Seminar Reader

Business Process Management
Winter Term 2005/2006

Editors

Frank Puhlmann
Hilmar Schuschel
Mathias Weske

Contact
Business Process Technology Group
Hasso Plattner Institute for IT Systems Engineering
at the University of Potsdam
P.O. Box 90 04 60
D-14440 Potsdam, Germany

Preface

This seminar reader contains the papers of the seminar part of the lecture Business Process Management II, held in the winter term 2005/2006 at the Hasso-Plattner-Institute. Master students of IT systems engineering discussed and researched current topics in the area of business process management. Each paper contained in this seminar reader was accompanied by a conference-style talk.

We had a broad range of topics, ranging from theoretical foundations for the representation of choreographies and orchestrations, over process planning up to methodologies and variant-rich processes. Almost all topics are grounded in lectures given beforehand, so the students had a very broad foundation and talks and papers dived right into challenging questions. The lecture had been divided into four major parts. Part one introduced advanced concepts of business process management based on service-oriented architectures. Part two discussed a formal foundation, the pi-calculus. Part three covered advanced service composition in detail. Part four concluded with an introduction to the European research project Adaptive Services Grid (ASG) that investigates dynamic service composition and enactment.

We would like to thank the students that attended our lecture and contributed to this seminar reader – it was a very interesting and inspiring experience! Further acknowledgements go to Harald Meyer and Guido Laures for giving additional lectures as well as Arnd Schnieders for supervising some of the topics.

Frank Puhlmann, Hilmar Schuschel, Mathias Weske
Potsdam, February 2006

Table of Contents

Concepts for a π-Calculus Simulator

Anja Bog

Hasso Plattner Institute for Software Systems Engineering
at the University of Potsdam,
D-14482 Potsdam, Germany
Anja.Bog@hpi.uni-potsdam.de

Abstract. This paper elaborates a possible data structure which can be used in the implementation of a simulator for the π-calculus. The data structure which is proposed is a tree structure and a mapping of the different constructs of the π-calculus to structure elements of the tree is given. Furthermore execution rules are defined in such a way that the resulting simulator behaves in the same way as the reduction semantics of the π-calculus.

1 Introduction

The π-calculus is a formal language for concurrent, communicating and mobile systems. It provides a framework for the representation, analysis, simulation and verification of these systems. Concurrent processes within a system communicate with each other via channels sending and receiving messages. The contents of these messages are names that can also represent channels and can therefore be used for further communication. Thus new links between active processes are dynamically created which constantly change the circuits in the system and thereby make it mobile.

A number of tools which address the π-calculus already exist. Some of them are for example *Another Bisimulation Checker*[5], *Open Bisimulation Checker*[6] and *Mobility Workbench*[4]. However those tools are mainly intended for bisimilarity and model checking, which means that instead of watching how a system may behave internally, they check if two systems have the same observable behavior. To some extent these tools are able to do step by step executions or show execution traces, but they do not offer visual output other than command line output.

The motivation for investigating concepts of a π-calculus simulator is the exploration of the internal behavior of π-calculus processes. The simulator should interact with the user if several ways of further execution are available and it should also provide the user with expedient visual output for a deeper understanding of the systems behavior. Hence the requirements for a simulator can be deduced. A step by step execution of the process definitions is wanted and in each step a visualization of the current systems state which shows the interacting agents and their communication channels shall be available. Furthermore some abstraction levels may be defined that tell the simulator which of these

processes the user is interested in seeing the internal behavior of and which of them shall be handled as black box systems. This might be useful if for example data structures (e.g. stacks, memory cells) are used that consist of several agents that have to communicate with each other, but they are not the point of interest, so the data structures might be presented as abstract processes without internal behavior in the visualization. Handling some processes as black boxes also demands of the simulator to be able to make choices internally and to some extent non deterministically if several ways of execution are possible.

Before a current state of a π-calculus system might be presented to the user, the simulator has to be able to execute the process definitions. This execution will be the scope of this paper. A data structure and associated execution rules will be explored. It is not the focus to discuss how the proposed data structure might be handled during visualization.

Chapter 2 will give a short overview of two ways the execution of the π-calculus might be handled and shows some advantages and disadvantages of the different approaches. For the following chapters one of these approaches will be chosen and further investigated. Chapter 3 defines the data structure the π-calculus process definitions can be translated into and also specifies the execution rules for this data structure in a such a way that the later execution of the processes shows the same behavior as using the reduction semantics of the π-calculus. In chapter 4 an example will be introduced that is translated into the proposed structure and it is shown how it will be processed.

2 Handling the π-Calculus

The π-calculus grammar can be seen as a Domain Specific Language (DSL). A DSL is a language that is targeted at a particular kind of problem, rather than a general purpose language that is aimed at any kind of software problem. In the case of the π-calculus the DSL is intended to describe the behavior of communicating and mobile systems. Two different approaches of handling DSL's will be briefly discussed in this chapter.

2.1 Embedded Domain Specific Languages

Instead of defining a new language, the DSL is weaved into an existing general purpose language without changing this language. Accordingly the elements of the DSL have to be expressed in the syntax of the base language. A disadvantage which evolves from this is that the expressiveness of the base language might narrow the expressiveness of the DSL and workarounds have to be found to avoid this. For this reason the language that is going to be used as a base language has to fulfill some premises[1]. On the one hand it should provide for a lightweight syntax that stays out of the way when defining problems in the DSL, so complex twists for expressing certain DSL elements can be avoided. On the other hand it should offer some control to create new syntax if for example no workaround can be found to express a certain element in the base language. An advantage

of using this approach is that the compiler or interpreter of the base language can be used for the execution. Languages that are appropriate for using as base languages are for example Ruby, Lisp or Smalltalk.

Concerning the π-calculus Simulator this approach might not be as useful because process definitions can become very cryptic if one has to care about not violating the syntax of the base language. Furthermore intermediate results like step by step execution of the processes, user interaction if several choices are available and visualization of the systems current state after each step are wanted. Providing intermediate results using embedded DSL's might be hard to supply because this would have to be specified by the user as well or an intermediate step has to be taken which inserts the processing steps in the process definitions of the user. A solution for theses problems might be to include another preprocessor or macro processing step. The advantages which arise from this is that the user now can specify his process definition in a π-calculus like syntax and the preprocessor will translate these definitions into the constructs of the base language and also inserts the execution rules for the intermediate results. But there is still one disadvantage which also resides in this solution, namely that errors in the syntax of the user specified processes are not detected immediately but might cause wrong syntax in the base language which will be detected by the base languages compiler/interpreter or it causes strange behavior during execution. In any case the user will be confronted with error messages on the level of the base language and not on domain level.

2.2 External Domain Specific Languages

This approach is similar to the approach of using a preprocessor beforehand for embedded DSL's with the difference, that the preprocessor is integrated in the interpreter. Therefore type checking on the domain level becomes available and the user can be provided with domain specific feedback on the errors. For a dedicated interpreter a parser is needed which translates the input into a usable data structure for execution. Many tools exist which are able to generate such a parser based on a grammar specified for the DSL. The decision which one to use may result from the data structure that will be used during further execution and to what extend a certain tool assists in building that structure. The advantage of this approach is that the input is independent of the programming language used but solely depends on the grammar specified. As a result an already existing input style might be used and the parser can be tailored to this format. This is especially useful if there already exists a tool that helps the user to model processes and whose output are π-calculus process definitions. Using a general input style also presents the possibility of integrating existing tools like for example the Mobility Workbench to provide additional functionality. Furthermore the production of intermediate results has to be implemented in the interpreter anyway therefore no restrictions are applied here. The disadvantage of this approach is that the whole interpreter aside from the parser which can be generated has to be implemented from scratch.

In the following chapters this approach will be further investigated and concepts will be elaborated that lead to a possible implementation of an interpreter for the π-calculus.

3 Data Structure and Execution

For the execution an appropriate data structure has to be defined which in this case is a tree structure. It can be applied because the π-calculus consists of split and decision constructs, but no joins. Synchronization between processes and process parts is done by using positive and negative prefixes. For the tree structure to be applied all the structures of the π-calculus have to be mapped to tree structure elements. Section 3.1 deals with this mapping between the π-calculus and the tree and section 3.2 defines how these structures are handled during execution.

First off if a parser is going to be generated, a grammar for the π-calculus is needed. The grammar which will be used in this paper is the one proposed in *The Polyadic π-Calculus: a Tutorial*[3] as follows:

$$\begin{aligned} P &::= M \mid P|P' \mid \mathbf{v}zP \mid !P \mid A(\tilde{y}) \\ M &::= \quad 0 \mid \pi.P \mid M + M' \\ \pi &::= \ \overline{x}\langle y\rangle \mid x(z) \mid \tau \mid [x = y]\pi \end{aligned}$$

A different grammar that can produce different constructs is for example the one suggested in *A Calculus of Mobile Processes, Part I*[2]. This grammar for instance allows for less restrictive summations.

