WS-CDL and Pi-Calculus

Paul Bouché

Seminar Business Process Management II
Hasso-Plattner-Institute for Software Systems Engineering
paul.bouche@hpi.uni-potsdam.de

Abstract. With the publishing and creation of the Web Services Description Language (WS-CDL) and the success of the Pi Calculus it has been stated that the former is based on the latter. A well reasoned answer for or against this claim is not to be found. Here an approach to a well reasoned answer is portrayed and criteria on how to evaluate this statement are presented. An investigation of the relationship of Pi Calculus and WS-CDL based on these criteria is carried out. Finally a conclusion based on the results of the investigation is drawn.

1 Introduction

A core motivation for creating the Web Services technologies and the Service Oriented Architecture (SOA) has been to fully automate business-to-business interaction in a cost effective and reliable way. In order to achieve this goal the need for an unambiguous and verifiable description of the involved interactions between the parties arose. The Web Services Description Language (WS-CDL) [5] was developed to satisfy this need.

As it has been published there has been a lot of discussion weather or not WS-CDL is based on Pi-Calculus [6]:

- "WS-CDL Has Sound Industrial and Mathematical Foundations" [7]
- "WS-CDL, being based on the Pi-Calculus [...]" [8]
- "WS-CDL is based on a formal model" [9]

There is not a well reasoned answer to be found. We would like to contribute to a well reasoned answer.

We will do this by first introducing the concepts of Pi Calculus and WS-CDL (section 2). Secondly criteria for evaluating the above statements and the relationship of WS-CDL and Pi Calculus are developed (section 3). Thirdly these are applied and investigated in section 4. Finally from the results a conclusion is drawn in section 5.

2 Preliminaries

In this section we will give a short introduction to Pi-Calculus and WS-CDL and their main concepts. We will first start to introduce Pi-Calculus and then shall present WS-CDL.

2.1 The Pi-Calculus and its main concepts

The Pi-Calculus is a formalism that was developed by Robin Milner, Joachim Parrow and David Walker. It was published among other papers in [6]. Milner introduces Pi-Calculus as "a calculus of communicating systems in which one can naturally express processes which have changing structure. Not only may the component agents of a system be arbitrarily linked, but a communication between neighbours may carry information which changes that linkage." [6, p.5] The motivation for creating Pi-Calculus was to be able to express this kind of concept, i.e. the changing structure and linkage of processes in a formal, yet simple and explicit way.

This concept is called *mobility* in Pi-Calculus. This mobility exists among the concept of a *process* or an *agent*. A process or agent represents a computational entity. Processes communicate *names* to other processes across links. A name represents a reference to either a link between processes, a piece of data or a variable.

A name has a *scope*, i.e. in Pi-Calculus it can be bound or restricted to certain processes and thus only these processes "know" that name. The scope of the name is changed when a name is sent to another process which did not know that name before. Computation is represented as the sending of names across links.

Changing linkage structure of the involved processes is formally expressed by the reduction rules as defined in [12]. This is then called evolution, i.e. the system of the involved processes evolves in the sense that the links and processes move in an abstract space of linked processes [2, slide 3a#3]. The control structures that are part of Pi-Calculus are parallelism, XOR-choice, recursion and if-then.

In formula 1 the Pi-Calculus grammar in its polyadic version is shown. In polyadic Pi-Calculus *several names at once* can be sent. We will now explain the semantics of this grammar informally by an example.

In figure 1 an example system of Pi-Calculus processes is depicted. This illustration is done as an informal flow graph of the processes and links. Here a process *A* has a link *b* to process *B* which is only known to both of them and *A* has a link *c* to process *C*. Process *A* will either send *five* to *B* over *b* or send the link *b* over *c* along with *five* to C which sends it over *b* to *B*.

In Formula 2 this behaviour is written in Pi-Calculus notation. First we see a name *SYS* defined as the parallelism (notated by the | operator) of *A*,*B* and *C* where the name *b* is bound to the processes *A* and *B*.

$$\begin{aligned} P &::== M \mid P|P \mid \boldsymbol{\nu} z\, P \mid !\,P \\ M &::== \mathbf{0} \mid \pi.P \mid M+M' \\ \pi &::== \bar{x}\langle \tilde{y} \rangle \mid x(\tilde{y}) \mid \tau \mid [x=y]\pi \end{aligned} \tag{1}$$

$$\begin{aligned} SYS &= (b)(A|B)|C \\ A &= (\boldsymbol{\nu}\ five)(\bar{b}\langle five\rangle.\mathbf{0} + \bar{c}\langle b, five\rangle.\mathbf{0}) \\ B &= b(d).\mathbf{0} \\ C &= c(l,m).\bar{l}\langle m\rangle.\mathbf{0} \end{aligned} \tag{2}$$

Secondly process *A* is defined as generating a new name *five* and behaving as described above; the choice is expressed by the + operator. Sending over a link is notated as an overlined occurrence of its name. A reception over a link is notated by the occurrence of its name.

