A Unified Formal Foundation for Service Oriented Architectures

Frank Puhlmann

Business Process Technology Group
Hasso-Plattner-Institute for IT Systems Engineering
at the University of Potsdam
D-14482 Potsdam, Germany
`frank.puhlmann@hpi.uni-potsdam.de`

Abstract: This paper summarizes how an algebra for mobile systems, the π-calculus, can be applied as unified formal foundation to service oriented architectures (SOA). The concepts accounted are orchestrations including data and processes, as well as choreographies consisting of interacting processes. Since SOAs incorporate agile binding of interaction partners, static process structures as found in Petri nets are not sufficient for completely representing orchestrations and choreographies. The π-calculus, in contrast, inherently supports link passing mobility required for agile interacting processes.

1 Introduction

Service oriented architectures (SOA) as introduced by Burbeck in [Bur00] consists of three major roles. A *service provider* publishes information about the services it offers at a *service broker*, where in turn they can be found by *service requesters*. Once a service requester decides to incorporate a service provider, it dynamically binds to it. While today there exist many standards from the WS-stack (e.g. [CCMS01, W3C04, BEA03]) that describe how certain aspects of a SOA can be implemented, as well as scientific research regarding specific areas (e.g. [vAW01, Mar05, SBS04]), a unified formal foundation for service oriented architectures is still missing. None of the existing standards is based on a common formal foundation, and most of the scientific approaches utilize either Petri nets or proprietary formalizations. Since this leads to a lack of common understanding and continuous research, this paper summarizes our results on investigating an algebra for mobile systems, the π-calculus [MPW92], as a unified foundation for SOA. In contrast to Petri nets [Pet62], the π-calculus inherently supports link passing mobility required for dynamic binding in agile interactions.

Figure 1 depicts link passing mobility. The left hand side shows the three different roles of a SOA, denoted as circles. A service requester (R) knows a service broker (B), denoted by the line (link) between, where the dot denotes the target. The service broker, in turn, has knowledge about a number of service providers (P). The service broker evaluates the
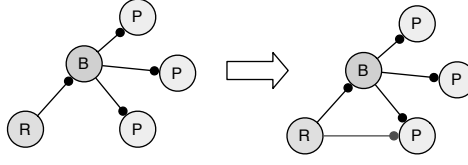
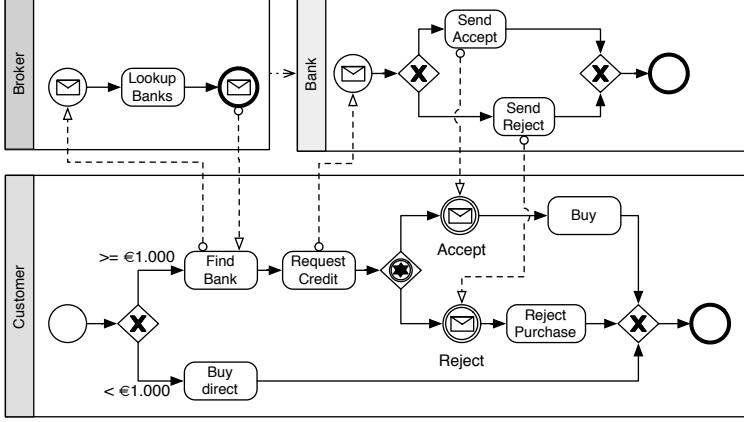Figure 1: A service oriented architecture in π-calculus.

Figure 2: An example choreography in BPMN notation.

request of the service requester and returns the corresponding link. The service requester then uses this link to dynamically bind to the service provider. Hence, the link structure of the example changes over time as shown at the right hand side of the figure.

The remainder of this paper introduces how the π-calculus can be used to represent data and processes required for orchestrations as well as additional ingredients to interactions such as correlations required for choreographies. It is supplemented by a running example and a section presenting application areas. We start by extending the motivation and introduce the π-calculus.