Many different tools exist which starting from a given grammar generate a parser for it. As an example the tools ANTLR (ANother Tool For Language Recognition)[7] and JavaCC (Java Compiler Compiler)[8] might be useful because they already support building abstract syntax trees as intermediate structures and they also provide for functionality to modify these trees with further information while parsing and while the information is directly available from the process definitions. These two tools also support building custom structures if the provided tree structure is not adequate after all. Accordingly no further actions in translating the tree structure and adding information to it need to be done after the parsing step and the process execution can start immediately.

3.1 Specification of the Data Structure

The π-calculus consists of the following constructs as are represented in the grammar

- Summation: $M + M'$
- Composition: $P \mid P'$
- Defined Agents: $A(\tilde{y})$
- Prefix: $\overline{x}.\langle y\rangle$, $x(z)$, τ

- Match: $[x = y]\pi$
- Restriction: $\mathbf{v}zP$
- Replication: $!P$
- Inaction: 0

For most of these constructs special tree node types will be needed with special properties. All tree nodes have one property the process identifier (PID) in common. The process identifier provides the information, which process or sub process this node and the whole subtree belongs to. It will be needed for the scoping of names within and between processes. This sections will define how the mentioned constructs can be mapped to structures in the tree, properties of tree nodes and other structures like tables.

Prefix Prefixes do the actual work within the processes. There are three different types of prefixes. The negative prefix notated as $\overline{y}\langle x\rangle$ defines the action of sending the name x over the channel y. The process which the action is part of will be blocked as long as there is no other process available that can receive a name on channel y. The positive prefix notated as $y(x)$ defines the action of receiving a name x over the channel y. Similar to the negative prefix this process is blocked as long as there is no other process sending a name on channel y. The silent prefix τ performs an unobservable action. The unobservable action can always be performed if it is not blocked by a match, sending or receiving prefix node, that has to be executed beforehand.

As a result three node types are needed, that are *Send*, *Receive* and *Tau*. Additionally the *Send* and *Receive* nodes need two more fields, the first containing the name of the channel on which a name is sent or received and the second containing a set of names which are sent/received. Figure 1 shows these nodes. The names attribute contains a set of names if the polyadic π-calculus is used. A sequence of actions in a process is represented as the child/parent relation in the tree. If an action has to be done before another action, the node that represents the first action will be the parent of the node representing the second action. An example for a sequence of actions with respect to the process definition $x(y).\tau.\overline{y}.0$ is represented in figure 2.

Defined Agents Defined agents may occur within process definitions, if a process is supposed to behave like another agent further on or start all over again. An example for using defined agents is to express recursion. A defined agent holds a list of parameters as placeholders for the names that will be replaced by other names in the agents process definition. The notation for a defined agent in the π-calculus is $A(x_1, x_2, \ldots, x_n)$ with A as the identifier of the agent and $x_1, x_2, \ldots, x_n$ as its parameter values. The node of type *AgentId* is depicted in Figure 3. Extra properties of the agent node are the name of the agent *AgentName* and *Names*, which is a list of the parameters given to that agent.

Summation A summation behaves like one subprocess out of a set of them. Which subprocess will be picked depends on the actions that can be done right

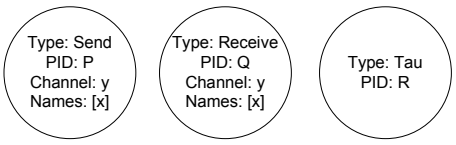


Fig. 1. Prefix nodes.

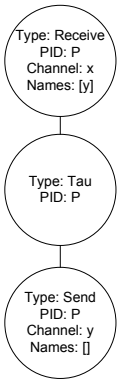


Fig. 2. Sequence of actions.

away, this includes some non-determinism if more than one action can be done. Summations in the π-calculus grammar have the syntax $M_1 + M_2 + \ldots M_n$ with $M_1, M_2, \ldots, M_n$ as possible subprocesses. The summation is represented as a special node type "+" in the tree with the choices as this nodes children. Figure 4 represents the process definition

$$P(a, b) = \overline{a}\langle b\rangle.0 + a(x).0 + \tau.0$$

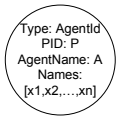


Fig. 3. Defined Agent node.

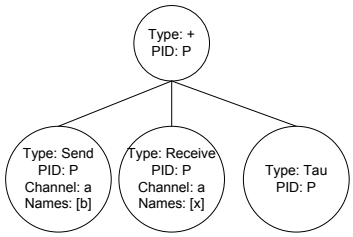


Fig. 4. Summation node.

Composition A composition consists of processes or subprocesses which run in parallel and may interact with each other using channels to exchange names. The π-calculus notation for composition is $P_1 \mid P_2 \mid \ldots \mid P_n$ with the agents

$P_1, P_2, \ldots, P_n$ running in parallel. For this construct a new node type *Composition* is needed and the concurrent agents are simply inserted as its child nodes. Figure 5 shows an example of the process definition

$$\begin{aligned} A(a,b) &= P(a,b) \mid R(b) \\ P(a,b) &= \quad a(x).0|\overline{b}.0 \\ R(b) &= \qquad b.0 \end{aligned}$$

Match The syntax for a match is $[x = y]\pi$ where the action π is done if the condition *x is equal to y* is met. The grammar allows for a conjunction of several match conditions, for this to be represented in the tree a node for each condition is created and inserted as a child node of the previous condition, whereas the order of insertion of the match nodes is of no real importance, because of the commutativity of conjunction. Figure 6(a) shows the match tree node of the construct $[x = y]\overline{a}\langle x\rangle.0$ and Figure 6(b) depicts the process definition of $[a = c][x = y]\overline{a}\langle x\rangle.0$ which is equal to the definition $[x = y][a = c]\overline{a}\langle x\rangle.0$

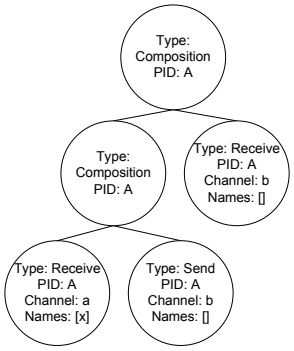


Fig. 5. Composition node.

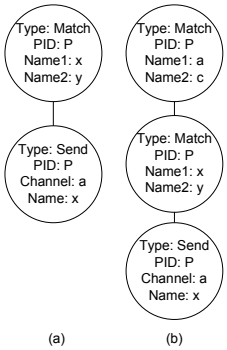


Fig. 6. Match node.

Replication The meaning of the replication construct is that an infinite number of instances of the replicated process are running in parallel. Since an infinite number of processes can not be supplied at any point of time a workaround may be proposed which spawns a replicated process every time the first action of the replication process is executed. This will assure that no deadlock of another process wanting to use a replicated instance will occur. Replication will be matched to a new node type *Replication* with just the property *PID* and the contents of the process definition as child node(s). Figure 7 shows the process definition $\overline{y}.!A(a)$ as an example.

Restriction With the help of restriction new unique names are created and bound to the process(es) they have been created for, which means that they are private within the scope defined by the Restriction. The restriction in the π-calculus is notated as $\mathbf{v}xP$ where x is a placeholder for the private unique name to be created for the process P. One or more names can be restricted to one or more processes, e.g. $S = \mathbf{v}x\mathbf{v}y(P \mid Q) \mid R$. In this case the names x, y are private to the agents P and Q. R does not have any access to these names, yet. It might receive access through scope extrusion during execution later on. The restriction nodes attributes are the name which will be replaced and the process ID of the process it belongs to. The restriction node is depicted in Figure 8 for the process definition of agent S. Additionally a restriction table will be kept which holds an entry for each restricted name with a list of process identifiers belonging to the processes the name is restricted to.

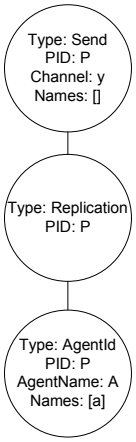


Fig. 7. Replication node.

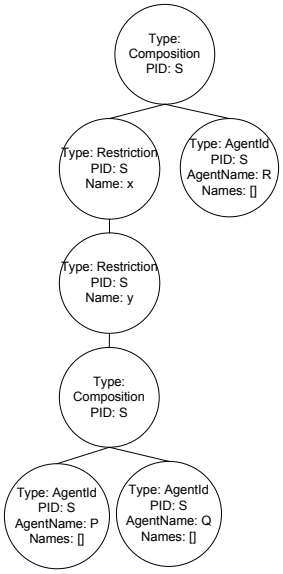


Fig. 8. Restriction node.

Free Names and Bound Names Every time a name is received over a channel a check has to be done beforehand to confirm that no free names become bound by this action and no bound names become overridden. Hence one more table may be kept for optimized execution which holds a process identifier, a list of free names of this process and a list of bound names of this process. Otherwise every time a name is received the whole subtree has to be checked if this name

is already contained somewhere in the process. These tables can be obtained before execution starts.

3.2 Handling of the Data Structure During Execution

After parsing a tree is created which holds the definitions of all defined agents for future reference and the definition of the process which will be executed. The definition of the process to be executed can be found under an extra node called *execution node*. Another approach would be to receive two trees from the parser, one which holds the definitions for all the agents and the other holds the definition of the process that will be executed. Since these approaches are similar to each other concerning the included information the decision which is the easiest to use depends on the later implementation. The first approach will be used in the specifications and examples further on. Figure 9 depicts the basic structure of the tree. The following execution rules only refer to the part of the tree that will be executed which is the subtree of the *execution node*.