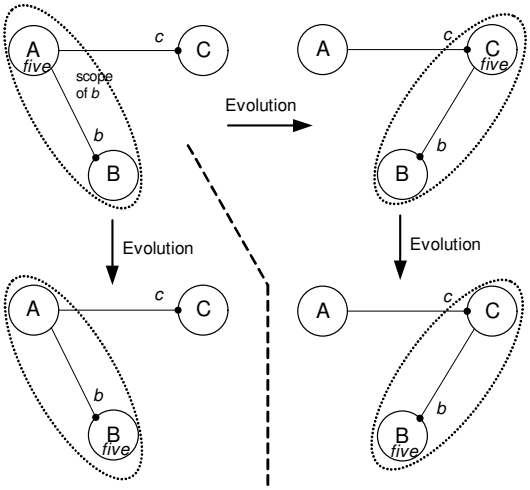

Fig. 1. Pi-Calculus Example of a System of Processes

2.2 WS-CDL and its main concepts

The WS-CDL specification [5] states the following summarization of WS-CDL: "The Web Services Choreography Description Language (WS-CDL) is an XML-based language that describes peer-to-peer collaborations of participants by defining, from a global viewpoint, their common and complementary observable behavior; where ordered message exchanges result in accomplishing a common business goal." So the perspective of WS-CDL is a multiparty perspective where only the common and observable activities of the processes are visible. The main purpose of defining a choreography is the common business goal of the involved parties in contrast to an orchestration where a single entity perspective is taken and only the private business goal of the orchestrating party is important. Once a choreography has been defined and been agreed to jointly, it can serve as a contract and a means by which each participant can generate their orchestration stubs and verify the conformance of the resulting real interactions.

In figure 2 WS-CDL and its concepts, parts and also the structure of the resulting XML documents are shown. For XML elements a new rectangle has been drawn, attributes are enumerated and parent-child relationship is visualized through rectangles being contained in one-another. We will explain the main concepts of WS-CDL now.

A *roleType* abstractly represents a role a party takes in a choreography. It is an abstraction from concrete behavior certain entities may exhibit. A roleType is constrained by a *relationshipType* which is an abstract representation of a relationship between the involved parties. All interaction in a choreography takes place between roleTypes. A *participantType* "groups together those parts of the observable behavior that must be implemented by the same logical entity or abstract organization" [5], represents a participant and is assigned to one or more roles. WS-CDL is typed and a type is modeled by an *informationType*. InformationTypes are referenced by *variables* and *tokens*. A token denotes a reference to an instance of an informationType.

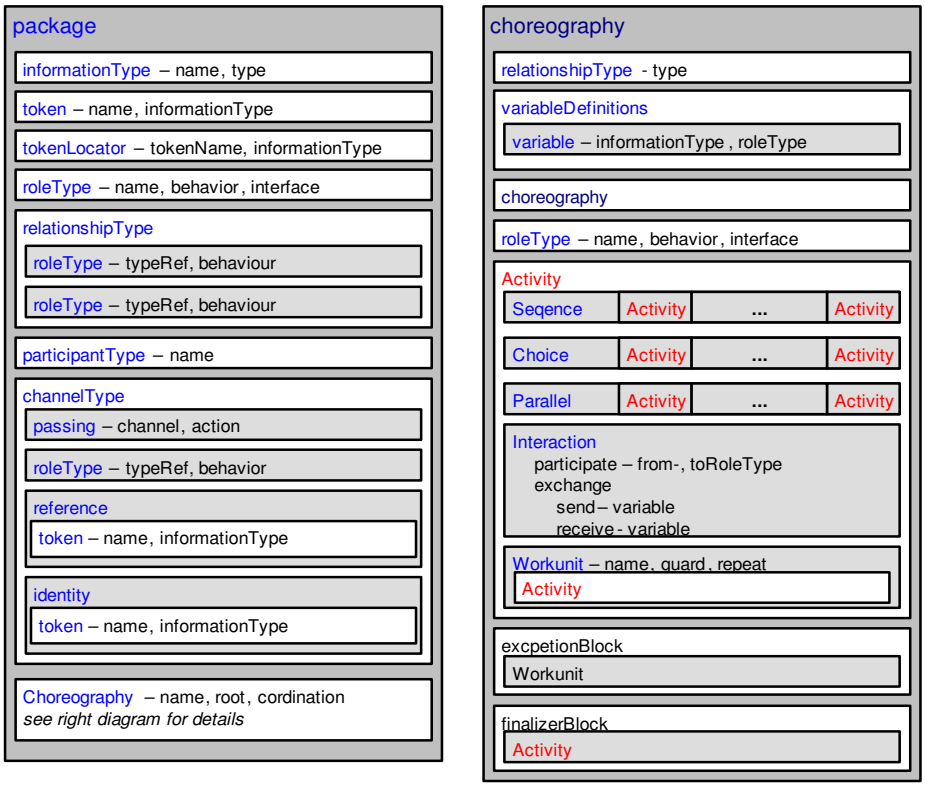

Fig. 2. WS-CDL at a glance

Tokens are used to correlate information and provide a means of identification for channelType instances. A *variable* resides at a certain roleType or is accessible by several or all roleTypes and contains information which can either be data sent during an interaction, a state within a roleType or a channelType instance. A *channelType* is the abstract representation of a channel between two or more involved parties. Over channels variables are sent or they are used for synchronization (where no actual values are sent). Thus channelType instances can be sent from one roleType to another. In order to correlate information and as a means to identify a session tokens are used.

The actual possible interactions between the parties are described in the activity part of the choreography element. An activity can either be a *sequence* of activities, a *parallelism* of activities, a *choice* between activities, an *interaction* or a *workunit*. Sequence, parallelism and choice have the usual semantics as control structures. An interaction actually models the actual exchange of information and observable behavior that takes place. During an interaction variables are exchanged or synchronization between roleTypes takes place. A workunit contains activities that may be needed for finalizing, rolling back, compensating or handling exceptions during a choreography. A workunit has a guard condition attached which either enables it or disables it de-

pending of the value of the variables specified in this condition[1]. A workunit may have a repetition condition.

The core concepts of WS-CDL are channelTypes, variables, interactions and guarded workunits. We will discuss these further in section 4.