2 Motivation and Related Work

Figure 2 contains a choreography in BPMN [BPM04] notation. The choreography consists of the purchase process of a *Customer*. It dynamically includes a *Bank* found by a *Broker* if the purchase is above a threshold value. The *Customer's* pool shows the purchase process. First a data-based decision is made between the upper and lower part of the process. If the price of the purchase is lower then a given value, the purchase is made directly. If the price is above, a *Broker* is contacted to find a *Bank* with the lowest interests regarding the

purchase. Thereafter, the credit is requested at the *Bank*. In the considered scenario, the *Bank* can send two possible answers, either accept or reject the credit. This is evaluated in the *Customer's* process using a deferred choice. Based on the decision of the *Bank*, the purchase is made or rejected.

While the example choreography looks like a static interaction between *Customer*, *Broker*, and *Bank*, it is indeed an agile one. Only the *Broker* is known at design time, so it is directly coded into the *Find Bank* activity. However, the credit request relies on the answer of the *Broker's* lookup. Thus, the *Customer's* process dynamically binds to a certain *Bank* only known at runtime. The example contains typical ingredients of a SOA: internal processes with data (orchestrations) as well as static and dynamic interactions (choreographies).

The example can be discussed using approaches ranging from Petri nets to process algebra as well as existing standards. Petri nets have a long tradition for formalizing business processes [vAvH02]. They have also been extended to support distribution, e.g. Weske et al. in [vAW01]. Recent work investigated the representation of services and compatibility, e.g. Martens using *usability* in [Mar03], refined by Massuthe et al. using *operating guidelines* in [MRS05]. However, regarding distributed business processes, Petri net based approaches have two major drawbacks. First, they do not support more advanced routing pattern required for real-world business processes [vAtH03]. Second, they do not support interaction patterns that require dynamic binding [BDtH05]. Regarding the given example, only simple routing patterns are contained that cause no problems. However, the dynamic binding would generate infinite Petri nets if we assume an unknown number of possible participants. Extensions for special cases are possible but have a low general adequacy. Process algebra based approaches often neglect mobility aspects [BS05], but even if they consider it, there exist no investigations on full adequacy regarding advanced routing patterns. Thus, there exist no process algebra based approach until now that has been scientifically investigated regarding distributed business processes. Our motivation on investigating the π-calculus can be found in [Puh06].

3 The π-calculus

The π-calculus is based on a labeled transition system given by $(P, T, \xrightarrow{t})$, where P represents the set of states, i.e. all possible process descriptions, T is a set of transition labels, and $\xrightarrow{t}$ is a transition relation for each $t \in T$. To get started with the π-calculus, knowledge about process structures is sufficient, while the semantics can be used informally. Inside π-calculus processes *names* are used to represent links or pointers. Processes are denoted by uppercase letters and names by lowercase letters. The processes (i.e. states) of the π-calculus are given by:

$$P ::= M \mid P|P' \mid \mathbf{v}zP \mid !P \mid P(y_1, \cdots, y_n)$$
$$M ::= \mathbf{0} \mid \pi.P \mid M + M' \tag{1}$$
$$\pi ::= \overline{x}\langle \tilde{y} \rangle \mid x(\tilde{z}) \mid \tau \mid [x = y]\pi \ .$$

The informal semantics is as follows: $P|P'$ is the concurrent execution of P and P', $\mathbf{v}zP$ is the restriction of the scope of the name z to P, $!P$ is an infinite number of copies of P, and $P(y_1, \cdots, y_n)$ represents recursion. $\mathbf{0}$ is inaction, a process that can do nothing, $M + M'$ is the exclusive choice between M and M'. The output prefix $\overline{x}\langle\tilde{y}\rangle.P$ sends a sequence of names $\tilde{y}$ via the co-name $\overline{x}$ and then continues as P. The input prefix $x(\tilde{z})$ receives a sequence of names via the name x and then continues as P with $\tilde{z}$ replaced by the received names (written as $\{^{na\tilde{m}e}/_{\tilde{z}}\}$). Matching input and output prefixes might communicate, leading to an interaction. The unobservable prefix $\tau.P$ expresses an internal action of the process, and the match prefix $[x = y]\pi.P$ behaves as $\pi.P$, if x is equal to y.

4 Unification

This section discusses, based on the example previously introduced, how the π-calculus can be used as a unified formal foundation for service oriented architectures. It starts by formalizing data, followed by processes, and is concluded by interactions.