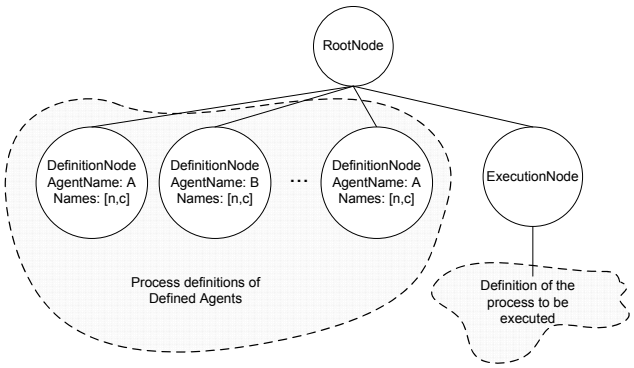


Fig. 9. Basic structure of the tree after parsing.

The execution tree is processed top-down. Actions that are direct children of the execution node are available for execution. Node Types that may occur as the execution nodes children are prefix nodes, replication nodes and summation nodes. Defined agent nodes, composition nodes, match and restriction nodes will not be encountered as the execution nodes direct children during execution because they can be resolved right away as will be specified in the associated execution rules. The only time they might occur as child nodes of the execution node is right after the parsing step is done, but the first steps of the execution will involve the immediate replacement or execution of these nodes.

As mentioned before the simulator is supposed to show the same behavior as the reduction semantics of the π-calculus. The axioms of reduction and structural congruence as specified in [9] are shown in table 1 and table 2 and will be referred to in the execution rules of the associated tree nodes.

Axiom 1	$(\overline{x}\langle y\rangle.P_1 + M_1) \mid (x(z).P_2 + M_2) \rightarrow P_1 \mid P_2\{^y/_z\}$
Axiom 2	$\tau.P + M \rightarrow P$

Table 1. The axioms of reduction.

Sc-Mat	$[x = x]\pi.P \equiv \pi.P$
Sc-Sum-Assoc	$M_1 + (M_2 + M_3) \equiv (M_1 + M_2) + M_3$
Sc-Sum-Comm	$M_1 + M_2 \equiv M_2 + M_1$
Sc-Sum-Inact	$M + \mathbf{0} \equiv M$
Sc-Comp-Assoc	$P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3$
Sc-Comp-Comm	$P_1 \mid P_2 \equiv P_2 \mid P_1$
Sc-Comp-Inact	$P \mid \mathbf{0} \equiv P$
Sc-Res	$\mathbf{v}z\mathbf{v}wP \equiv \mathbf{v}w\mathbf{v}zP$
Sc-Res-Inact	$\mathbf{v}z\mathbf{0} \equiv \mathbf{0}$
Sc-Res-Comp	$\mathbf{v}z(P_1 \mid P_2) \equiv P_1 \mid \mathbf{v}zP_2$, if $z \notin fn(P_1)$
Sc-Rep	$!P \equiv P \mid !P$
Sc-Unfold	$A(\tilde{y}) \equiv P\{^{\tilde{y}}/_{\tilde{x}}\}$, if $A(\tilde{x}) \stackrel{def}{=} P$

Table 2. The axioms of structural congruence.

Execution of Defined Agent Nodes The defined agent node will not become a first level child of the execution node during execution, because the first action of a process has to be known for matching prefixes. Doing the resolving late means to do it every time a matching prefix is looked for instead of only one time when the defined agent node is first met. Therefore before linking a defined agent node to the execution node, it has to be resolved. Resolving denotes to copy the process definitions of that agent from the part of the tree where they are kept and assigning its parameter values with regard to renaming bound names if needed. The rule of structural congruence which leads to the same behavior of renaming the parameters is SC-UNFOLD in table 2. As an effect the resulting structure will now be linked to the execution node.

The list of parameters for a defined agent has to include all free names that occur in the process definition of this agent. If this is not the case and the agent is part of another process, problems might arise because the replacement of names that have existed in the other process and that are also free names of the agent has to take effect in the agent, too, but would not if the name is not included in its list of parameters. In this case the replacement would be lost. As an example the following process definitions will show the issue:

$$P(a) = a(x).Q(x)$$
$$Q(x) = \quad \overline{x}.0$$

Process P for example receives the name d over the channel a, now x is replaced by d everywhere in P and also x will be replaced by d in Q when the defined agent is resolved, because P as defined above is structurally congruent to the definition:

$$P(a) = a(x).\overline{x}.0$$

If Q does not have this free name x as a parameter value, its x is not replaced by d regarding the execution rule as defined above and the behavior would not be consistent with the reduction semantics of the π-calculus.

Execution of Prefix Nodes If a tau node is encountered as a first level child, the unobservable action can be executed immediately, as can be seen in the second axiom in table 1, the tau node will be removed and its child node(s) will be linked to the execution node instead.

For a send prefix node a matching receive prefix node has to be found. Such a node can be found as a sibling node or as a child of a summation or replication sibling node. Thus finding such a node demands in the worst case to check all the sibling subtrees until the first prefix node is found in each. If a matching node is found in one of the sibling subtrees the current node will be removed from the tree and its child node will be linked to the execution node instead. In case of looking for a matching node the scope of the channel name has to be taken into account additionally. Furthermore in the event of sending a restricted name the scope has to be expanded whereas the scope has to be reduced if the sent restricted name does not occur in the sending process any longer. Scope extrusion, intrusion and reduction will be done by updating the entries in the restriction table. For this operation the subtree which belongs to this process has to be traversed.

The receive prefix node basically behaves like the send prefix node during execution once a matching send prefix node is found. Additional tasks which have to be performed relate to taking care of the received name, which includes substitution and renaming if necessary. If a restricted name is received, the receiving processes identifier will be added to the entry of this name in the restriction table. After the name is received the node will be replaced by its child node in the tree. The execution rules for send and receive prefixes correspond to the first axiom in table 1.

In case a node does not have any more child nodes, which means that this process becomes inactive, all its entries in the tables will be removed.

Execution of Match Nodes Match nodes will not be linked to the execution node but are executed before this would happen. Execution in this case means checking if the two names in the fields *Name1* and *Name2* are the same or not. If they represent the same name, according to SC-MAT in table 2 the match node will be replaced by its child node. Otherwise the whole subtree including the match can be removed because the process would be blocked from now on, which is equal to inaction in this case.

Execution of Summation Nodes Summation nodes that are direct children of the root node can be traversed anytime and if one child node is chosen for execution, its execution causes the deletion of its sibling nodes together with their subtrees. The rules of SC-SUM-ASSOC and SC-SUM-COMM are implicitly given in the tree structure since no order on the sibling nodes has been defined.

Execution of Composition Nodes A composition node will not be linked to the root node. Before this would happen it will be executed. The child nodes of the composition node will receive new distinct process identifiers, the former processes entries in the table of restriction will be replaced by the set of new process identifiers and the old processes entries in the tables of bound and free names will be copied to the entries of the new process identifiers. Afterwards the changed child nodes are linked to the execution node and the old composition node is removed. The child nodes have to receive distinct process identifiers, because otherwise their entries in the tables would not be distinguishable and the replacement of a name existing in several of the sub processes should only affect the one sub process that actually received the name. Precisely as the rules of associativity and commutativity for summations, the respective rules for composition SC-COMP-ASSOC and SC-COMP-COMM are also given implicitly.

Execution of Replication Nodes As mentioned before every time the first action of the replication process is executed, a new replicated process is spawned, corresponding to the rule SC-REP in table 2. The new process will receive a new process identifier and its bound and free name lists are copied from the entries of the old process identifier as well as its process identifier is added to the same restriction lists the old one occurs in.

Execution of Restriction Nodes As soon as a restriction node is encountered a new unique name is created and inserted into the restriction table, the scope of the newly created name will be the process identifier of the process the name has been created for. The new name also replaces its placeholder in the subtree.

4 Example

This section shows an example π-process definition which represents processes using memory cells and interacting with each other. The systems representation as a tree is given and it is shown how the process definitions are executed.

The following system S will be examined:

$$\begin{aligned} S(m, hi, hello, cell) &= A(m, hi, cell) | B(m, hello, cell) | Cell_0(cell) \\ A(m, hi, cell) &= cell(a).\overline{a}\langle hi\rangle.\overline{m}\langle a\rangle.0 \\ B(m, hello, cell) &= cell(b).\overline{b}\langle hello\rangle.m(x).x(o).b(p).0 \\ Cell_0(cell) &= !(\mathbf{v}c)\overline{cell}\langle c\rangle.Cell_1(\bot, c) \\ Cell_1(n, c) &= \overline{c}\langle n\rangle.Cell_1(n, c) + c(x).Cell_1(x, c) \end{aligned}$$

The memory cell is represented by the agents $Cell_0$ and $Cell_1$. Process A creates a new memory cell, saves the string hi in this cell, sends the accessor to the memory cell over the channel m and terminates. B also creates a memory cell and saves the string *hello*. It then receives a name over the channel m, which is the accessor to A's memory cell. B reads the saved names from both cells and

terminates. In the following lines a description will be given for an exemplary evolvement of the system. The descriptions can be followed with the help of some figures, that can be found in the appendix A.