3 Criteria for evaluating the relationship of Pi-Calculus and WS-CDL

In this section we want to develop the criteria which we will use in the following section to evaluate the statement "WS-CDL is based on Pi-Calculus" that were cited in the introduction and to analyze the relationship between Pi-Calculus and WS-CDL.

A general description of *based on* in the dictionary leads to the following synonyms and explanations: being founded on; to make, form or serve as a base for; built on; executed in; grounded on; rooted in; derived from; anchored in. These give us a general idea of the meaning of *based on*. More technically speaking the synonyms "being founded on" and "derived from" seem to clarify the meaning more. The words "derived from" allude to a software design concept: the concept of inheritance. These two associations lead us to the following illustration depicted in figure 3.

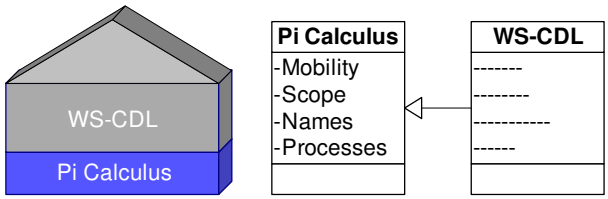

Fig. 3. Pi-Calculus is the basis for WS-CDL

The left-hand illustration shows the "WS-CDL house" and its alleged foundation the Pi-Calculus. The right-hand illustration shows a UML-class diagram where the class WS-CDL inherits from the class Pi-Calculus.

There are different understandings for inheritance in software design; yet in all is common that the inheriting class will inherit all functionality from the superclass but may extend it. Especially all the attributes or properties of the superclass the subclass is expected to have yet it may not behave exactly as the superclass.

When one thing is the foundation of another, let us call it building, it is expected that the building is held by the foundation and that all of the building is routed in the foundation.

Because of the statement we want to analyze we have to look from the direction of Pi-Calculus to WS-CDL: the other direction would not help us gain information concerning the given statement.

---

[1] The evaluation of a guard conditions is called matching. The specification is unclear about when this occurs. When several workunit's guard conditions are matched to true – how this concurrency is resolved is also left unclear unless these workunits are part of a choice ordering structure where the first workunit which matches to true is taken.

These considerations lead us to the following criteria which we will further use to evaluate the question at hand.

3.1 All concepts of Pi-Calculus should be found in WS-CDL

If WS-CDL is based on Pi-Calculus then all the concepts that are part of Pi-Calculus should be found in WS-CDL. If there is one concept in Pi-Calculus that does not exist in WS-CDL then we can conclude that this statement is not true for WS-CDL and Pi-Calculus.

We will investigate weather or not all the main concepts of Pi-Calculus can be expressed in WS-CDL and how concise this WS-CDL representation of a Pi-Calculus concept is. The investigation of this statement will be by example and by general discussion.

If it is not possible to represent all concepts of Pi-Calculus in WS-CDL as a weaker version of this criterion we will evaluate how many of the Pi-Calculus concept can be expressed of WS-CDL and if this is less than 50% we shall conclude that the weaker version of this criterion does not hold.

3.2 All Pi-Calculus systems of processes should be representable as WS-CDL choreographies

If WS-CDL finds its formal grounding in Pi-Calculus then all possible Pi-Calculus systems of processes should be representable as a WS-CDL choreography. This amounts to defining a mapping which maps any Pi-Calculus process notation to a WS-CDL choreography retaining if possible all of the original semantics.

Consequently to falsify this statement only one Pi-Calculus system of processes has to be found which is not representable as a WS-CDL choreography.

To provide a complete mapping function, if it is possible to do so, is out of the scope of this paper yet we will do partial investigation of this point.

3.3 All properties that are valid for Pi-Calculus should be valid for WS-CDL

If WS-CDL is based on Pi-Calculus then all properties and attributes that are valid for Pi-Calculus have to be valid for WS-CDL. Such properties include but are not limited to formulation and proveableness of bisimulation, bisimilarity, deadlock and liveness.

To show how these properties could be proved in WS-CDL or how if WS-CDL is rooted in Pi-Calculus the proof in Pi-Calculus can be transferred to WS-CDL is out of the scope of this paper. Yet we choose for our investigation a property of Pi-Calculus which has been shown in our context: that in Pi-Calculus all Service Interaction Patterns [3] are expressible [1]. This shall serve us as a property we want to investigate for WS-CDL. If in WS-CDL it is possible to express all the service interaction patterns then we have gained no information for the question at hand yet if there is one pattern which cannot be expressed in WS-CDL we have falsified the statement 3.3 and thus are able to deduce a conclusion.

4 Evaluation of WS-CDL with respect to Pi-Calculus

In this section we will investigate the criteria of the preceding section. In section 4.1 criterion 3.1 is evaluated by example and by further reasoning about the concepts of Pi-Calculus. The second criterion 3.2 is assessed in section 4.2 by general discussion. Finally in section 4.3 the last criterion from section 3 is investigated by trying to express the service interaction patterns [3] in WS-CDL.

4.1 Evaluation of the concepts of Pi-Calculus in WS-CDL

To evaluate the relationship of the concepts of Pi-Calculus and WS-CDL we will use an example choreography specified in the Business Process Modeling Language (BPMN) [4]. From this example we will analyze WS-CDL's "concept inheritance" from Pi-Calculus but will also do a general discussion of the concepts of Pi-Calculus.

We have chosen an example from the e-business domain, i.e. online hotel broking. The BPMN diagram of the example is shown in figure 4.