4.1 Data

While the π-calculus is a process algebra and states are denoted as a syntactical description of a process given by the grammar from equation 1, it still allows for representing arbitrary data structures. Since pointers in the π-calculus are represented by names, names are used as accessors for data structures represented by processes. But how can static data like a boolean value be represented in the π-calculus? A straightforwards way is defining two restricted names for true and false, i.e. $(\mathbf{v}\top)$ for true and $(\mathbf{v}\bot)$ accordingly for false. These names are globally accessible by all processes inside the system, e.g.

$$SYSTEM = (\mathbf{v}\top, \bot)(A \mid B)$$
$$A = \tau.(\overline{ch}\langle\top\rangle.A + \overline{ch}\langle\bot\rangle.A)$$
$$B = ch(x).([x = \top]\tau.B' + [x = \bot]\tau.\mathbf{0}) .$$

The process $SYSTEM$ first creates two restricted names used for true and false and thereafter starts the processes A and B in parallel. However, only A can start execution immediately in the example, since B waits for a name x on ch. A does some internal calculation denoted by τ and thereafter sends either true or false via the name ch. In both cases, process A continues execution using recursion. Process B receives the name, where technically the placeholder x is replaced by the received name, i.e. either $\top$ or $\bot$. So, if A has sent $\top$, B evolves as follows:

$$ch(x).([x = \top]\tau.B' + [x = \bot]t.\mathbf{0}) \xrightarrow{ch(\top)} [\top = \top]\tau.B' + [\top = \bot]\tau.\mathbf{0} .$$

Since $\top \neq \bot$, only one possibility of the choice for B remains, making it deterministic (B executes τ and thereafter B'). If A had send $\bot$ instead, the second part of B would have

been executed. When looking at ch as a pointer, it clearly points to a process (A), that is able to return either true or false. Consequently, the *type* of the name ch can be said to be boolean, since it always points to a process representing boolean values in the example. By comparing two names of the type boolean, an AND process can be constructed:

$$AND = and(b1, b2, resp).b1(x).b2(y).([x = \top][y = \top]\overline{resp}\langle\top\rangle.AND+$$
$$[x = \bot]\overline{resp}\langle\bot\rangle.AND+$$
$$[y = \bot]\overline{resp}\langle\bot\rangle.AND) \ .$$

The AND process is invoked via and with three parameters: Two names $b1$ and $b2$ representing pointers to booleans, and a third name used as response channel. First, AND fetches the actual values of the boolean pointers. Second, it returns true if both names, $b1$ and $b2$, are true and false otherwise. A possible extension for representing a byte instead of a boolean would be for A to return eight parameters of either true or false instead of one: $\overline{ch}\langle\bot, \bot, \top, \bot, \top, \bot, \top, \bot\rangle$ (representing decimal 42). One can now construct similar processes for adding, subtracting, or comparing bytes. Using these, other types can be implemented. Based on the observations made, we define a syntactical refinement to the π-calculus, namely typed names. A name is typed using common data types such as $\mathbf{v}i : Integer$. We further on introduce $[x == y], [x < y], [x > y]$, and $[x \neq y]$ for typed names similar to their obvious meaning. Technically, each of these construct is expanded to $(\mathbf{v}resp)\overline{proc_{type}}\langle x, y, resp\rangle$, where $proc_{type}$ calls a process for the corresponding operation and data type similar to AND.

After it has been shown how data can be represented in the π-calculus, it remains open how it can be made persistent, i.e. allowing for memory cells, stacks, queues, lists, and so on. A memory cell able to capture and allow modification of a single π-calculus name is denoted as:

$$CELL = !(\mathbf{v}c)\overline{cell}\langle c\rangle.CELL_1(\bot)$$
$$CELL_1(n) = \bar{c}\langle n\rangle.CELL_1(n) + c(x).CELL_1(x) \ .$$

The process $CELL$ is replicated each time a fresh, restricted name c is read using $cell$. A new memory cell is initialized with the default name $\bot$ (false). The name c retrieved is then used as read and write accessor to the cells content. Thus, using c as an output prefix, the content of the memory cell can be changed, while using c as an input prefix, the content is read. Based on the memory cell, arbitrary data structures can be defined in the π-calculus.