Before starting the execution a tree is build, the tree resulting from the above process definitions is depicted in figure10. On the left side of the tree the process definitions of the agents that are part of the system can be seen. These agents are S, A, B, $Cell_0$ and $Cell_1$. On the right hand side of the tree the execution node can be seen with the process definitions of agent S as children, this part of the tree will be executed so it is assigned a valid process identifier, in this case S.

The first steps of the execution will be to resolve the composition node and the defined agent nodes. Resolving the composition node leads to its children being assigned new process identifiers (A, B, C) and resolving the agent nodes leads to copying the process definitions from the left part of the tree to the places where the agents have been and assigning the associated process identifiers to the sub nodes. Also the entries of the tables for free and bound names are copied to the entries of the new process identifiers. The sub tree under the execution node is the only part of the tree that will change during the execution, so the following figures will focus on this part. What the execution subtree and the table look like after these steps is depicted in figure 11.

Subsequently two ways of execution are possible, either process A or process B might initialize a memory cell. Figure 12 shows what the execution subtree looks like after process A has done the initialization. A new replicated process has been spawned off with a new process identifier $C1$ and a new name d for accessing the memory cell C has been created. This name is shared between the processes C and A, as can be read from the restriction table. For further execution the defined agent $Cell_1$ has to be resolved in process C, this is shown in figure 13.

Now process B is still able to initialize a memory cell and process A might write the name hi in its own memory cell C using the private channel d. Figures 14 and 15 show the execution subtree during the steps of process A writing in the memory cell. Figures 16 and 17 show how process B initializes a memory cell, where a new replicated process $C2$ is spawned off and a new name e is created and shared between the processes B and $C1$, which is used by B to save the name $hello$ in the memory cell.

Process A's last step is to send its accessor for the memory cell to process B, which enables B to use this memory cell. Afterwards A becomes inactive and its entries are deleted from the tables, depicted in figure 18. Further actions that are available now are for process B to read from both memory cells.

As an interesting issue to point out after processes A and B have become inactive is, that processes C, $C1$ and $C2$ are still in the system but no further actions can be done. This shows another topic that has to be investigated, which is memory management. The question that has to be answered is if a process like the memory cell can be removed, since there is no other process in the system that holds an accessor to it. This problem might become interesting if

long running processes that use certain data structures are inspected whereas system ressources have to be taken into account.

5 Conclusion

In this paper two ways of handling the π-calculus as a domain specific language have been discussed and one of these approaches, the approach of implementing a dedicated interpreter has been further investigated. For the chosen approach to be applicable for execution a data structure has been defined that represents the π-calculus process definitions. The data structure that was explored is a tree structure and mapping rules have been given that construct the tree structure from the process definitions. Furthermore rules on the structure have been introduced that lead to an execution of the process definitions. The execution rules are based on the axioms of reduction and structural congruence and therefore the simulator resulting from the definitions behaves in the same way as reducing the process definitions considering the reduction semantics of the π-calculus. Secluding an example has been presented to illustrate the defined data structure and rules.

Further investigations will have to deal with the visual representation of the current systems state and how information for this representation can be deduced from the tree structure. Additionally the option for defining abstraction levels by the user has to be investigated and how the given data structure can be enriched with this information, so the component for visualizing the system is able to interpret and display processes and process parts as black and white box systems according to the users specifications. Additionally some research has to be done on memory management. Cases have to be detected when processes do not have any more influence on the future evolvement of the system and can therefore be cleared.

References

1. Jim Weirich. *Speaking the Lingo: Creating Domain Specific Languages in Ruby.* RubyConf2005 - Speaking the Lingo `http://onestepback.org/articles/lingo/` (December 17th, 2005)
2. Robin Milner, Joachim Parrow, David Walker. *A Calculus of Mobile Processes, Part I.* June 1989 (Revised 1990)
3. Robin Milner. *The Polyadic π-Calculus: a Tutorial.* Computer Science Department, University of Edinburgh, October 1991
4. Björn Victor, Faron Moller, Mads Dam, Lars-Henrik Eriksson. *The Mobility Workbench.* University of Uppsala, Department of Information Technology, `http://www.it.uu.se/research/group/mobility/mwb` (January 15th, 2006)
5. Sebastian Briais. *Another Bisimulation Checker.* `http://lampwww.epfl.ch/~sbriais/abc/abc.html`, (January 15th, 2006)
6. Ulrik Fendrup, Jesper Nyholm Jensen, Hans Hüttel. *Open Bisimulation Checker, Checking for Open Bisimilarity in the π-Calculus.* `http://www.cs.auc.dk/research/FS/ny/PR-pi/`, (January 15th, 2006)

7. Terence Parr. *ANother Tool For Language Recognition, ANTLR Parser Generator*. `http://www.antlr.org/`, (January 20th, 2006)
8. Java Compiler Compiler. *JavaCC - The Java Parser Generator*. `https://javacc.dev.java.net/`, (January 20th, 2006)
9. Davide Sangiorgi, David Walker. *The π-calculus, A Theory of Mobile Processes*. Cambridge University Press, 2001

A Memory Cell Example Figures

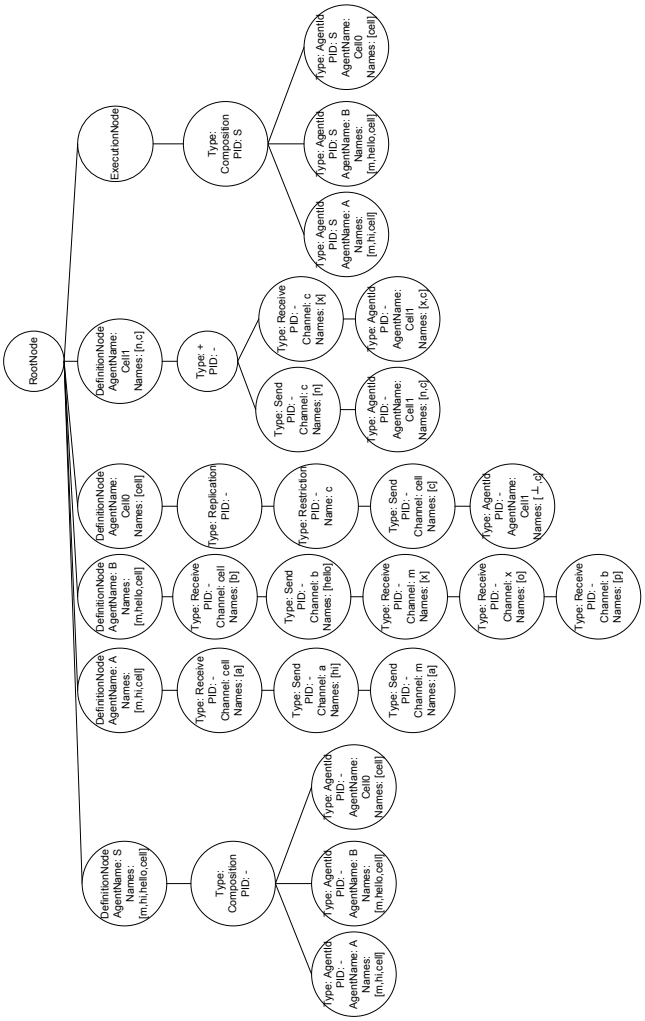


Fig. 10. Tree after parsing.

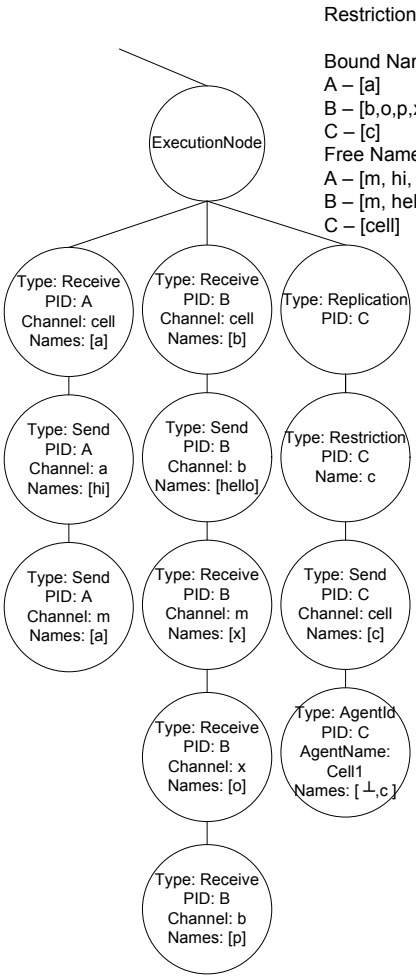


Fig. 11. Process Definitions under Execution Node.

