborations methods together.

2. π-calculus

The π-calculus (Yin, 2005) is built around two classes of identifiers, which are *Channels* and *Levels*. *Channel names* (Milner, 97) are drawn from infinite supply. A specification language must be powerful enough to express the mapping of a simulation model to any protocol on any target agent architecture. The π-calculus expresses the move of data but also of code. This offers a solid base to make use of the *Agent* generation. The πBPEL paradigm is a construct where mobility is achieved by allowing agents to be passed as values in a communication. This model of concurrent computation based upon the notion of naming. The most primitive entity in π - calculus is a name which refers to a link or a channel. Semantics is done in terms of a reduction system and a version of labeled transitions called commitment. In the πBPEL not only names, but also agents of arbitrarily high-order, can be transmitted. The syntax of the πBPEL is an extension of the syntax of the π-calculus. Let be a, b, ...and x, y,..... a set of names and P, Q,..... a set of processes.

- ϵ *null*
- $(x).P$ *New input action*
- $a[P]$ *ambient*
- *in M* *Enter into M*
- $M \stackrel{def}{=}$ *Masseges Loc*
- *out M* *Exit of M*
- $M.M'$ *Loc*
- *open M* *open M*
- $M.P$ *Capability actions*
- $\langle M \rangle$ *asynchronous output action*

The following grammars define the languages of processes in between P2P, Grid-to-Grid, network-to-network and Internet.

Channels:

Host *Ch* ::= | ( ) @host
Net *Ch* ::= | ( ) @host | @Net

Names:

Meta-Variable Levels *host, net,.....* ∈ *Level*

Net, places and agents

$Net \stackrel{def}{=} Customer \mid Client \mid Server_i \mid Server(Required_server)$

The *Net* consists of three places. *Client* is a place where *customer* for enquiring information of particular goods will resides. $Server_i$ is different organizations where it hold an agent that gives a visiting agent an advice for shops selling a particular goods and *Server* is required shop where product available.

3. Formal Model of πBPEL Process within BPEL

3.1 WS-BPEL

BPEL is, in practice, a layer on top of WSDL. The information contained in WSDL documents defines message types and port types which represent the operations supported by the service and the interaction modalities. This information is then used by BPEL for specifying the flow of actions to perform. A BPEL document is an XML-based document that can be executed by an orchestration engine which is the central coordinator. The engine will read the BPEL document and will invoke the necessary collaboration services in the specified order. The process itself will be offered as a collaboration service and can be invoked in the same way.

<process name = "The Process">
- . *<partnerLinks>* for listing the external web service invoked from within the workflow
- . *<variable>*, *<assign>*, *<copy>* and XPath queries for data storage and manipulation
- . *<corelarionSets>* for specifying binding for a set of operations to a service instance
- . *<invoke> and <invoke>* ..... *< receive>* for consuming synchronous and asynchronous web services
- . *<scope> and <faultHandlers>* for execution management
- . *<flow> and <sequence>* for parallel execution paths
- . *<switch>* for conditional branching
- . *</process>*

To make more intelligible the syntax, elements will be expressed by using BNF formalisms instead of XML-Schema (in the same style of the BPEL specification).

3.2 The πBPEL Technology Framework

This section outlines the main features of the πBPEL, a distributed model of computing for large-scale open networks. We briefly introduce πBPEL, discuss the objectives of the work and present the abstractions of the framework, which we exploit for reporting data travelers' attitudes. In Section 2 shows the syntax and basic communication of πBPEL. Common reductions and notions of concurrent process algebrian. Name and boundary are used earlier work (Memon, 2005).

πBPEL can be generally described as resource access control with hierarchical locations mobility. Our aim is not to provide distributed high level-programming model but only we explore hiding localities in distributed programming and πBPEL enables representation of synchronization/asynchronization programming constructs. Another view of πBPEL is as advanced distributed services, which is implementing in higher-level languages. πBPEL is thus expressive enough to define the attitudes of agent as processes P,Q. It can be expressed as ambient host and ambient agent like (*host[agent[P|Q| ...]...]*). In the light of πBPEL task takes place for the lowest common denominator among various database organizations applications. Specially complicated services that require higher degrees of lucidity can be built-on. Moreover it is assumed that goings-on internal communication remote messaging with other external communication. In other terminology may any interactions occur outside but not inside of hosts. This logically implies that observable action of each host or agent do not change even manifold hosts or agents actions are distributed. Such collaborative services are needed distributed Memory management, location independently, secure messaging and channels with guaranteed quality of service. So πBPEL can provide a model of mobility, which subsumes message passing, remote evaluation as well as process migration, along with models user mobility.