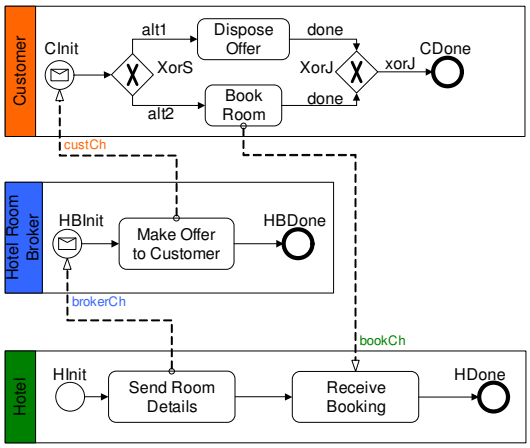

Fig. 4. Simple choreography example

A collaboration process is shown were each pool name represents a role a participant can take in an instance of this overall process description. The processes depicted represent abstract processes were only the public activities are visible. Hence this is a choreography description.

Three roles are shown: a hotel – wanting its spare rooms rented out, a hotel broker – broking spare hotel rooms to customers and a customer – wanting to book a room for a good price. There are biunique names added to those BPMN artifacts which are usually not named, e.g. "CInit", "alt1" etc. These are needed for the mapping to Pi-Calculus according to [2].

The hotel does not know any further customers; the hotel broker knows the hotel and the customer; the customer does not know any hotel but receives offers from the

hotel broker about free rooms and prices for those rooms. The interaction between the partners are as follows:

1. The hotel sends information about available rooms and how these rooms can be booked (how the hotel can be reached to book, i.e. booking channel – *bookCh*) to the hotel broker over the broker channel (*brokerCh*)
2. The hotel broker constructs an offer from this information and sends it along with the *bookCh* to the customer over the customer channel (*custCh*)
3. The customer receives the offer and either will dispose of it or will book a room at the hotel over the *bookCh*

4.1.2 Example in Pi-Calculus

We have mapped this choreography description to Pi-Calculus using the method developed in [2]. All BPMN flow objects are mapped to unique Pi-Calculus processes identifiers and each sequence flow to a unique name. The role names (Hotel, Hotel Broker, Customer) are mapped to a Pi-Calculus process where an index $i \in \mathbb{N}$ indicates a specific instance of this role.

The message flow elements (*brokerCh*, *custCh*, *bookCh*) are represented with the same name but with an index specific to the implementing instance of the associated role. The special set $\Sigma = \{brokerCh_x, custCh_y, bookCh_z \mid x, y, z \in \mathbb{N}\}$ denotes these names. In formula 3 the choreography is shown as a top-level Pi-Calculus process

$$CH = (\prod_{i=1}^{c} Cust_i \mid \prod_{i=1}^{b} Broker_i \mid \prod_{i=1}^{h} Hotel_i) \tag{3}$$

CH defines a process of *c* customers, *b* hotel room brokers and *h* hotels. The Hotel generates new private names $bookCh_i$ and $roomInf_i$ (available rooms and prices) and sends it to a freely choosen broker via the free name $brokerCh_x$. The hotel processes are shown in formula 4.

$$\begin{aligned}
&Hotel_i = (\boldsymbol{\nu}\, hInit, sendR, receiveB, bookCh_i, roomInf_i) \\
&\qquad (HInit_i \mid SendR_i \mid RecieveB_i \mid HDone_i) \\
&HInit_i = \tau_{HInit_i}.\overline{hInit_i}.\mathbf{0} \\
&SendR_i = hInit.\tau_{SendR_i}.\overline{brokerCh_x}\langle bookCh_i, roomInf_i\rangle.\overline{sendR}.\mathbf{0} \\
&ReceiveB_i = sendR.bookCh_i(booking).\tau_{ReceiveB_i}.\overline{receiveB}.\mathbf{0} \\
&HDone_i = receiveB.\tau_{HDone_i}.\mathbf{0}
\end{aligned} \tag{4}$$

The broker receives the $bookCh_z$ and *roomInf* from one of the hotels over its corresponding name $brokerCh_i$. and sends the private name *offer* along with $bookCh_z$ to one of its customers $custCh_y$ (formula 5):

$$\begin{aligned}
&Broker_i = (\boldsymbol{\nu}\, hbInit, makeO, offer)(HBInit_i \mid MakeO_i \mid HBDone_i) \\
&HBInit_i = brokerCh_i(bookCh_z, roomInf).\tau_{HBInit_i}.\overline{hbInit}_i\langle bookCh_z, roomInf\rangle.\mathbf{0} \\
&MakeO_i = hbInit(bookCh_z, roomInf).\tau_{MakeO_i}.\overline{custCh_y}\langle bookCh_z, offer\rangle.\overline{makeO}.\mathbf{0} \\
&HBDone_i = makeO.\tau_{HBDone_i}.\mathbf{0}
\end{aligned} \tag{5}$$

A customer receives the names *offer* and $bookCh_z$ and either finishes or sends the name *booking* over the name $bookCh_z$.

$$
\begin{aligned}
&Cust_i = (\nu\, cInit, alt1, alt2, xorJ, booking_i) \\
&\qquad (CInit_i \mid XorS_i \mid XorJ_i \mid DispO_i \mid BookR_i \mid CDone_i) \\
&CInit_i = custCh_i(bookCh_z, offer).\tau_{CInit_i}.\overline{cInit}_i\langle offer, bookCh_z\rangle.\mathbf{0} \\
&XorS_i = cInit(offer, bookCh_z).\tau_{XorS_i}. \\
&\qquad (\overline{alt1}\langle offer, bookCh_z\rangle.\mathbf{0} + \overline{alt2}\langle offer, bookCh_z\rangle.\mathbf{0}) \\
&DispO_i = alt1(offer, bookCh_z).\tau_{DispO_i}.\overline{done}.\mathbf{0} \\
&BooR = alt2(offer, bookCh_z).\tau_{BookR_i}.\overline{bookCh_z}\langle booking\rangle.\overline{done}.\mathbf{0} \\
&XorJ_i = done.\tau\, XorJ_i.\overline{xorJ}.\mathbf{0} \\
&CDone_i = xorJ.\tau CDone_i.\mathbf{0}
\end{aligned}
\tag{6}
$$