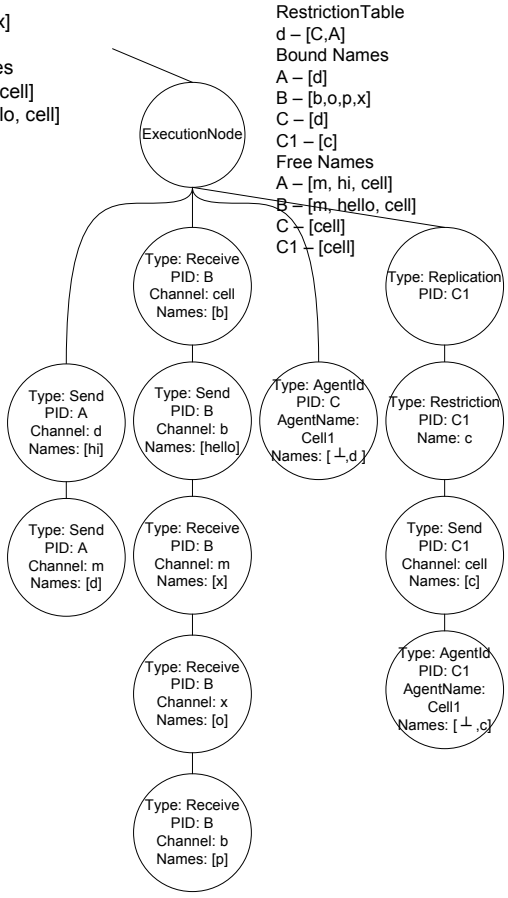


Fig. 12. *A* initializes memory cell.

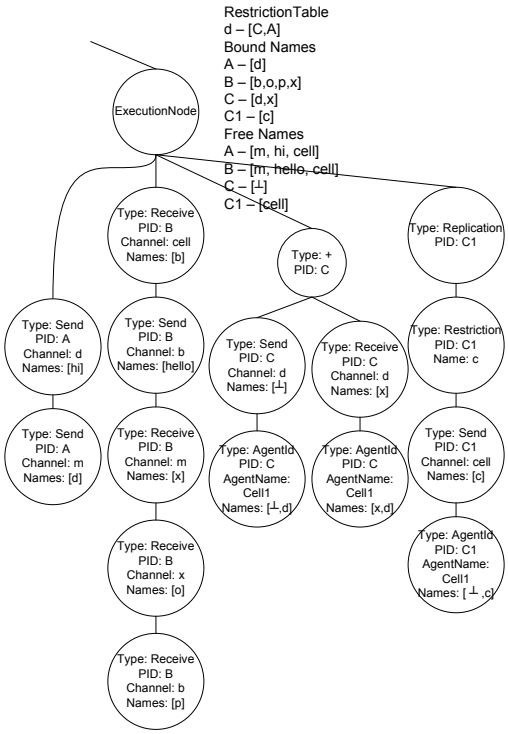


Fig. 13. Resolve agent $Cell1$.

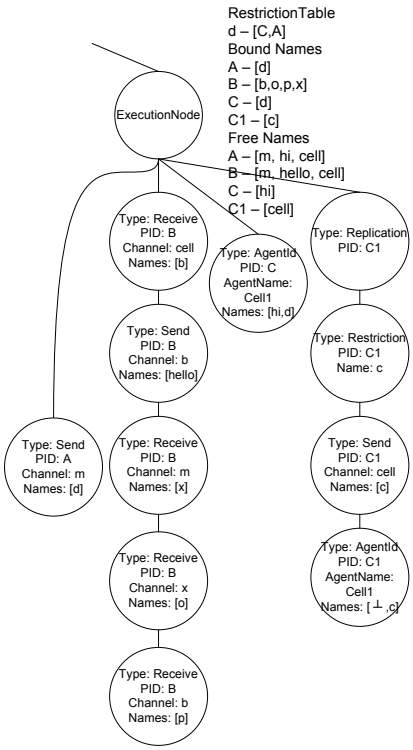


Fig. 14. A writes in memory cell.

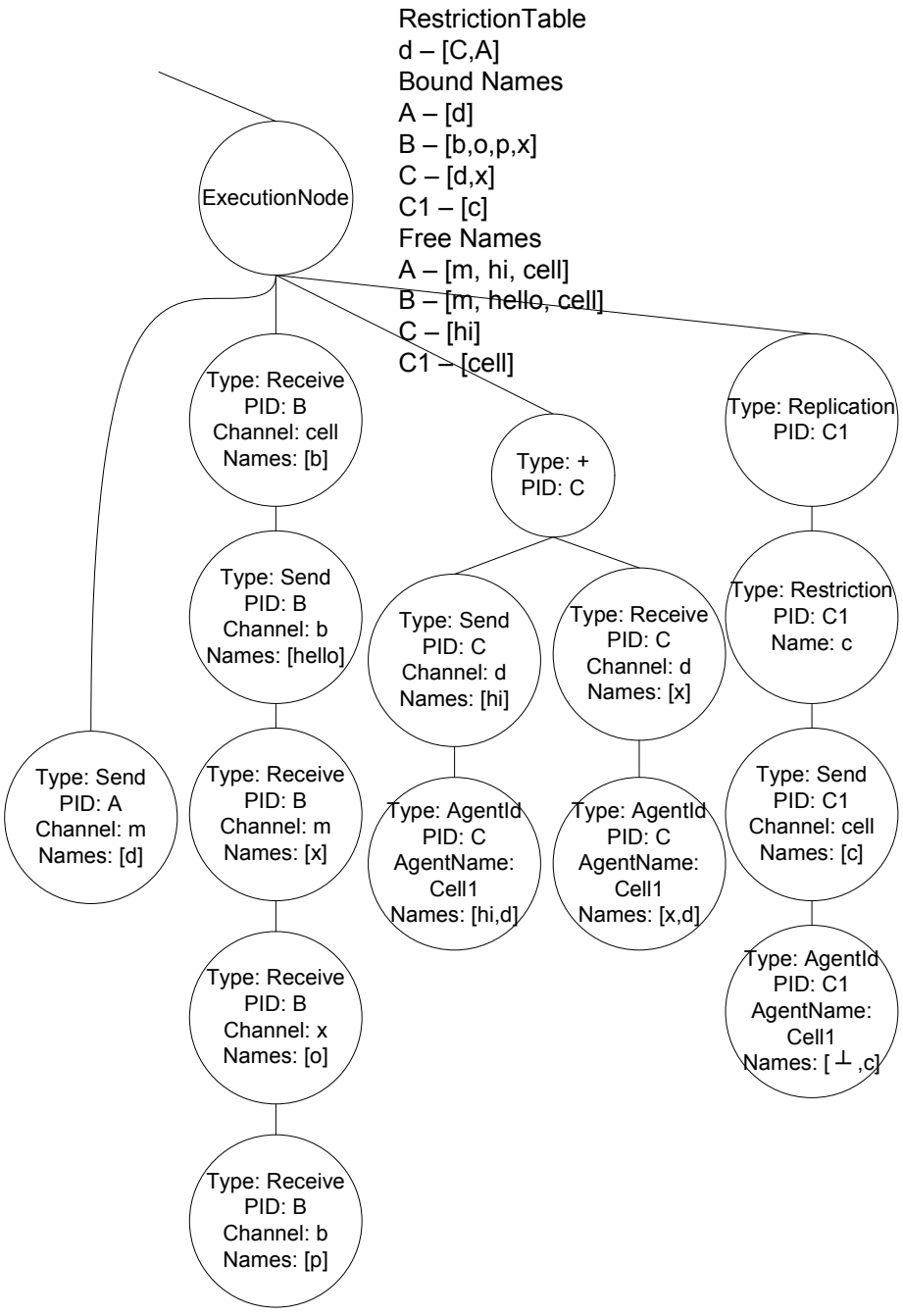


Fig. 15. Memory cell after A wrote in it.