3.2.1 The πBPEL Interaction

We first characterize interaction actions for remote that provides a service as activity process. Name-value procedure handles information the form of identifiers by BPEL. πBPEL using the terminology of records [Luca Cardelli, 1998] directly in Name-values. As $rec\, r(l_1 = v_1,...,l_n = v_n)$ is received record that has consists of pairs of variables labels *l*, *with* value v_i, $get\, r\, l(x).P$ obtain a data of the variable *l* in the

record and continues as P, and at last $setv(x)P$ sets data of the variable l in the record and continues as P. We denote $l_1 = v_1, \ldots, l_n = v_n$ as $l = v$ when the label Names and *data* with reductions an $rec(l = v, l_i = v_i)$

$(vec\ l_i)\langle x\rangle P \to rec(l_1 = v_1, \ldots, l_i = x, l_n = v_n) \mid P$. The values of Name-values in the BPEL process are l_{agent} and l_{host} *which contains special entries. In order to get the Names,* we communicate as bellow.

$$getloc(agent_host_i)\ P \stackrel{def}{=} getMem\ agentlagent_i)\ getMem\ host(host_i)\ P$$

Asynchronous send and synchronous receive are representing remote messaging. External services activities are swap of action messages in the similar process.

Remote Input

$$cmp_{send}(M, x) \stackrel{def}{=} pkt_{input/output}[M\langle x\rangle]$$

$$send(M, x).P \stackrel{def}{=} cmp_{send}(M, x) \mid P$$

Remote Output

$$cmp_{recv}(M, x, n) \stackrel{def}{=} pkt_{input/output}[M\langle x\rangle.n[M^{-1}\langle x\rangle]$$

$$recv(M, x).P \stackrel{def}{=} (vx)cmp_{recv}(M, x, n) \mid !(open\ n) \mid (x)\ P$$

P and Q process launched I/O components !open pktinput/output to make I/O actions embedded in pktinput/output work

$$agent_1[send(out\ agent_1\ in\ ch, n)\ P] \mid recv(out\ agent_2\ in\ ch, x)\ Q] \mid$$
$$ch[!open\ pkt_{input/output}] \to agent_1[P] \mid agent_2[Q\{x \leftarrow n\}] \mid ch[!open\ pkt_{input/output}]$$

In our work, an agent activity is contained a remote I/O action in a host. As seen in the following

Activity Remote Input

$$send'(ch, x) \stackrel{def}{=} getloc(agent_1, host_1)$$
$$send(out\ agent_1\ out\ host_1\ in\ ch, x)$$

Activity Remote Output

$$recv'(ch, x) \stackrel{def}{=} getloc(agent_1, host_1).$$
$$recv(out\ agent_1\ out\ host_1\ in\ ch, x)$$

3.2.2 Service locator for each πBPEL process

We reveal a large-scale locator communication for each πBPEL process, that have record *location* (loc) which returns a path to the mediator that has the most recent address of Name. We explore that each mediator has its own service space *Memory* as we prove earlier that represent real service swap in between client-server. Virtually only required service should be sent to one mediator to different mediator by analyzing service dependencies in linking different activities. In the following figure mediator performs *setv for* locating service.

Service Access

$$cmp_{loc}(M, l, M', n) \stackrel{def}{=} pkt_{loc}[M\ setloc\ l(M')\ n[M^{-1}]$$ *i.e.* M is roaming Loc, M' is Update Loc and M^{-1} is reverse Loc

$$setv(M, l, x, M')\ P \stackrel{def}{=} (vn)(cmp_{loc}(M, l, M', n) \mid open\ n\ setMem\ l(x)\ P)$$

$$cmp_{get}(M, l, x, n) \stackrel{def}{=} pkt_{loc}[M\ getloc\ l(M')\ pkt_{Mem}[M'\ getMem\ l(x)\ n[(M^{-1})^{-1}M^{-1}\langle x\rangle]]$$

$$getv(M, l, x)\ P \stackrel{def}{=} (vn)(cmp_{get}(M, l, x, n) \mid open\ n\ (x)\ P)$$

All Agents and locator services must control $!open\ pkt_{input/output}$ and $!open\ pkt_{Mem}$.
In our work all above said actions are communicate by the following methods.
We are to embody action in πBPEL as a process interface with I/O remote processes and remote variable access.