4.2 Processes

After having introduced the core foundations for each information system, namely data, the representation of business processes in the π-calculus is investigated. Business processes are composed out of routing patterns that specify the control flow between activities. Examples are *Split*, *Join*, *Discriminator*, or patterns representing *Multiple Instances* of activities. A widely acknowledged catalogue of such patterns has been published by

van der Aalst et al. [vAtHKB00] as *Workflow Patterns*. In [PW05] we have shown how all these patterns can be represented in the π-calculus. In [OPW05] the pattern catalogue is extended by another pattern common in distributed business processes, called *Event-based Rerouting*. In general, each activity (including gateways etc.) of a business process is mapped to a π-calculus process, according to the following structure:

$$\{x_i\}_{i=1}^m.\{[a=b]\}_1^n.\tau.\{\overline{y_i}\}_{i=1}^o.\mathbf{0} . \tag{2}$$

Curly brackets are used to denote a static sequence of actions, the same holds for $\prod$ and $\sum$ for products and summations. Each activity consists of pre- and postconditions for routing the control flow using π-calculus names. The (abstracted) functional perspective of the activity is represented by the unobservable action τ. The precondition is split into two part: (1) $\{x_i\}_{i=1}^m$ denotes that the activity waits for m incoming names, and (2) $\{[a=b]\}_1^n$ denotes n additional guards that have to be true to execute the activity. The postcondition denotes the triggering of o outgoing names, i.e. $\{\overline{y_i}\}_{i=1}^o$. It is notable that the π-calculus representation replaces the type vs. instance view of business processes by a prototypical approach. Business processes are described formally in π-calculus and duplicated for execution.

By looking at the example again, the Workflow Patterns found in figure 2 are *Sequence* (BPMN sequence flow), *Exclusive Choice* and *Simple Merge* (BPMN exclusive OR gateways), as well as *Deferred Choice* (BPMN event-based gateway). A *Sequence* between two activities is represented by two different π-calculus processes:

$$A = \tau_A.\overline{b}.\mathbf{0} \qquad B = b.\tau_B.B' ,$$

where A signals the name b as a postcondition to B, that in turn can start execution. To make the formalization actually work, a third process, representing the composed system is required:

$$SYSTEM = (\mathbf{v}b)(A \mid B) .$$

Such a process is assumed for all further Workflow Patterns. An *Exclusive Choice* between two different activities B and C after an activity A is represented by:

$$A = \tau_A.(\overline{b}.\mathbf{0} + \overline{c}.\mathbf{0}) \qquad B = b.\tau_B.B' \qquad C = c.\tau_C.C' .$$

Note that the pattern given makes a non–deterministic choice between B or C. However, by using techniques such as $[x == y]$ as shown in the previous section, the choice can be made deterministic. The *Simple Merge* pattern waits for either one of the preceding activities, e.g. activity D waits for either B or C:

$$B = \tau_B.\overline{d_1}.\mathbf{0} \qquad C = \tau_C.\overline{d_2}.\mathbf{0} \qquad D = d_1.\tau_D.D' + d_2.\tau_D.D' .$$

A *Deferred Choice* pattern defers the decision until an external event occurs:

$$A = \tau_A.(b_{env}.\overline{b}.\mathbf{0} + c_{env}.\overline{c}.\mathbf{0}) \qquad B = b.\tau_B.B' \qquad C = c.\tau_C.C' .$$

After all patterns required for the example have been defined, the orchestrations from figure 2 can be formalized in π-calculus. Basically, each orchestration can be seen as a
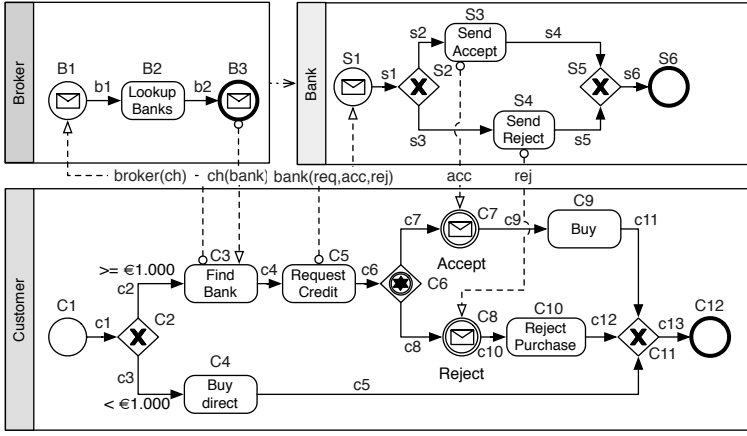
Figure 3: Example choreography annotated with identifiers.