We observe the following about the Pi-Calculus choreography description: the role names are expressed as an arbitrary indexed number of processes with the same name[2], there are associated free indexed names $brokerCh_i$, $custCh_i$ and private names $bookCh_i$ whose scope is extruded [6, p.15], the private names *booking*, $roomInf_i$ and *offer* are extruded as well, and the corelation of the "right" *bookCh* and *roomInf* are inherent in polyadic Pi-Calculus.

4.1.3 Example in WS-CDL and discussion of Pi-Calculus concepts in WS-CDL

In this section we will express the choreography example from the previous section in WS-CDL. Rather than to converse about the full XML document in detail, which can be found in appendix I, we will discuss crucial points of the WS-CDL representation of the choreography and how this relates to the concepts of Pi-Calculus.

Pool names are mapped to `roleType` names: Hotel, Broker and Customer. These represent the role any involved party may act in and thus abstracts from the actual involved number of parties. The indexed Pi-Calculus process names are thus represented.

The content of the exchanged messages is typed by `informationTypes`: `RefT`, `OfferT`, `BookingT`, `RoomInfoT`. `Tokens` are defined for later reference and are of the type `RefT`. They are used to among other things to correlate information during exchanges. In Pi-Calculus there are no types. Information correlation is implicit in Pi-Calculus but is made explicit in WS-CDL through tokens.

The message flow of the BPMN diagram is modeled by three channelTypes: `brokerChT`, `bookChT` and `custChT`. In order to pass a `channelType` instance over another the `passing` element is specified expressing which `channelType` is passed. Thus the mobility concept of Pi-Calculus is represented.

Variables for channel instances and containing data (including the information a token references to) are defined. In Pi-Calculus there is no distinction between data, links and variables – this is all represented as a name. Thus the concept of a name is only in part captured in WS-CDL. The `roleType` attribute of variable specifies where the variables "resides", i.e. where it is visible. This can be thought of as the representation of the Pi-Calculus scope of a name. Yet in order to express that initially a name is private and its scope is extruded using the same *syntactical* name three variables were specified in WS-CDL, i.e. the private name *bookCh* is expressed in the WS-

[2] These processes represent all the possible potential participant that can act in any of the involved roles: customer, hotel room broker and hotel.

CDL variables: `bookCh`, `bookCh@Broker`, `bookCh@Cust`. If the same name had been used, the variable would have been in the visibility of all roles [5, §§ 5.2, 6.2.3]. In Pi-Calculus the syntactical name in a process might be changed during evolution using substitution [6, p.10] yet in the initial specification of the processes the syntactical name can be the same, not so in WS-CDL.

The `interaction` elements contain the information exchanges that may occur. Variables are sent and received during an exchange within an interaction. The WS-CDL specification states that several such exchanges may happen during an interaction. Thus the polyadic Pi-Calculus concept of sending several names at once is only in part captured[3].

The control structures in WS-CDL are choice, parallelism, sequence, loop and if-then. The last two are only expressible in a workunit. Therefore all of the Pi-Calculus control structures are expressible in WS-CDL. Recursion is not directly expressible in WS-CDL but recursion can always be transformed to one or several loops but this might require some effort.

The exact concept of a process in Pi-Calculus cannot be captured directly, it can either be expressed as roleType, yet this is only feasible in the case of a set of processes which are indexed and carry the same "label", i.e. $Hotel_i$ or as a participantType but then for this participantType a role as to be modeled because a participantType cannot interact in WS-CDL.

We summarize our investigation in table 1.

Table 1. Summary of the investigation of Pi-Calculus concepts in WS-CDL

Pi-Calculus concept	**Expressible in WS-CDL?**
Process	Almost fully
Name	Partially
Scope of a name	Partially
Mobility	Fully
Control structures	Almost Fully
Polyadic extension	Almost fully[3]

4.2. Investigations on a mapping function

In this section we will investigate on a possible mapping function from a given Pi-Calculus process to a valid WS-CDL document.

A complete mapping function would have to map the Pi-Calculus grammar to a WS-CDL grammar or template document and in mapping to a document realize the semantical reduction rules for Pi-Calculus [12]. Thus the syntax and the semantic of any given Pi-Calculus process definition has to be considered.

In investigating the statement "WS-CDL is based on the Pi-Calculus" it is at first obvious that along with such a statement the mapping function we are discussing should be provided. A missing of such a function indicates that the statement may not

[3] It is unclear from the specification of WS-CDL weather or not there can be several request-exchange elements within one interaction element. From the WS-CDL reference implementation [4] and the examples we glean that only one is allowed, hence the conclusion.

be true or not well grounded. In deed we were not able to find any public official or unofficial document which contains a mapping function from Pi-Calculus to WS-CDL.

We will try to informally specify a general idea on how to construct a mapping from Pi-Calculus to WS-CDL and will indicate from the difficulties we face how possible a mapping is.

In a first step the top-level[4] (let us call it level 0) process of the given process definition has to be determined. This is in our examples: *SYS* (section 2.1) and *CH* (section 4.1.2). The name of this process can serve as the name for the root choreography in WS-CDL.