Remote Input

$cmp_{rec}(M,x) \stackrel{def}{=} pkt_{agent/output}[M.(x)]$

$send(M,x).P \stackrel{def}{=} cmp_{send}(M,x) \mid P$

Remote Output

$cmp_{rec}(M,x,n) \stackrel{def}{=} pkt_{agent/output}[M.(x).n[M^{-1}.(x)]$

$recv(M,x).P \stackrel{def}{=} (\nu x)cmp_{rec}(M,x,n) \mid !(open\ n) \mid (x).P$

P and Q process launched I/O components,'open pktinput/output to make I/O actions embedded in pktinput/output work.

i.e., $agent_1[send(out\ agent_1\ in\ ch,n).P] \mid recv(out\ agent_2\ in\ ch,x).Q] \mid$
$ch[!open\ pkt_{agent/output}] \rightarrow *agent_1[P] \mid agent_2[Q\{x \leftarrow n\}] \mid ch[!open\ pkt_{agent/output}]$

In our work, an agent activity is contained a remote I/O action in a host. As seen in the following

Activity Remote Input

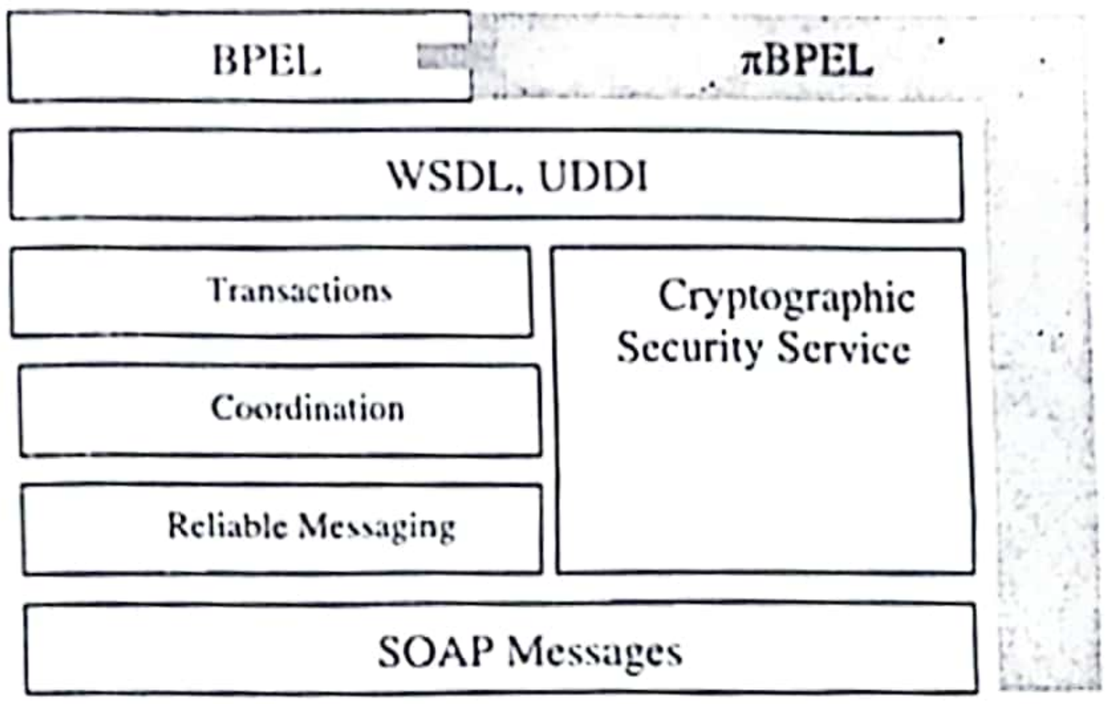
$send'(ch,x) \stackrel{def}{=} getloc(agent_i, host_i).$
$send(out\ agent_i\ out\ host_i\ in\ ch,x)$

Activity Remote Output

$recv'(ch,x) \stackrel{def}{=} getloc(agent_i, host_i).$
$recv(out\ agent_i\ out\ host_i\ in\ ch,x)$

We are to embody an activity in πBPEL as a process interaction with remote I/O processes and remote variable access.

4. πBPEL Mapping Activity to Process



Bellow we explore the πBPEL interaction activity process and expressions consequent to those in BPEL.

4.1 BPEL Process and πBPEL

First we judge the expression of a BPEL process. A host is activated in BPEL with the help of πBPEL expressions. We show bellow.