strongly connected graph with exactly one initial and one final node. The nodes of the graph represent activities, events, or routing elements, while the edges represent dependencies between them. Each node has a semantics, e.g. an activity can be executed, an event can be consumed, or a routing decision can be made. Obviously, the semantics is given to the nodes by using the π-calculus pattern formalizations according to the following simplified algorithm:

- Assign all nodes of the graph a π-calculus process identifier.

- Assign all edges of the graph a unique π-calculus name.

- Define the π-calculus processes according to the π-calculus mapping of the Workflow Patterns found in [PW05, OPW05] as given by the type of the corresponding node. Each functional part of an activity is represented by the unobservable prefix τ since it is abstracted from concrete realizations.

- Define a global process that places all π-calculus processes representing nodes in parallel. □

An extended mapping approach from BPMN to π-calculus can be found in [PW06]. Regarding the example, we first have to annotate each node of the graphs representing the orchestrations with a π-calculus process identifier and each flow with a π-calculus name. The result is shown in figure 3. We also annotated the BPMN message flows, however this is not required until the complete choreography is defined in the next subsection. The global process (orchestration) of the broker is a quite simple:

$$BROKER = (\mathbf{v}b1, b2)(B1(b1) \mid B2(b1, b2) \mid B3(b2)) \,,$$

with the following components representing the nodes:

$$B1(b1) = \tau_{B1}.\overline{b1}.\mathbf{0} \qquad B2(b1, b2) = b1.\tau_{B2}.\overline{b2}.\mathbf{0} \qquad B3(b2) = b2.\tau_{B3}.\mathbf{0} \,.$$

Initially, all nodes are placed in parallel in process $BROKER$. However, only component $B1$ can start immediately, since all other components require preconditions denoted by π-calculus names. The orchestration of the bank contains an exclusive decision that is not further specified. Due to space limitations, we only show the important π-processes:

$$S2(s1, s2, s3) = s1.\tau_{S2}.(\overline{s2}.\mathbf{0} + \overline{s3}.\mathbf{0})$$
$$S5(s4, s5, s6) = s4.\tau_{S5}.\overline{s6}.\mathbf{0} + s5.\tau_{S5}.\overline{s6}.\mathbf{0} \ .$$

The orchestration of the customer extends the representation of a decision node by taking data values into account, thus making the decision deterministic.[1] Again, we showcase the relevant process:

$$C2(c1, c2, c3) = c1(value).\tau_{C2}.([value > 999]\overline{c2}.\mathbf{0} + [value < 1000]\overline{c3}.\mathbf{0}) \ .$$

The π-calculus process $C2$ representing a data-based exclusive choice receives the $value$ to be evaluated from the preceding process. Based on the value, the decision is made. Note that $value$ is a pointer to a process representing a real number. Furthermore, $[value > 999]$ and $[value < 1000]$ are just placeholders for data-evaluation processes as introduced in the previous section. The representation of the event-based gateway will be given in the next subsection, since the decision is based on interactions between the customer and the bank.

4.3 Interaction

Agile choreographies, as contained in the example, are closely linked to the *Service Interaction Patterns* by Barros et al. [BDtH05]. These patterns describe possible behavior inside choreographies. Examples are *Send* and *Receive*, or *Dynamic Routing*, where the next interaction partner is determined by the current activity. In [DPW06] we have shown how the interaction patterns can be represented in the π-calculus. A synchronous service invocation is denoted as follows:

$$A = \overline{b}\langle msg \rangle.A' \qquad B = b(msg).B' \ ,$$

where A is the service requester and B is the service provider. The formalization leaves it open if A knows the link b since design time or acquired it during runtime. If it is defined as

$$S = (\mathbf{v}b)(A \mid B) \ ,$$

A and B share the link b since design time. Using link passing mobility in π-calculus, we can model a repository $R = \overline{lookup}\langle b \rangle.R$ that transmits the link at runtime:

$$S = (\mathbf{v}lookup)(lookup(b).A \mid ((\mathbf{v}b)B \mid R)) \ .$$

A common problem in the area of service oriented architectures is the description of correlations between service requesters and service providers. Usually, some kind of correlation