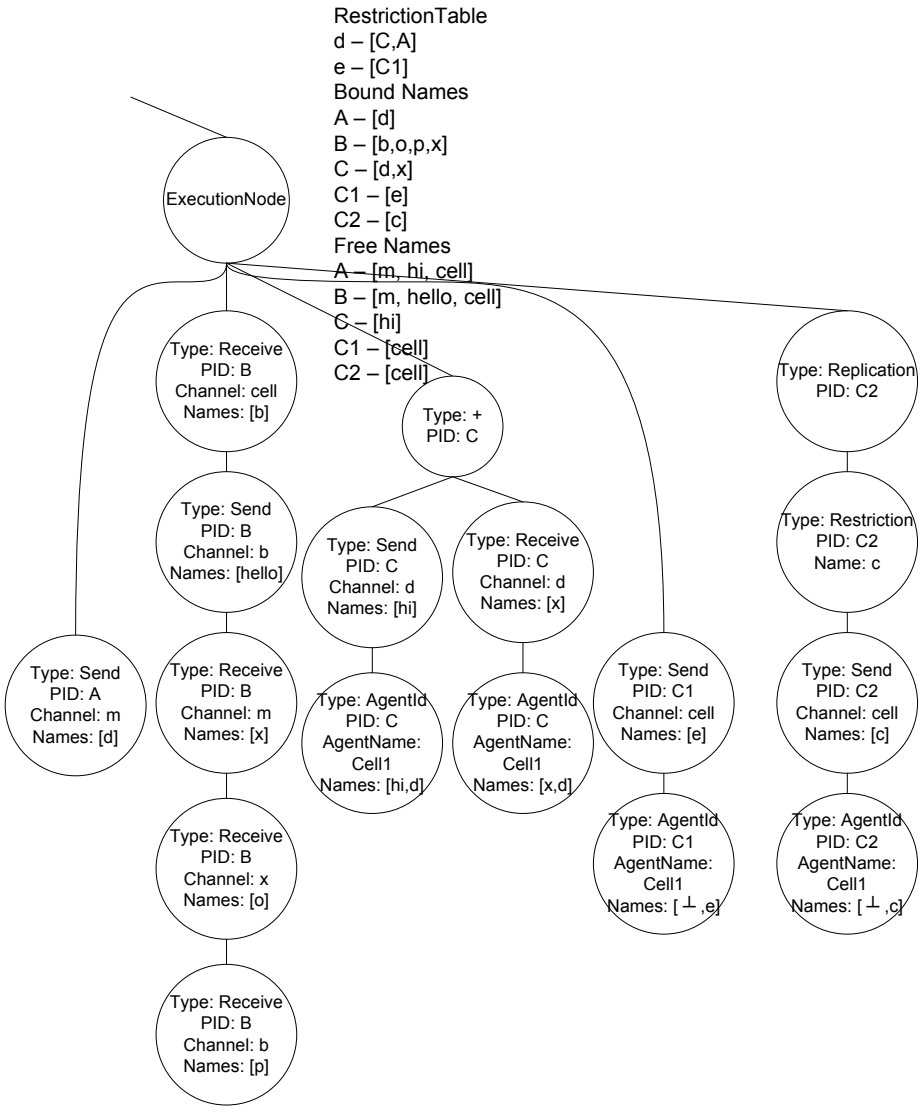


Fig. 16. B initializes a memory cell.

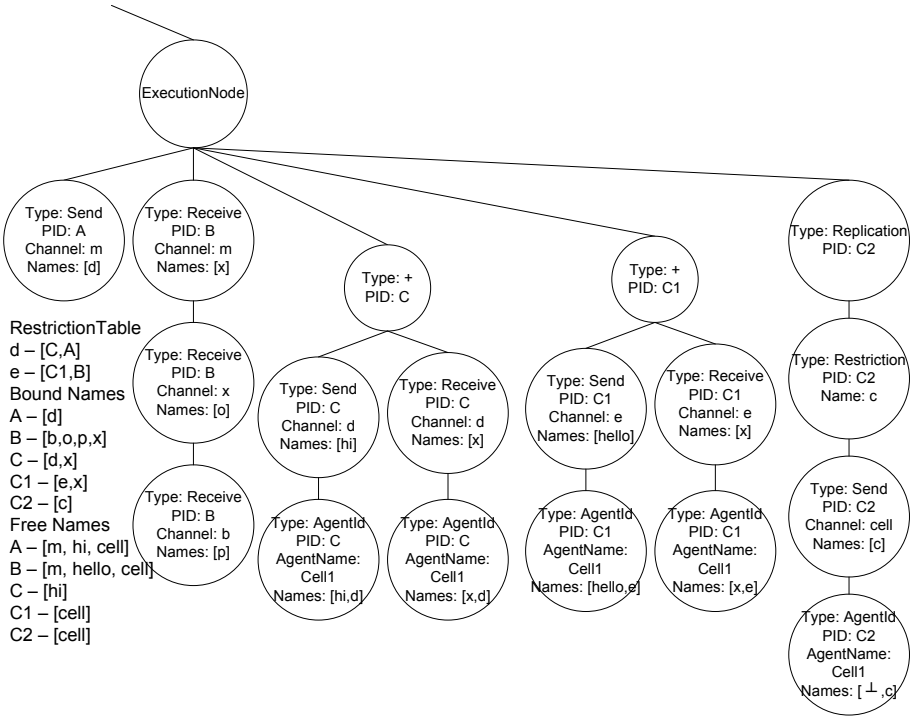


Fig. 17. B's memory cell after initializing.

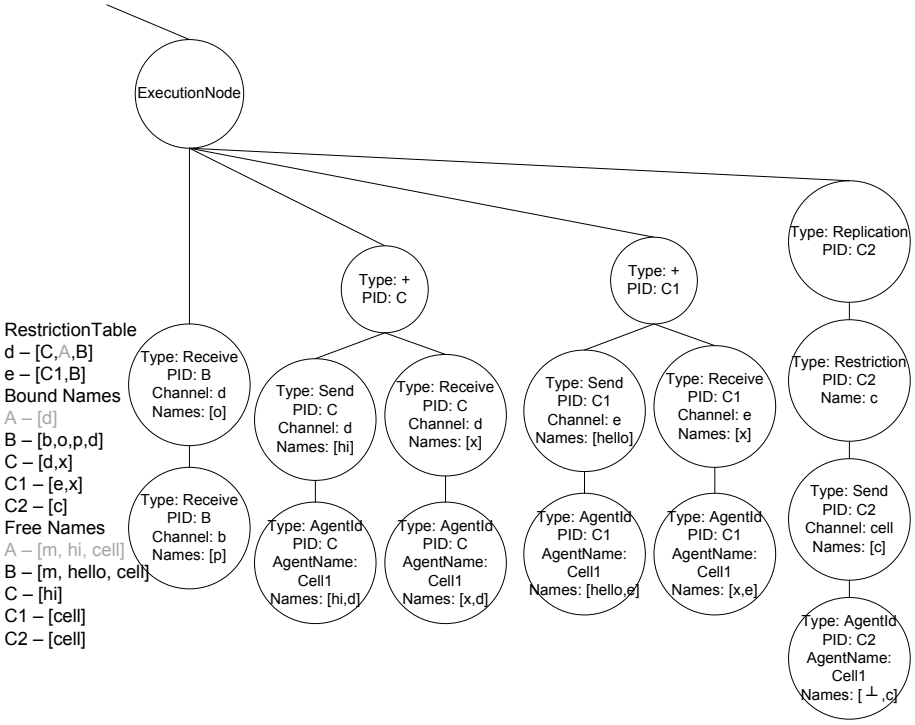


Fig. 18. A sends its accessor to B.

Formalizing Service Interactions

Gero Decker

Hasso-Plattner-Institute for IT-Systems Engineering at the University of Potsdam, Germany
Prof.-Dr.-Helmert-Strasse 2-3, 14482 Potsdam, Germany
gero.decker@hpi.uni-potsdam.de

Abstract. An important aspect of cross-organizational business process integration is the identification of the interactions between individual business partners. In order to unambiguously describe these interaction protocols (choreographies) and to allow for reasoning we need a formal grounding for every interaction type.
This paper shows how the Service Interaction Patterns can be formalized using π-calculus. A small set of formalizations based on Petri nets is also provided.
Since mobility is a central aspect of service choreographies we argue that π-calculus is better suited for formalizing service interactions than place/transition Petri Nets and Colored Petri Nets.

1 Introduction

Service-oriented architectures (SOA) as described in [1] aim at closely supporting business processes within a company and between business partners. Services are employed to perform tasks within these processes and processes themselves can be exposed as services. We distinguish between orchestrations where one business partner enacts a set of services in a given order and choreographies which represent the interaction protocols between several business partners (see [2]).

In a setting where the different business partners encapsulate their business logic as services, service interactions are at the center of attention. A lot of effort has been undertaken to identify the most common interaction scenarios from a business perspective, which have been published as "Service Interaction Patterns" in [3]. Barros et al categorize the patterns according to the number of participants in an interaction (bilateral vs. multi-lateral), the maximum number of exchanges (single-transmission vs. multi-transmission interactions) and whether the receiver of a response is necessarily the same as the sender of a request (round-trip vs. routed interactions).

A textual description is provided for each pattern, together with business examples and design choices. The authors also come up with implementation examples using BPEL [4] and other standards from the WS-* stack. However, formal representations of the patterns are still missing.

Such formalizations could be used for reasoning and simulation. Especially conformance checking between a given choreography and a set of orchestrations has gained a lot of interest recently.

When looking into Business Process Management (BPM) literature (e.g. [5]) we see that Petri nets in all their different flavors dominate the research community. Therefore, we have investigated if Petri nets are suitable for formalizing the interaction patterns.

In the last years π-calculus, a general purpose process algebra, came up in the BPM domain. We will analyze if communication and change, the core aspects of π-calculus, are also at the heart of the interaction patterns.

The next two sections will first cover our formalization approach using π-calculus and then using different classes of Petri nets. The Related Work section puts this paper into relation with other publications. Finally, a conclusion will be drawn and an outlook will be given.

2 Formalizing the Interaction Patterns using π-calculus

At the center of π-calculus we deal with processes that communicate with each other. The communication channels as well as the messages sent over these channels are called names. That way, channels can be passed as parameters to other processes and be used for communication later on. This capability of systems is called mobility. For more information on π-calculus please refer to [6], [7], [8] and [9].

In the pattern representations each interaction participant is modeled as a π-process. In the case of bilateral interactions we named them A and B, in the case of multi-lateral interactions A, B_i and P where $i = 1, 2, \ldots$.