$[< process > activities < / process >] ::= [(\nu\ agent\ (l_1 ... l_n, ch_1 .... ch_n) sv)$

$agent[[activities]] \mid rec\ Mem(l_1 =\in, ..., l_n =\in l_{agent} = agent\ l_{host} = host) \mid !open\ pkt_{Mem}$

$\mid ch_1[out\ agent\ out\ host.!open\ pkt_{agent/output}] \mid ..... \mid ch_n[out\ agent\ out\ host.!open\ pkt_{agent/output}] \mid sv[rec\ loc(l_1 =\in, ..., l_n =\in)]]$

The Every new agent contains own Memory as well as processes communication channel (*channels* $ch_1,...,ch_n$) for action. Service locator *sv* to be activated the host Name. With existing processes P, as $(host[P \mid agent[...]])$ are contained in the host mutually. Activation and completion of control messages expressed as exchange control by structured activities (SA) in BPEL process. In section 4.2, shows the communicating of progressive movement from first to last *channel* c_i *for* action $activity_i$. In BPEL structured activities (switch, while, and pick) are expressions of branch and iteration [16 Luca Cardelli, 98]. All nested flow activities are activates simultaneously.

Theorem 3.1 Every πBPEL events can be uttered as a process P, πBPEL events are related to *SA*. *Bellow said* πBPEL events processes to be satisfied.

- $0 \in SA$.
- *if* $P' \xrightarrow{t} P$ *and* $P \in SA$ *then* $P' \in SA$.
- *if* $P \in SA$ *then* $ract.P \in SA$ *where* $ract \in \{send'(ch,x), recv'(ch,x), setv'(sv,l,x), getv'(sv,l,x)\}$.
- *if* $P,Q \in SA'$ *then* $P \mid Q \in SA'$.

An πBPEL events executed block is expressed as $P \in SA$. and also as comparable composition behavior belongs to SA.

Mobility: Mobility allows a code to move on other execution platform in proactive mode: a copy of the code and its current state is moved to the destination platform for being started from a specific point (weak mobility). Unlike the clonation, the code that performs the move primitive comes destroyed if the execution of the

$$migrate\ ch.P \stackrel{def}{=} (vb)(recv'(Ch,th).getloc(agent_t, host_t).out\ host_t\ in\ th.setr\ Mem\ l_{new}\ th.open\ b \mid b[P])$$

In predefined *channel*, first of all arrived Name of *th* (objective host) then roaming from the existing host to *th*, action will be updated in host *Memory*, and lastly starts execution of behavior in the block are invoked the platform We have the following reduction.

$$host_1[agent[migrate\ ch.P \mid Q \mid rec\ Mem\ (l_{new} = host_1, \bar{T} = \bar{v})] \mid R] \mid host_2[S] \mid ch[!open\ pkt_{migrate}] \mid send(in\ ch, host_2)$$

$$\rightarrow \cdot host_1[R] \mid host_2 \mid agent[P \mid Q \mid rec\ Mem\ (l_{new} = host_2, \bar{T} = \bar{v})]] \mid ch[!open\ pkt_{migrate}]$$

Clonation: A clone is a set of functions containing all projections and closed under composition. Clonation allows duplicating the code and the state of a running code. After a clonation there will be two identifiable codes in execution. The smallest clone containing a set of functions P is denoted by [P]. The clonatioa action is formulizing as bellow.

$$clone.P \stackrel{def}{=} (v\,agent)(recv'(ch,c).getloc(agent_t', host_t).agent[out\ agent_t.P$$
$$\mid rec\ Mem(l_{agent} = agent, l_{host} = host_t, \bar{T} = \bar{c}) \mid !open\ pkt_{Mem}]$$

Above said actions are predefined *channel* that is fired by the invoked platform, and makes a new-fashioned agent containing *P* outside of the inventive agent. We have the bellow reduction.

$$host_1[agent_1[clone.P \mid Q \mid rec\ Mem\ (l_{agent} = agent_1, l_{host} = host, \bar{T} = \bar{v})] \mid ch[!open\ pkt_{input/output}] \mid send(in\ ch, \in)$$
$$\rightarrow (v\,agent)\ (host[agent_1[Q \mid rec\ Mem\ (l_{agent} = agent_1, l_{host} = host, \bar{T} = \bar{v})] \mid agent[P \mid rec\ Mem\ (l_{agent} = agent, l_{host} = host, \bar{T} = \bar{c})$$
$$\mid !open\ pkt_{Mem}]] \mid ch[!open\ pkt_{input/output}]$$

***Communication*:** Communication (through exchange of messages) is possible only between cloning and cloned. Messages are sent-received in asynchronous-synchronous fashion. The virtual machine relies on a calculus which describes the semantics of the minimal set of operation isolated to characterize a platform supporting mobile code [Jianwei Yin, 2005]. The calculus for modeling πBPEL technology is summarized here below.