---

[1]This does not imply that the bank makes non-deterministic choices. However, the algorithm is not contained in the (abstract) orchestration.

identifier is placed inside each request and reply. The invoker as well as the provider have to take care to match all requests. In the π-calculus, the unique identifier of a request is also the channel used for reply from the service. By merging these two concepts, a clear representation of the correlations is straightforward. A new unique identifier is a π-calculus name, which is created with the **v** operator. A refined treatment of this topic can be found in [OPW05].

After having introduced the prerequisites, we are now ready to construct a complete formalization of the example. As already hinted in the previous subsection, the identifiers of the message flows between the different participants will be used therefore. First of all, the *Find Bank* activity of the *Customer* contacts a *Broker* known at design time. Therefore it uses a π-calculus name $broker$ that contains a restricted name ch used as a response channel:

$$C3(c2, c4, broker) = (\mathbf{v}ch)c2.\overline{broker}\langle ch\rangle.ch(bank).\tau_{C3}.\overline{c4}\langle bank\rangle.\mathbf{0} \ .$$

The π-calculus processes of the broker have to be changed accordingly:

$$B1(b1, broker) = broker(ch).\tau_{B1}.\overline{b1}\langle ch\rangle.\mathbf{0}$$
$$B2(b1, b2, banklist) = b1(ch).banklist(bank).\overline{b2}\langle ch, bank\rangle.\mathbf{0}$$
$$B3(b2) = b2(ch, bank).\tau_{B3}.\overline{ch}\langle bank\rangle.\mathbf{0} \ .$$

This time, the *Broker* can not immediately start working because $B1$ is waiting for an external request on the name $broker$. The name $banklist$ in $B2$ is a pointer to a priority list filled with *Banks* sorted by interests. By reading from this list a pointer to the *Bank* with the lowest interest is returned (i.e. the first item of the list). Finally, the pointer to the *Bank* is returned via ch in $B3$.

The *Customer's* deferred choice between *Accept* and *Reject* processing is implemented by $C6$, while the bank is first contacted at $C5$ (*Credit Request*):

$$C5(c4, c6) = (\mathbf{v}req, acc, rej)c4(bank).\tau_{C5}.\overline{bank}\langle req, acc, rej\rangle.\overline{c6}\langle acc, rej\rangle.\mathbf{0}$$
$$C6(c6, c7, c8) = c6(acc, rej).\tau_{C6}.(acc.\overline{c7}.\mathbf{0} + rej.\overline{c8}.\mathbf{0}) \ .$$

The names acc and rej are used as environmental triggers for deciding the deferred choice. , whereas req represents the request. The complete choreography of the example is given by:

$$CHO(banklist) = (\mathbf{v}broker)(BROKER(broker, banklist) \ | \\ CUSTOMER(broker)) \ ,$$

with all the different *Banks* reachable over the $banklist$ pointer. Note that the actual management of the *Banks* such as adding or removing, is not contained inside the choreography. New *Banks* can register at runtime, whereas existing ones can withdraw or change their credit offers and conditions.
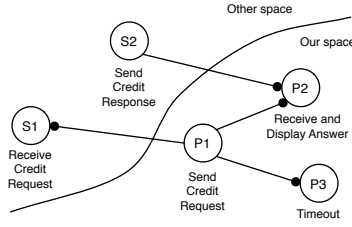
Figure 4: A distributed view of a service oriented architecture.

5 Application

The previous sections stated how the π-calculus can be used to model data, processes, and interactions found in service oriented architectures. Beside providing a uniform, unambiguous description of service oriented architectures, two further applications are of particular concern. The first one is execution and simulation, whereas the second one is reasoning.