The second step involves mapping the next level process names (those referenced in the definition of the former), level 1, to either roleTypes and participants or just roleTypes. Indexed processes names such as $Hotel_i$ map to a roleType with the name of the process without the index such as *Hotel*. Unindexed enumerated process names are mapped to new participantTypes with corresponding new roleTypes. The semantics of the remaining process names (level 2,..,n) has to be analyzed to decide weather to map them to roleType / participantTypes or not.

In a third step Pi-Calculus names have to be mapped to either data variables, channelType instances or state capturing variables. Along with these the corresponding informationTypes, tokens and tokenLocators have to be modeled. The decision to map a name to a data variable or channelType instance is not trivial and requires knowledge of the "Pi-Calculus process modeler" or the intended meaning of the process definition is known. The defined sendings and receptions of names are mapped to interaction elements with corresponding exchange elements including the control structures of the involved processes. The recursions have to be analyzed through recursion trees etc. in order to express them in iterative form, i.e. loops which can be expressed in WS-CDL through workunits.

We indicate that to construct a complete mapping function several issues have to be resolved which are non-trivial, i.e. mapping 2-nd,...,n-th level processes to roleType etc, names to variables or channels and resolving multiple recursion into iterations. In a full formal investigation it could show to be impossible to do this for certain Pi-Calculus process systems at least for the former two issues.

4.3 Investigation on the support of Service Interaction Patterns in WS-CDL

In this section we will investigate if all of the Service Interaction Patterns as described in [3] are expressible in WS-CDL. We will use the order in which they appear in [3].

Pattern 1-3: Send, Receive, Send/Receive. These pattern are successfully expressed through a single interaction containing an exchange element with the attribute action set to request or respond or two exchange elements with request and respond sent over a channel which has an identity element defined.

---

[4] This already presupposes a structure or hierarchy of the processes where no such thing is part of the Pi-Calculus. We note that without such a presupposition it is not possible to construct a mapping function.

The next set of patterns involves an arbitrary number of parties to interact with one transmission each: *Pattern 4: Racing Incoming Messages*, *Pattern 5: One-to-many send*, *Pattern 6: One-from-many receive* and *Pattern 7: One-to-many send/receive*. The support of these patterns in WS-CDL depends on how one interprets a roleType. A roleType may stand for an arbitrary number of parties of the same "type", i.e. for the roleType "Hotel" the "Best Western Inn", "Hilton" etc. or it may represent a single party. In the case of pattern 5 our example of section 4.1 may be interpreted as an instance of this pattern but only if for each instance of the process model the hotel broker is the same and the customer is different. Yet in a case where a large number of receiving parties have to be enumerated the WS-CDL document would grow larger and larger. It is possible to express but may become unfeasible to do. In Pi Calculus all instances of these patterns are feasible to express.

Pattern 8 to 10 are of the type where an arbitrary number of transmissions between the parties occurs: *Pattern 8*: *Multi-responses*, *Pattern 9*: *Contingent Requests* and *Pattern 10*: *Atomic multicast notification*. Patterns 8 and 9 are supported through workunits with appropriate guards set. Pattern 10 is supported through the fault handling and exceptionBlock workunits which realize the transactional nature of this pattern.

The last set of patterns 11 to 13 is concerned with routing: *Pattern 11: Request with referral*, *Pattern 12: Relayed Request*, *Pattern 13: Dynamic Routing*. Pattern 11 is successfully captured by specifying a channelType with the appropriate passing attributes. Pattern 12 is expressible as well with similar constructs. The description of pattern 13 is: "A request is required to be routed to several parties based on a routing condition. The routing order is flexible and more than one party can be activated to receive a request. When the parties that were issued the request have completed, the next set of parties are passed the request." Routing can be dynamic. "The set of parties though which the request should circulate might not be known in advance. The specification of ordering should support service/role late binding." This pattern cannot be expressed at all in WS-CDL because current constructs of WS-CDL cannot be used to realize the requirements of this pattern.

We summarize out investigation in table 2.

Table 2. Support of Service Interaction Patterns [3] in WS-CDL

Pattern	**Expressible in WS-CDL?**
Send, Receive, Send / Receive	Fully
Racing Incoming Messages, One-to-many send, One-from--many receive, One-to-many send / receive	Almost Fully Issues: all involved parties have to be enumerated and (statically) linked
Multi-responses, Contingent Requests, Atomic multicast notification	Fully
Request with referral, Relayed Request	Fully
Dynamic Routing	Not expressible

5 Conclusion

Based on the criteria developed in this paper for evaluating the relationship of Pi Calculus and WS-CDL in section 3 we conclude the following.

Not all concepts of Pi Calculus are found or are expressible in WS-CDL. The concept of a name in Pi Calculus is not expressible in WS-CDL because WS-CDL differentiates between data, variables and channels. The scope of a name is not expressible in WS-CDL because for one private name whose scope is extruded representations of it have to be specified at all receiving roleTypes.

A mapping function from Pi Calculus to WS-CDL is not specified. The missing of it indicates that it might not be possible to state it. A mapping function involves the non-trivial problems of mapping Pi Calculus names to either data or channel variables and of reducing possible multiple recursion to iterations (loops).

Pi Calculus supports all service interaction patterns but WS-CDL does not support pattern 13, dynamic routing, and may not support the multilateral[5] interaction patterns. The properties of provableness of deadlock, livelock etc. are not expressed in the WS-CDL specification. No formal treatment of these properties with respect to WS-CDL is published. We assume it does not exist at the date of this writing.