Since timers and exception handling are explicitly called for in the patterns we introduce an environmental process $\mathcal{E}_X$ per interaction participant ($X = A, B, B_i, P$). It is left open how timeouts and exception handling are implemented. $\overline{settimer_{\mathcal{E}_X}}\langle timer\rangle$ is supposed to set a new timer where a timeout is thrown by sending on channel $timer$. Exceptions can be thrown by sending on channel $\overline{fault_{\mathcal{E}_X}}$.

In the π-calculus a message is sent and received at the same time. I.e. if a process wants to send a message via a link then it blocks until a receiver actually receives the message. Therefore, the π-calculus assumes synchronous communication as well as reliable and guaranteed delivery as the default case.

If we want to model that a message sent from A to B can get lost or that the delivery can be delayed we might introduce a medium M in the following way:

$$
\begin{aligned}
A &= \overline{b_1}\,\langle msg\rangle\,.\mathbf{0} \\
M &= b_1(msg).((\overline{b_2}\,\langle msg\rangle\,.\mathbf{0} + \tau_0.\mathbf{0}) \mid M) \\
B &= b_2(msg).\mathbf{0}
\end{aligned}
$$

The medium decides non-deterministically if the message is either discarded or forwarded to B. Due to the sequence within M, sending msg in A and receiv-

ing msg in B do not happen at the same point in time, which models a potential delay as well as asynchronous communication. Another property of the medium concerning the order of delivery is that FIFO (first in first out) is not guaranteed any longer. Introducing such a medium allows for reasoning if a process still meets its requirements even in the case of delayed or unreliable message delivery.

However, we assume that the underlying infrastructure (represented by the environmental processes $\mathcal{E}$) guarantees reliable FIFO delivery. In the cases where this cannot be met we assume that an exception $\overline{fault_{\mathcal{E}}}$ is raised for the whole composition.

As proposed in [8] we omit the termination symbol **0** in process definitions and we use the Polyadic π-calculus for better readability. The Polyadic π-calculus allows for a shorter representation of sending several names over the same channel such as $\overline{b}\langle x, y, z\rangle$.

The following subsections present formalizations for every pattern. The pattern descriptions at the beginning of each subsection are directly taken from [3].

2.1 Pattern 1: Send

A party sends a message to another party.

The pattern definition distinguishes between blocking send and non-blocking send. In the case of blocking send the sending process cannot proceed until it can be sure that the message has been received. As already mentioned above this blocking behavior is inherent to π-calculus.

Blocking send:

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

This pattern formalization leaves it open if the receiver of the message is known at design-time or not. If we define the system as

$$S = (\nu\ b)(A \mid B)$$

then A knows the link to B at design-time. If we define it as

$$S = (\nu\ lookup)(lookup(b).A \mid (\nu\ b)(B \mid D))$$

then A would get the link to B at run-time. In this case D could be something like a UDDI directory where we can lookup the receiver.

A' and B' represent the so called continuations mentioned in the pattern descriptions.

Non-blocking send:

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

Strictly speaking we could omit the formalization for B. However, for illustration purposes we provide one possible implementation for B to have a valid choreography.

Most interaction patterns describe the interactions from the perspective of one single participant. In order to get a minimal choreography we have to plug several patterns together (e.g. send for A and receive for B).

2.2 Pattern 2: Receive

A party receives a message from another party.

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

2.3 Pattern 3: Send / receive

A party A engages in two causally related interactions: in the first interaction A sends a message to another party B (the request), while in the second one A receives a message from B (the response).

In order to keep track of the relation between the two interactions we have to introduce some kind of correlation mechanism. In π-calculus we can create an infinite number of new names for a process that are not known to any other process. We create a new name for every set of correlated interactions. If we then use new this channel for communication we can be sure that any message that is sent over this channel is correlated to the other interactions.

In the following example A creates a new name a used for receiving the response. A blocks until the corresponding message has been received.

Blocking send / receive:

$$A = (\nu\ a)\overline{b}\,\langle a, req\rangle .a(resp).A'$$
$$B = b(a, req).\tau_B.\overline{a}\,\langle resp\rangle .B'$$

As already mentioned above the formalization of B is only one valid example. E.g. we could imagine interactions with other processes between receiving the request from A and sending the response.

Non-blocking send / receive:

$$A = (\nu\ a, h)(A_1|A_2)$$
$$A_1 = \overline{b}\,\langle a, req\rangle .(\overline{h} \mid A_1')$$
$$A_2 = h.a(resp).A_2'$$
$$B = b(a, req).\tau_B.\overline{a}\,\langle resp\rangle .B'$$

The new name h has to be introduced since the pattern descriptions demands that the request has been sent before a response can be received.

2.4 Pattern 4: Racing incoming messages

A party expects to receive one among a set of messages. These messages may be structurally different (i.e. different types) and may come from different categories of partners. The way a message is processed depends on its type and/or the category of partner from which it comes.

Normally names are not typed in π-calculus. In order to retrieve the type of a message we could explicitly receive a second name representing the type. We opted for a more elegant way: for each type a channel is created and thus the channel a message is sent over determines the message's type.

In the following formalization we assume that there are two different types of messages. Each B_i can a message over channel a_1 if it is of the first type or over channel a_2. Depending on the type of the message the continuation for A is either A'_1 or A'_2.

The pattern distinguishes between discarding remaining messages and keeping them for further interactions.

Remaining messages are not discarded:

$$A = (a_1(msg).A'_1 + a_2(msg).A'_2)$$
$$B_i = (\overline{a_1}\langle msg\rangle .B'_i + \overline{a_2}\langle msg\rangle .B'_i)$$

Once again the formalization for B_i is just an example. In this case every B_i can sent messages of every type. If we want to model that the continuation of A depends on the category of the sender we could define $B_i = \overline{a_1}\langle msg\rangle .B'_i$ and introduce another category $C_i = \overline{a_2}\langle msg\rangle .C'_i$.

A generic formalization for an arbitrary number of different types / categories would be

$$A = \Sigma_{i=1}^{n} a_i(msg).A'_i$$

In the following formalization all remaining messages are received but not taken care of. The number of messages is not known at run-time:

$$A = a_1(msg).(A'_1 \mid A_{discard}) + a_2(msg).(A'_2 \mid A_{discard})$$
$$A_{discard} = !a_1(msg) \mid !a_2(msg)$$
$$B_i = (\overline{a_1}\langle msg\rangle .B'_i + \overline{a_2}\langle msg\rangle .B'_i)$$

In order to be able to receive an arbitrary number of messages we have to use the !-operator which stands for an infinite number of processes that run in parallel. Therefore, A never terminates. It can be seen as a drawback of π-calculus that discarding an unknown number of messages cannot be modeled in a different way.

2.5 Pattern 5: One-to-many send

A party sends messages to several parties. The messages all have the same type (although their contents may be different).

Number of recipients known at design-time:

$$\begin{aligned} A &= (\nu\ h)(\Pi_{i=1}^{n}\overline{b_i}\,\langle msg_{B_i}\rangle\,.\overline{h}\ \mid\ \{h\}_1^n.A') \\ B_i &= b_i(msg).B_i' \end{aligned}$$

Number of recipients known at run-time:

$$\begin{aligned} A =&(\nu\ h)(A_1(h)\ \mid\ h.A') \\ A_1(h) =&hasnext(hn). \\ &([hn = true]next(b).(\nu\ i)(\overline{b}\,\langle msg\rangle\,.i.\overline{h}\ \mid\ A_1(i)) \\ &+ [hn = false]\overline{h}) \\ B_i =&b_i(msg).B_i' \end{aligned}$$

For this pattern we have introduced the concept of an iterator that iterates over a list of recipients. The "operation" *hasnext* returns either $true$ or $false$ and tells us if there are more elements in the list. *next* returns the next recipient in the list. An iterator could be defined using data structures such as a stack (see e.g. [10]).

2.6 Pattern 6: One-from-many receive

A party receives a number of logically related messages that arise from autonomous events occurring at different parties. The arrival of messages needs to be timely so that they can be correlated as a single logical request. The interaction may complete successfully or not depending on the set of messages gathered.

The pattern introduces the notion of a stop condition and a success condition. In the simplest flavor of the pattern the interaction succeeds if n messages have been received. In this case we stop the interaction as soon as this success condition is met or a timeout has occurred:

$$\begin{aligned} A =&(\nu\ timer, kill)(\overline{settimer_{\mathcal{E}_A}}\langle timer\rangle.\ \{a(msg)\}_1^n\ .\overline{exec} \\ &\mid\ (exec.\overline{kill}.A' + kill)\ \mid\ (timer.\overline{kill}.\overline{fault_{\mathcal{E}_A}} + kill)) \\ B_i =&\overline{a}\,\langle msg\rangle\,.B_i' \end{aligned}$$

In order to model arbitrary stop and success conditions we can use non-deterministic choices:

$$A = (\nu\ exec, timer)(\overline{settimer_{\mathcal{E}_A}}\langle timer\rangle.(!a(msg).(\underbrace{\overline{exec} + \tau_0}_{stop\ condition}))$$
$$\mid timer.\overline{exec} \mid exec.(\underbrace{A' + \overline{fault_{\mathcal{E}_A}}}_{success\ condition}))$$
$$B_i = \overline{a}\,\langle msg\rangle\,.B_i'$$

An infinite number of messages can be received over channel a. Each time a message arrives we check if the stop condition is already met. If this is the case then the process sends over channel $\overline{exec}$. Now it is checked if the success condition is met.