Since the π-calculus is able to support data, orchestrations, as well as choreographies within one single algebra it can be seen as the least common denominator of service oriented architectures. Especially the direct support of dynamic binding by link passing mobility distinguishes the π-calculus from other approaches. By taking into account the results shown, execution environments for service oriented architectures should be based on the same concepts as the π-calculus. This minimal set of concepts already supports data, processes, and interactions as the core ingredients of service oriented architectures. Of course, the π-calculus is not applicable to business users or even most information technology experts. Thus, higher level constructs have to be introduced that can be broken down to π-calculus if required. For instance, a business process modeler uses the BPMN to model a business process that is then automatically mapped to π-calculus expressions for execution. Other results are more of a theoretical nature, for instance the representation of data types in the π-calculus. While a name can easily represent a pointer to some kind of data, such as an integer, the actual implementation has to be native for performance issues. Furthermore, the π-calculus describes a highly distributed representation of orchestrations and choreographies as shown in figure 4. In the figure, all activities of a choreography are shown as independent, distributed circles representing π-calculus processes. The processes use names to interact with each other, denoted by the lines between, where the dotted ends represent processes with the input prefix. While the activities of the right hand side belong inside *our* company, representing an orchestration, the left hand side activities belong to some other companies. Since the activities are all distributed and the link structure can be changed all the time, high flexibility is guaranteed. In a service oriented architecture, all activities are actually services reachable over *uniform resource locators* (URL) that can be deployed and interact as needed. The π-calculus provides a formal abstraction layer for such systems.

Another advantage of a formal representation of service oriented architectures are the

possibilities for reasoning. Especially in the context of open environments such as the Internet, where services are usually deployed and executed, a sound implementation is crucial. By using bisimulation techniques of the π-calculus [San93], orchestrations and choreographies can be checked before deployment and matchmaking of compatible services regarding the interaction behavior is possible. In [PW06] we have already shown how choreographies can be proved to be free of deadlocks and livelocks using a criterion called *Lazy Soundness*. An extension of this approach can be used to prove complete choreographies to be free of deadlocks and livelocks from the viewpoint of a certain orchestration. Thus, not only control flow dependencies are analyzed, but also interactions. Finally, the correctness of service implementations can be shown by proving the interaction equivalence between a service specification (i.e. an abstract orchestration) and a certain implementation.

6 Conclusion

In this paper we have shown how the π-calculus, an algebra for mobile systems, can be applied as a unified formal foundation for service oriented architectures. The unified formal representation of all key aspects of service oriented architectures — data, processes, and interactions — in one canonical minimal formal framework builds a foundation for further research and development. Yet recent standards like XLang (now superseded by BPEL4WS) [Mic01, BEA03] or WS-CDL [W3C04] claim to be based on the π-calculus. While they only took partial aspects of the π-calculus into consideration, recent research into service interaction patterns [BDtH05] identified the need for dynamic binding in service oriented environments. Until now, dynamic binding has been mostly neglected in scientific research. This paper highlighted the importance of the concepts found in the π-calculus for service oriented architectures.

References

[BDtH05]   Alistair Barros, Marlon Dumas, and Arthur ter Hofstede. Service Interaction Patterns. In W.M.P. van der Aalst, Boualem Benatallah, and Fabio Casati, editors, *Proceedings of the 3rd International Conference on Business Process Management, volume 3649 of LNCS*, pages 302–318, Berlin, 2005. Springer-Verlag.

[BEA03]   BEA Systems, IBM, Microsoft, SAP, Siebel Systems. *Business Process Execution Language for Web Services Version 1.1 (BPEL4WS)*, May 2003.

[BPM04]   BPMI.org. *Business Process Modeling Notation*, 1.0 edition, May 2004.

[BS05]   Lucas Bordeaux and Gwen Salaün. Using Process Algebra for Web Services: Early Results and Perspectives. In Ming-Chien Shan, Umeshwar Dayal, and Meichun Hsu, editors, *TES 2004, volume 3324 of LNCS*, pages 54–68, Berlin, 2005. Springer-Verlag.

[Bur00]   Steve Burbeck. The Tao of E-Business Services. Available at: `http://www-128.ibm.com/developerworks/library/ws-tao/`, 2000.

[CCMS01]   Erik Christensen, Francisco Curbera, Greg Meredith, and Weerawarana Sanjiva. *Web Service Description Language (WSDL) 1.1*. IBM, Microsoft, March 2001. W3C Note.