4.2 Barbed Equivalence

In section 3.2.1 and 3.2.2, we classify barbs that hold initiate mechanism of remote I/O events. Though general barbs Theorem 3.1. catch the primitive actions which is based on observable behaviors of processes[M. Vigliotti, 2002]. As we use the barbed bisimulation notion.

$$P \downarrow send'(ch,x) \stackrel{def}{=} P \equiv (vm_1 \ldots m_n)(cmp_{\ldots,x}(in\ ch,x) \mid P')$$

$$P \downarrow setv'(sv,l,x) \stackrel{def}{=} \exists M', n\ such\ that\ P \equiv (vm_1 \ldots m_n)(cmp_{\ldots}(in\ sv.l, out\ sv.M', n) \mid P')$$

$$P \downarrow recv'(ch,x) \stackrel{def}{=} \exists_n\ such\ that\ P \equiv (vm_1 \ldots m_n)(cmp_{\ldots,x}(in\ ch,x,n) \mid P')$$

$$P \downarrow getv'(sv,l,x) \stackrel{def}{=} \exists n\ such\ that\ P \equiv (vm_1 \ldots m_n)(cmp_{sv.l}(in\ sv,l,x,n) \mid P')$$

These barbs exited the agent, host and to go through *channel or sv*. The Name (n) is for synchronization process for launching mechanism, and M' is the return process path. In barbs n and M' are ignored. So the following equivalence doesn't concern with host and agent components.

$$P \Downarrow M \overset{def}{=} P \rightarrow^{*} Q \text{ and } Q \downarrow M$$

Barbed Bisimulation satisfies the following
Conditional relation **R** with bisimulation

if PRQ *then*

if $P \Downarrow M$ *then* $P \Downarrow M$

if $P \rightarrow P'$ *then for some* $Q', Q \rightarrow^{*} Q'$ *and* $P'RQ'$

If there exists bisimulation **R** is satisfying PRQ, *then* $P \approx Q$.

4.3 Barbs Equivalence πBPEL primary lemmas

In this section, our work is focused on internal and external actions (host and agent), which behave the *location* activities.

Location

$$\frac{P \approx 0}{host_1[agent_1[P \mid rec\ Mem(l_{agent} = agent_1, l_{host} = host_1, \tilde{l} = \tilde{v})]] \approx host_2[agent_2[P \mid rec\ Mem(l_{agent} = agent_2, l_{host} = host_2, \tilde{l} = \tilde{v})]]}$$

The bisimulation of two processes are added to inner processes. Different activities can be held to obtain similar actions recognizable from outside, conditionally the particular entries are correctly set and the mechanism take the related actions after the receiving each existing agent or host.

Same way, two inner equivalent *composition* processes indicates barbs equivalence of parallel composition. It is fact that without *channels* or locator services remote events cannot finish their events outside of hosts or directly process never accesses in its environments.

Composition

$$\frac{\begin{array}{l} host_1[agent_1[P_1 \mid rec\ Mem(l_{agent} = agent_1, l_{host} = host_1, \tilde{l} = \tilde{v})]] \approx host_2[agent_2[P_2 \mid rec\ Mem(l_{agent} = agent_2, l_{host} = host_2, \tilde{l} = \tilde{v})]] \\ host_1[agent_1[Q_1 \mid rec\ Mem(l_{agent} = agent_1, l_{host} = host_1, \tilde{l} = \tilde{v})]] \approx host_2[agent_2[Q_2 \mid rec\ Mem(l_{agent} = agent_2, l_{host} = host_2, \tilde{l} = \tilde{v})]] \end{array}}{host_1[agent_1[P_1 \mid Q_1 \mid rec\ Mem(l_{agent} = agent_1, l_{host} = host_1, \tilde{l} = \tilde{v})]] \approx host_2[agent_2[P_2 \mid Q_2 \mid rec\ Mem(l_{agent} = agent_2, l_{host} = host_2, \tilde{l} = \tilde{v})]]}$$

where $P, Q, P', Q' \in SA$ and $ract \in \{send'(ch,x), recv'(ch,x), setv'(sv,l,x), getv'(sv,l,x)\}$

5. πBPEL Technology Approach

We now present a formal description of participants (*Net, Customer, Client and Server*) and how they can be composed in e-Business scenario.

$$Net(\,) := (a_c)(a_p)(c_1)(p_1)(c_2)(p_2)\,(c_3)(p_3)\,(Customer(a_c, a_p) \mid Client(a_c, a_p, c_1, p_1, c_2, p_2, c_3, p_3) \mid Server_i((c_i, p_i)$$