Based on these results of our investigation we conclude that the statement "WS-CDL is based on Pi Calculus" is false, i.e. WS-CDL is not based on Pi Calculus. Pi-Calculus and WS-CDL have connections on different levels and some concepts of WS-CDL may have been inspired by Pi Calculus. WS-CDL and Pi Calculus are of different abstraction levels and domains therefore a comparison seems unfitting yet we did not associate the two but claim about the association of the two.

Concerning the statement "WS-CDL is based on a formal model" we conclude that currently this formal model is at a very early stage in development as seen in [14, 15] but seems not to have been there a priori. Thus WS-CDL was hardly based on a formal model because this model did not exist at the time of creation of WS-CDL. It may become based on a formal model in the future.

The statement "WS-CDL Has Sound Industrial and Mathematical Foundations" is also false in the light of the presented results.

It remains future work on how Pi Calculus can be used to specify choreographies. The global calculus that is suggested in [14] makes connections between parties explicit whereas in Pi Calculus these are implicit through name matching.

[5] Multilateral [3] means interaction of an arbitrary number of parties greater than 2.

References

1. G. Decker: Formalizing Service Interactions, seminar paper BPM II, unpublished, Hasso-Plattner-Institute, Potsdam, Germany, February 2006
2. F. Puhlmann: Introduction to the Pi-Calculus, Lecture Slides BPM II, unpublished, Hasso-Plattner-Institute, Potsdam, Germany, February 2006
3. A. Barros, M. Dumas and A. ter Hofstede: Service Interaction Patterns: Towards a Reference Framework for Service-based Business Process Interconnection. Technical Report FIT-TR-2005-02, Faculty of Information Technology, Queensland University of Technology, Brisbane, Australia, March 2005.
4. S. White (Editor), Business Process Management Initiative: Business Process Modeling Notation, Specification Document V1.0 – May 3rd 2004, bpmi.org, May 2005
5. N. Kavantzas, D. Burdett, G. Ritzinger, et. al. (Editors), World Wide Web Consortium (W3C): Web Services Choreography Description Language Version 1.0, W3C Candidate Recommendation, *work in progress*, Nov 9th 2005
6. R. Milner, J. Parrow and D. Walker: A Calculus of Mobile Processes - Part I, LFCS Report 89-85, University of Edinburgh, June 1989
7. World Wide Web Consortium: World Wide Web Consortium Publishes First Public Working Draft of Web Services Choreography Description Language 1.0, Press Release Note, Internet, April 2004, http://www.w3.org/2004/04/wschor-pressrelease
8. S. Ross-Talbot: Orchestration and Choreography: Standards, Tools and Technologies for Distributed Workflows, unpublished, available at http://www.nettab.org/2005/docs/NETTAB2005_Ross-TalbotOral.pdf, Pi-Calculus4 Technology, London, UK and W3C, Geneva, Switzerland, October 2005
9. N. Kavantzas: Aggregating Web Services: Choreography and WS-CDL, Oracle Corporation, April 2005
10. R. Milner: Communication and Concurrency, Prentice Hall, 1989
11. R. Milner, J. Parrow and D. Walker: A Calculus of Mobile Processes - Part II, LFCS Report 89-85, University of Edinburgh, June 1989
12. Davide Sangiorgi, David Walker: The pi-calculus: a Theory of Mobile Processes, INRIA Sophia,University of Oxford, Cambridge University Press, 2001, ISBN 0-521-78177-9
13. Pi4SOA, Pi4 Technologies Ltd. : WS-CDL Tool Suite, http://sourceforge.net/projects/pi4soa/, Nov. 2005
14. Honda, Yoshida, et. al.: A Theoretical Basis of Communication-Centred Concurrent Programming, unpublished, W3C, University of London, *work in progress*, Nov 2005 http://lists.w3.org/Archives/Public/public-ws-chor/2005Nov/att-0015/part1_Nov25.pdf
15. N. Kavantzas: Aggregating Web Services: Choreography and WS-CDL, presentation, Oracle, April 2004, http://www.oracle.com/technology/tech/webservices/htdocs/spec/WS-CDL-April2004.pdf