[DPW06]   Gero Decker, Frank Puhlmann, and Mathias Weske. Formalizing Service Interactions. In S. Dustdar, J.L. Fiadeiro, and A. Sheth, editors, *Proceedings of the 4th International Conference on Business Process Management (BPM 2006), volume 4102 of LNCS*, pages 414–419, Berlin, 2006. Springer Verlag.

[Mar03]   Axel Martens. On Compatibility of Web Services. *Petri Net Newsletter*, 65:12–20, 2003.

[Mar05]   Axel Martens. Analyzing Web Service based Business Processes. In Maura Cerioli, editor, *Proceedings of Intl. Conference on Fundamental Approaches to Software Engineering (FASE'05)*, volume 3442 of *Lecture Notes in Computer Science*. Springer-Verlag, April 2005.

[Mic01]   Microsoft. *XLang Web Services for Business Process Design*, 2001.

[MPW92]   Robin Milner, Joachim Parrow, and David Walker. A Calculus of Mobile Processes, Part I/II. *Information and Computation*, 100:1–77, September 1992.

[MRS05]   Peter Massuthe, Wolfgang Reisig, and Karsten Schmidt. An Operating Guideline Approach to the SOA. *Annals of Mathematics, Computing & Teleinformatics*, 1(3):35–43, 2005.

[OPW05]   Hagen Overdick, Frank Puhlmann, and Mathias Weske. Towards a Formal Model for Agile Service Discovery and Integration. In Kunal Verma, Amit Sheth, Michal Zaremba, and Christoph Bussler, editors, *Proceedings of the International Workshop on Dynamic Web Processes (DWP 2005)*, IBM technical report RC23822, Amsterdam, December 2005.

[Pet62]   Carl Adam Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für Instrumentelle Mathematik, Bonn, 1962.

[Puh06]   Frank Puhlmann. Why do we actually need the Pi-Calculus for Business Process Management? In W. Abramowicz and H. Mayr, editors, *9th International Conference on Business Information Systems (BIS 2006), volume P-85 of LNI*, pages 77–89, Bonn, 2006. Gesellschaft für Informatik.

[PW05]   Frank Puhlmann and Mathias Weske. Using the Pi-Calculus for Formalizing Workflow Patterns. In W.M.P. van der Aalst, Boualem Benatallah, and Fabio Casati, editors, *Proceedings of the 3rd International Conference on Business Process Management, volume 3649 of LNCS*, pages 153–168, Berlin, 2005. Springer-Verlag.

[PW06]   Frank Puhlmann and Mathias Weske. Investigations on Soundness Regarding Lazy Activities. In S. Dustdar, J.L. Fiadeiro, and A. Sheth, editors, *Proceedings of the 4th International Conference on Business Process Management (BPM 2006), volume 4102 of LNCS*, pages 145–160, Berlin, 2006. Springer Verlag.

[San93]   Davide Sangiorgi. A Theory of Bisimulation for the Pi-Calculus. In *CONCUR '93: Proceedings of the 4th International Conference on Concurrency Theory*, pages 127–142, Berlin, 1993. Springer-Verlag.

[SBS04]   Gwen Salaün, Lucas Bordeaux, and Marco Schaerf. Describing and Reasoning on Web Services using Process Algebra. In *ICWS '04: Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, page 43, Washington, DC, USA, 2004. IEEE Computer Society.

[vAtH03]   W. M. P. van der Aalst and A. H. M. ter Hofstede. YAWL: Yet Another Workflow Language (Revised version. Technical Report FIT-TR-2003-04, Queensland University of Technology, Brisbane, 2003.

[vAtHKB00] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. Technical Report BETA Working Paper Series, WP 47, Eindhoven University of Technology, 2000.

[vAvH02]   Wil van der Aalst and Kees van Hee. *Workflow Management*. MIT Press, 2002.

[vAW01]   W. M. P. van der Aalst and M. Weske. The P2P Approach to Interorganizational Workflow. In K.R. Dittrich, A. Geppert, and M.C. Norrie, editors, *Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE'01), volume 2068 of LNCS*, pages 140–156, Berlin, 2001. Springer-Verlag.

[W3C04]   W3C.org. *Web Service Choreography Description Language (WS-CDL)*, April 2004.