The process *Net*( ) composes the various participants to the scenario; first of all, it creates some global *channels*, used by the processes to interact together: the *channels* a_c and a_p are the *Client* interfaces exposed to the *Customers*. So, they are passed as arguments both to the *Customer* (a_c, a_p) and to the *Client* $(a_c, a_p, c_1, p_1, c_2, p_2, c_3, p_3)$ processes. The first one is used to require a price list, while the second one to emit a purchase orders. The other global *channels* are the set of pair c_i and p_i, which are respectively the query and the purchase interface of the i^{th} *Server*. Those names are passed as arguments to the *Client* $(a_c, a_p, c_1, p_1, c_2, p_2, c_3, p_3)$ and $Server_i(c_i, p_i)$ processes. We also assume the *Net* as a closed system, in the sense that $(fn(\,)\ Net(\,)) = \emptyset$.

5.2 Customer

$$Customer(a_c, a_p) := (\tilde{q}_1)(\tilde{q}_2)(\tilde{q}_3)(a_r)(a_s)(a_f)(\overline{a}_c\langle a_r, \tilde{q}_1, \tilde{q}_2, \tilde{q}_3, a_s, a_f\rangle \mid a_r(\tilde{l}_1, \tilde{l}_2, \tilde{l}_3).\overline{a}_p\langle a_r, \tilde{q}_1, \tilde{q}_2, \tilde{q}_3, a_s, a_f\rangle \mid a_s().S() \mid a_f().F())$$

The *Customer* process first browses a price list. When it receives an answer, it emits a purchase request, and waits for the outcome. To do this it creates these names: $\tilde{q}_1, \tilde{q}_2$ *and* $\tilde{q}_3$ which contain the two item preferences, the channel a_r, which is the restricted reply channel used by the *Client* to inform the *Customer* about the price list consultation, and the two *channels* a_s (success) and a_f (failure), which signal respectively the outcome of the purchase transaction. Then the *Customer* process sends the message $\overline{a}_c\langle \tilde{q}_1, \tilde{q}_2, \tilde{q}_3, a_r\rangle$ to the *Client* consultation interface. This message carries the items description and the reply channel. This first phase ends with the receipt of the reply message $a_r(\tilde{l}_1, \tilde{l}_2, \tilde{l}_3)$., which carries three names, $\tilde{l}_1, \tilde{l}_2$ *and* $\tilde{l}_3$, encoding the features of the requested items. When the *Customer* process receives one of this message, the purchase transaction has completed and it goes on with the appropriate task identified by $S(\,)$ or $F(\,)$. Moreover, it is guaranteed that either all the items have been bought, or the appropriate compensations have been emitted.

CamScanner

5.3 Client

$$Client\ (a_c, a_p, c_1, p_1, c_2, p_2, c_3, p_3) := a_c(\tilde{q}_1, \tilde{q}_2, \tilde{q}_3 a_r).\ (\pi BPEL(a_p, c_1, p_1, c_2, p_2, c_3, p_3, \tilde{q}_1, \tilde{q}_2, \tilde{q}_3, a_r)$$
$$| Client(a_c, a_p, c_1, p_1, c_2, p_2, c_3, p_3))$$
$$\pi BPEL(a_p, c_1, p_1, c_2, p_2, c_3, p_3, \tilde{q}_1, \tilde{q}_2, \tilde{q}_3, a_r) := Query(c_1, c_2, c_3, \tilde{q}_1, \tilde{q}_2, \tilde{q}_3, a_r)$$
$$| PURCHASE(a_p, p_1, p_2, p_3)$$
$$Query(c_1, c_2, c_3, \tilde{q}_1, \tilde{q}_2, \tilde{q}_3, a_r) := (r_1)(r_2)(\bar{c}_1\langle \tilde{q}_1, r_1\rangle | \bar{c}_2\langle \tilde{q}_2, r_2\rangle | \bar{c}_3\langle \tilde{q}_3, r_3\rangle | (r_1(\tilde{q}_1, \tilde{l}_1).r_2(\tilde{q}_2, \tilde{l}_2).r_3(\tilde{q}_3, \tilde{l}_3)a_r\langle \tilde{l}_1, \tilde{l}_2, \tilde{l}_3\rangle$$
$$Purchase(a_p, p_1, p_2, p_3) := a_p(\tilde{q}_1, \tilde{q}_2, \tilde{q}_3, a_s, a_f).(r_s^1)(r_f^1)(r_s^2)(r_f^2)(r_s^3)(r_f^3) | \bar{p}_1\langle \tilde{q}_1, r_s^1, r_f^1\rangle | \bar{p}_2\langle \tilde{q}_2, r_s^2, r_f^2\rangle | \bar{p}_3\langle \tilde{q}_3, r_s^3, r_f^3\rangle$$
$$| WAIT(r_s^1, r_f^1, r_s^2, r_f^2, r_s^3, r_f^3, a_s, a_f))$$

The *Client* process exposes a service that can be used by a *Customer* to query some distributed price lists, and subsequently to purchase the items.