Appendix I

```
<?xml version="1.0" encoding="UTF-8"?>
<package xmlns="http://www.w3.org/2005/10/cdl" xmlns:xsd="http://www.w3.org/2001/XMLSchema" au-
thor="paul.bouche" name="BMPN_Example" targetNamespace="bpt.hpi.uni-potsdam.de" version="1.0">
  <description type="documentation">
    Illustrating Example
  </description>
  <informationType name="RefT" type="xsd:string"/>
  <informationType name="OfferT" type="Offer"/>
  <informationType name="BookingT" type="Booking"/>
  <informationType name="RoomInfoT" type="RoomInfo"/>
  <token informationType="RefT" name="brokerChRef"/>
  <token informationType="RefT" name="bookChRef"/>
  <token informationType="RefT" name="custChRef"/>
  <token informationType="RefT" name="sesID"/>
  <token informationType="RefT" name="offerID"/>
  <token informationType="RefT" name="bookID"/>
  <tokenLocator informationType="BookingT" query="/BO/bookID" tokenName="bookID"/>
  <tokenLocator informationType="RoomInfoT" query="/Exp/Date" tokenName="sesID"/>
  <roleType name="Hotel">
    <behavior interface="IWebService" name="hotelBehaviour"/>
  </roleType>
  <roleType name="HotelRoomBroker">
    <behavior interface="IWebService" name="brokerBehaviour"/>
  </roleType>
  <roleType name="Customer">
    <behavior interface="IWebService" name="customerBehaviour"/>
  </roleType>
  <relationshipType name="Hotel_Broker_Rel">
    <roleType behavior="hotelBehaviour" typeRef="Hotel"/>
    <roleType behavior="brokerBehaviour" typeRef="HotelRoomBroker"/>
  </relationshipType>
  <relationshipType name="Broker_Cust_Rel">
    <roleType behavior="brokerBehaviour" typeRef="HotelRoomBroker"/>
    <roleType behavior="customerBehaviour" typeRef="Customer"/>
  </relationshipType>
  <relationshipType name="Cust_Hotel_Rel">
    <roleType behavior="customerBehaviour" typeRef="Customer"/>
    <roleType behavior="hotelBehaviour" typeRef="Hotel"/>
  </relationshipType>
  <channelType action="request" name="brokerChT">
    <passing action="request" channel="bookChT"/>
    <roleType typeRef="HotelRoomBroker"/>
    <reference>
      <token name="brokerChRef"/>
    </reference>
    <identity type="primary">
      <token name="sesID"/>
    </identity>
  </channelType>
  <channelType action="request" name="bookChT">
    <roleType typeRef="Hotel"/>
    <reference>
      <token name="brokerChRef"/>
    </reference>
    <identity type="primary">
      <token name="bookID"/>
    </identity>
  </channelType>
  <channelType action="request" name="custChT">
    <passing action="request" channel="bookChT"/>
    <roleType typeRef="Customer"/>
    <reference>
      <token name="custChRef"/>
    </reference>
    <identity type="primary">
      <token name="offerID"/>
    </identity>
  </channelType>
  <choreography name="BPMN_Example" root="true">
    <relationship type="Hotel_Broker_Rel"/>
    <relationship type="Broker_Cust_Rel"/>
    <relationship type="Cust_Hotel_Rel"/>
    <variableDefinitions>
```

```
      <variable informationType="RoomInfoT" name="roomInf" roleTypes="Hotel"/>
      <variable informationType="RoomInfoT" name="roomInf@Broker" roleTypes="HotelRoomBroker"/>
      <variable channelType="bookChT" name="bookCh" roleTypes="Hotel"/>
      <variable channelType="bookChT" name="bookCh@Broker" roleTypes="HotelRoomBroker"/>
      <variable channelType="bookChT" name="bookCh@Cust" roleTypes="Customer"/>
      <variable channelType="custChT" name="custCh" roleTypes="HotelRoomBroker Customer"/>
      <variable channelType="brokerChT" name="brokerCh" roleTypes="Hotel HotelRoomBroker"/>
      <variable informationType="OfferT" name="offer" roleTypes="HotelRoomBroker"/>
      <variable informationType="OfferT" name="offer@Cust" roleTypes="Customer"/>
      <variable informationType="BookingT" name="booking" roleTypes="Customer"/>
      <variable informationType="BookingT" name="booking@Hotel" roleTypes="Hotel"/>
    </variableDefinitions>
    <sequence>
      <sequence>
        <description type="documentation">
          brokerCh&lt;bookCh,roomInf&gt;
        </description>
        <interaction channelVariable="brokerCh" name="sendBookCh" operation="receiveBookCh">
          <participate fromRoleTypeRef="Hotel" relationshipType="Hotel_Broker_Rel" toRole-
TypeRef="HotelRoomBroker"/>
          <exchange action="request" channelType="bookChT" name="transmitCh">
            <send variable="cdl:getVariable('bookCh','','')"/>
            <receive variable="cdl:getVariable('bookCh@Broker','','')"/>
          </exchange>
        </interaction>
        <interaction channelVariable="brokerCh" name="sendRoomInf" operation="receiveRoomInf">
          <participate fromRoleTypeRef="Hotel" relationshipType="Hotel_Broker_Rel" toRole-
TypeRef="HotelRoomBroker"/>
          <exchange action="request" informationType="RoomInfoT" name="transmitRoomInf">
            <send variable="cdl:getVariable('roomInf','','')"/>
            <receive variable="cdl:getVariable('roomInf@Broker','','')"/>
          </exchange>
        </interaction>
      </sequence>
      <sequence>
        <description type="documentation">
          custCh&lt;bookCh,offer&gt;
        </description>
        <interaction channelVariable="custCh" name="sendBookCh" operation="receiveBookCh">
          <participate fromRoleTypeRef="HotelRoomBroker" relationshipType="Broker_Cust_Rel"
toRoleTypeRef="Customer"/>
          <exchange action="request" channelType="bookChT" name="transmitCh">
            <send variable="cdl:getVariable('bookCh@Broker','','')"/>
            <receive variable="cdl:getVariable('bookCh@Cust','','')"/>
          </exchange>
        </interaction>
        <interaction channelVariable="custCh" name="sendOffer" operation="receiveOffer">
          <participate fromRoleTypeRef="HotelRoomBroker" relationshipType="Broker_Cust_Rel"
toRoleTypeRef="Customer"/>
          <exchange action="request" name="transmitOffer">
            <send variable="cdl:getVariable('offer','','')"/>
            <receive variable="cdl:getVariable('offer@Cust','','')"/>
          </exchange>
        </interaction>
      </sequence>
      <choice>
        <noAction roleType="Customer">
          <description type="documentation">
            Dispose Offer
          </description>
        </noAction>
        <interaction channelVariable="bookCh@Cust" name="bookCh&lt;booking&gt;" opera-
tion="receiveBooking">
          <participate fromRoleTypeRef="Customer" relationshipType="Cust_Hotel_Rel" toRole-
TypeRef="Hotel"/>
          <exchange action="request" informationType="BookingT" name="bookRoom">
            <send variable="cdl:getVariable('booking','','')"/>
            <receive variable="cdl:getVariable('booking@Hotel','','')"/>
          </exchange>
        </interaction>
      </choice>
    </sequence>
  </choreography>
</package>
```