When it receives a request $a_c(\tilde{q}_1, \tilde{q}_2, \tilde{q}_3, a_r)$, it executes a managing process — ($\pi BPEL(a_p, c_1, p_1, c_2, p_2, c_3, p_3, \tilde{q}_1, \tilde{q}_2, \tilde{q}_3, a_r)$). the $(\pi BPEL(a_p, c_1, p_1, c_2, p_2, c_3, p_3, \tilde{q}_1, \tilde{q}_2, \tilde{q}_3, a_r))$ process executes two sub-processes $Query(c_1, c_2, c_3, \tilde{q}_1, \tilde{q}_2, \tilde{q}_3, a_r)$ and $Purchase(a_p, p_1, p_2, p_3)$.

The first of these subtasks, $Query(c_1, c_2, c_3, \tilde{q}_1, \tilde{q}_2, \tilde{q}_3, a_r)$, receives the consulting *channels* c_1 and c_2, the *Customer* preferences $\tilde{q}_1, \tilde{q}_2$ *and* $\tilde{q}_3$ and the reply channel a_r. It emits in parallel the various price list consultations with the messages $\bar{c}_1\langle \tilde{q}_1, r_1\rangle$, $\bar{c}_2\langle \tilde{q}_2, r_2\rangle$ *and* $\bar{c}_3\langle \tilde{q}_3, r_3\rangle$ that contain the *Customer* preferences and the private *channels* r_1 and r_2 on which it will wait for a reply. Those replies contain the outcomes of the queries executed on the *Server*'s databases — encoded with names $\tilde{l}_1, \tilde{l}_2$ *and* $\tilde{l}_3$. When the *Client* receives them, it forwards them to the *Customer* application with the message $\bar{a}_r\langle \tilde{l}_1, \tilde{l}_2, \tilde{l}_3\rangle$, and it waits for a purchase order on the channel $a_p(\tilde{q}_1, \tilde{q}_2, \tilde{q}_3, a_s, a_f)$.

The process $Purchase(a_p, p_1, p_2)$ is called with the channel a_p, on which it will wait for the *Customer*'s order, and the *Servers*' *channels* p_1 and p_2. At this point, it creates a pair of success/failure reply *channels* r_s and r_f for each *Server*, and emits the purchase requests $p_1\langle \tilde{q}_1, r_s^1, r_f^1\rangle, p_2\langle \tilde{q}_2, r_s^2, r_f^2\rangle$ *and* $p_3\langle \tilde{q}_3, r_s^3, r_f^3\rangle$. When the requests have been emitted, the process *Purchase* (a_p, p_1, p_2, p_3) executes the process $WAIT(r_s^1, r_f^1, r_s^2, r_f^2, r_s^3, r_f^3, a_s, a_f)$, which will manage the purchase transactions' outcomes.

Due to space limitation, the following specified activates process are omitted such as CP_i, PP_i, DBP_i.

5.4 Server

$Server_i(c_i, p_i) := (db_c)(db_p)(\mathrm{CP}_i(c_i, db_c) | \mathrm{PP}_i(p_i, db_p)$ $|\mathrm{DBP}_i(db_c, db_p))$

The generic i^{th} *Server* receives two names as arguments, c_i and p_i. When the *Server* process begins its execution, it creates a pair of *channels*, which are used to interact with a database process. The *channel* db_c is used to invoke a price list consultation service exposed by the database; the *channel* db_p is used to emit a purchase order to the same database. After the creation of these *channels*, the *Server* creates three sub-processes, $\mathrm{CP}_i(c_i, db_c)$, $\mathrm{PP}_i(p_i, db_p)$ and $\mathrm{DBP}_i(db_c, db_p)$. The first two processes manage the consultation and the purchase orders emitted by the *Customer*, while the third one represents a database process.

6. Conclusion

The πBPEL is a programming language suited for modeling Internet applications. πBPEL captures their inherent parallelism, the repartition of computational elements amongst multiple locations, the mobility of resources and locations. We discussed a reference implementation and using BPEL based communication innovated in XML. The pattern-based design is the aim that a similar model can be implemented in any language with any collaboration services framework. Our propose in our calculus work to intend concrete contribution from the point of view of the orchestration language specifications in terms of unambiguous semantics and of the feasible implementations of BPEL orchestration engines. We introduced πBPEL, a simple extension of the π-calculus with long running transactions.

Some considerations about the choice of a π-calculus based language are needed. One peculiarity of such language is mobility: it is possible to transmit channel names that then can be used by any process receiving them. This aspect is essential in the formalization of BPEL and in particular plays an important role in the formalization of interaction with request–response services. Indeed, the invoker must send to the

service a channel name to be used to return the response, thus we have to exploit name mobility This proposed model representation in a calculus in which test and formal proofs could be performed, or into a executable representations in πBPEL languages. We deem that given formalization will help us to explore the practical aspects of our framework. This work contributes a simple, concise until now dominant and communicative formal language, with a concrete semantics that allows formal interpretation. The language shows a clear relation with the π-calculus, and the actual encoding of it with the π-calculus is a feasible task, while it would be quite harder to get such an encoding for BPEL and other collaboration services composition languages.

References

Bocchi, L., C. Laneve, and G. Zavattaro, (2003) "A Calculus for Long-running Transactions". In *FMOODS'03*. ©Springer-Verlag.

Bruni, R., C. Laneve, and U. Montanari, (2003) "Orchestrating Transactions in Join Calculus". In *CONCUR'02*. © Springer-Verlag.

Bugliesi, M., G. Castagna, and S. Crafa, January (2004) "Access Control for Mobile Agents: The Calculus of Boxed Ambients", *ACM Transactions on Programming Languages and Systems* 26(1): 57-124. © ACM Press.

Butler, M., and C. Ferreira., (2004) "An Operational Semantics for StAC, a Language for Modelling Long-running Business Transactions". In *COORDINATION'* 04*:* © Springer-Verlag.

Bruni, R. H. Melgratti, and U. Montanari. (2005) "Theoretical Foundations for Compensations in Flow Composition Languages". To appear in POPL-.

Cardelli, L., and A.D. Gordon, (1998) "Mobile ambients", in: M. Nivat (Ed.), Lecture Notes in Computer Science, Vol. 1378, 140–155. © Springer, Berlin, Germany.

Cardelli, L., G. Ghelli, and A.D. Gordon, (2000) "Secrecy and group creation". In CONCUR'2000: © Springer-Verlag.

Chessel, M., D. Vines, C. Griffin, V. Green, and K.Warr, January (2001) "Business Process Beans: System Design and Architecture Document". Technical report. IBM UK Laboratories.

Ishikawa, F., Y. Tahara, N. Yoshioka, and S. Honiden, July, (2004) "Behavior Descriptions of Mobile Agents for Web Services Integration", 2004 IEEE International Conference on Web Services (ICWS 2004) 342-349.

Kavantzas, N., G. Olsson, J. Mischkinsky, and M. Chapman, "Web Services Choreography Description Languages". [otn.oracle.com/tech /webservices/ htdocs/ spec/cdl v1.0.pdf.

Milner, R., October (1991) "The polyadic π–calculus: a tutorial. Laboratory for Foundations of Computer Science , Computer Science Department, University of Edinburgh, The King's Buildings, Edinburgh, UK.

Memon A.G., J. Yin, J.D. Maree, and Mujeeb-u-Rehman," (2005) Deploying π-calculus Technology in Inter-organizational Process ",WAIM2005© Springer-Verlag.

Parrow. J., (2001) "An Introduction to the π-calculus" *Handbook of Process Algebra,ed. Bergstra, Ponse and Smolka,* pages 479–543,. © Copyright Elsevier.

Parrow, J., and B. Victor, "The Fusion Calculus: Expressiveness and Symmetry in Mobile Processes". In *LICS'98:*. © IEEE Computer Society Press.

Sangiorgi, D., (1997) "The Name Discipline of Receptiveness". In Automata, Languages and Programming: ICALP 97. © Springer-Verlag,

Vigliotti, M., and I. Phillips, (2002) "Barbs and congruences for safe mobile ambients". In F-WAN: Foundations of Wide Area Network Computing.

Yin, J., A.G. Memon, J. D. Maree, and Mujeeb-u-Rehman," (2005) π_{RBT}-calculus Protocol for Web service Long Running Transactions", (ISCIT05).© IEEE.

CamScanner