The pattern description defines that the success of the interaction depends on the set of messages. This is not explicitly modeled in the formalization.

2.7 Pattern 7: One-to-many send / receive

A party sends a request to several other parties, which may all be identical or logically related. Responses are expected within a given timeframe. However, some responses may not arrive within the timeframe and some parties may even not respond at all. The interaction may complete successfully or not depending on the set of responses gathered.

If the recipients are known at design-time and we assume arbitrary stop and success conditions the formalization of one-to-many send / receive looks as follows:

$$\begin{aligned} A =& (\nu\ timer)(\overline{settimer_{\mathcal{E}_A}}\langle timer\rangle.\Pi_{i=1}^{n} A_i \\ & \mid timer.\overline{exec} \mid exec.(A' + \overline{fault_{\mathcal{E}_A}})) \\ A_i =& (\nu\ a)(\overline{b_i}\,\langle a, req\rangle\,.a(resp).(\overline{exec} + \tau_0)) \\ B_i =& b_i(a, req).\tau_{B_i}.\overline{a}\,\langle resp\rangle\,.B_i' \end{aligned}$$

If the recipients are known at run-time we introduce an iterator once again:

$$\begin{aligned} A =& (\nu\ timer)(\overline{settimer_{\mathcal{E}_A}}\langle timer\rangle.A_1 \mid timer.\overline{exec} \mid exec.(A' + \overline{fault_{\mathcal{E}_A}})) \\ A_1 =& hasnext(hn).([hn = true]next(b).(A_1 \mid A_2(b)) + [hn = false]\tau_0) \\ A_2(b) =& (\nu\ a)(\overline{b}\,\langle a, req\rangle\,.a(resp).(\overline{exec} + \tau_0)) \\ B_i =& b_i(a, req).\tau_{B_i}.\overline{a}\,\langle resp\rangle\,.B_i' \end{aligned}$$

2.8 Pattern 8: Multi-responses

A party A sends a request to another party B. Subsequently, A receives any number of responses from B until no further responses are required. The trigger of no

further responses can arise from a temporal condition or message content, and can arise from either A or B's side.

In the following formalization we do not explicitly model the temporal condition mentioned in the pattern description:

$$\begin{aligned} A &= (\nu\ a)\overline{b}\,\langle a, req\rangle\,.A_{receive} \\ A_{receive} &= a(resp).\tau_A.(A_{receive} + A' + \overline{b}\langle stop\rangle.A') \\ B &= b(a, req).B_{send} \\ B_{send} &= \tau_B.(\overline{a}\,\langle resp\rangle\,.(B_{send} + B') + b(stop).B') \end{aligned}$$

In $A_{receive}$ we see a non-deterministic choice with three alternatives. The first alternative can be taken if another message arrives from B. The second alternative is chosen if the message content of the last message indicated the termination of the interaction. The third option models the possibility that A decides to stop the interaction.

These three options also appear in B_{send}. Either another message is sent or B decides to terminate the interaction or a stop-message is received from A.

2.9 Pattern 9: Contingent requests

A party A makes a request to another party B1. If A does not receive a response within a certain timeframe, A alternatively sends a request to another party B2, and so on.

A retrieves the list of invocation targets at run-time. For that purpose we use an iterator once again.

An import design issue of this pattern is whether or not responses should be considered after the timeout has already occurred.

Do not consider responses from services we invoked earlier:

$$\begin{aligned} A =& hasnext(hn). \\ & ([hn = true]\, next(b).(\nu\ a)\overline{b}\,\langle a, req\rangle\,.(\nu\ timer)\overline{settimer_{\mathcal{E}_A}}\langle timer\rangle. \\ & (a(resp).A' + timer.A) \\ & + [hn = false]\,\overline{fault_{\mathcal{E}_A}}) \\ B_i =& b_i(a, req).\tau_{B_i}.\overline{a}\,\langle resp\rangle\,.B_i' \end{aligned}$$

Consider responses from services we invoked earlier:

$$
\begin{aligned}
A =&(\nu\ exec, h)(hasnext(hn).\\
&([hn = true]\, next(b).(\nu\ a)\overline{b}\,\langle a, req\rangle\,.(\nu\ timer)\overline{settimer_{\mathcal{E}_A}}\langle timer\rangle.\\
&(a(resp).\overline{exec} + timer.(A \mid (a(resp).\overline{exec} + h)) + h)\\
&+ [hn = false]\,\overline{fault_{\mathcal{E}_A}})\\
&\mid exec.(A' \mid !\overline{h}))\\
B_i =&b_i(a, req).\tau_{B_i}.\overline{a}\,\langle resp\rangle\,.B_i'
\end{aligned}
$$

2.10 Pattern 10: Atomic multicast notification

A party sends notifications to several parties such that a certain number of parties are required to accept the notification within a certain timeframe. For example, all parties or just one party are required to accept the notification. In general, the constraint for successful notification applies over a range between a minimum and maximum number.

We introduce the two names "accept" and "reject". If A receives a reject-message, the multicast notification has failed and the continuation A_2' is taken. If m accept-messages are received before a reject-message arrives the multicast notification has succeeded and the continuation A_1' is taken:

$$
\begin{aligned}
A =&(\nu\ a, h, exec)((\Pi_{i=1}^{n}\overline{b_i}\,\langle a, notification\rangle\,.a(resp).\\
&([resp = accept]\overline{h} + [resp = reject]\overline{exec}\,\langle reject\rangle))\\
&\mid \{h\}_1^m.\overline{exec}\,\langle accept\rangle\\
&\mid exec(resp).(([resp = accept]A_1' + [resp = reject]A_2') \mid \Pi_{i=1}^{n}\overline{b_i}\,\langle resp\rangle))\\
B_i =&b_i(a, notification).\tau_{B_i}.\\
&(\overline{a}\,\langle accept\rangle\,.b_i(resp).([resp = accept]B_{i1}' + [resp = reject]B_{i2}') +\\
&\overline{a}\,\langle reject\rangle\,.B_{i2}' + b_i(resp).B_{i2}')
\end{aligned}
$$

2.11 Pattern 11: Request with referral

Party A sends a request to party B indicating that any follow-up response should be sent to a number of other parties (P1, P2, ..., Pn) depending on the evaluation of certain conditions. While faults are sent by default to these parties, they could alternatively be sent to another nominated party (which may be party A).

While the pattern descriptions talks about a number of parties P_i, the following formalization only presents the case of one party P for better readability:

$$
\begin{aligned}
A &= (\nu\ a)\overline{b}\,\langle a, p, req\rangle\,.a(resp).A'\\
B &= b(a, x, req).\tau_B.\overline{x}\,\langle a, msg\rangle\,.B'\\
P &= p(a, msg).\tau_P.\overline{a}\,\langle resp\rangle\,.P'
\end{aligned}
$$

If we want the follow-up responses to be sent to several P_i we could incorporate the pattern one-to-many send into B.

The following formalization models that all faults are sent to A:

$$A = (\nu\ a, toa)\overline{b}\,\langle a, p, toa, req\rangle\,.(a(resp).A' + toa.\overline{fault_{\mathcal{E}_A}})$$
$$B = b(a, x, faultto, req).\tau_B.(\overline{x}\,\langle a, faultto, msg\rangle\,.B' + (\overline{faultto} \mid \overline{fault_{\mathcal{E}_B}}))$$
$$P = p(a, faultto, msg).\tau_P.(\overline{a}\,\langle resp\rangle\,.P' + (\overline{faultto} \mid \overline{fault_{\mathcal{E}_P}}))$$

In this pattern it becomes obvious that every participant needs his own exception handling mechanism which is implemented in the corresponding environmental processes $\mathcal{E}_A$, $\mathcal{E}_B$ and $\mathcal{E}_P$.

2.12 Pattern 12: Relayed request

Party A makes a request to party B which delegates the request to other parties (P1, ..., Pn). Parties P1, ..., Pn then continue interactions with party A while party B observes a "view" of the interactions including faults. The interacting parties are aware of this "view" (as part of the condition to interact).

Like we already did it for the last pattern we only model one party P:

$$A = (\nu\ a)\overline{b}\,\langle a, req\rangle\,.a(resp).A'$$
$$B = b(a, req).\overline{p}\,\langle a, b, req\rangle\,.b(resp).B'$$
$$P = p(a, b, req).\tau_P.(\nu\ h)(\overline{a}\,\langle resp\rangle\,.\overline{h} \mid \overline{b}\,\langle resp\rangle\,.\overline{h} \mid h.h.P')$$

2.13 Pattern 13: Dynamic routing

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 subject to dynamic conditions based on data contained in the original request or obtained in one of the "intermediate steps".

Since the pattern description covers a broad range of possible interactions we will only focus on two aspects. The first aspect is routing the request to a third party based on the content of the message. If there are n possible recipients C_1 to C_n known at design-time the formalization looks as follows.

$$A = \overline{b}\,\langle req_1\rangle\,.A'$$
$$B = b(req_1).\tau_B.\Sigma_{i=1}^{n}\overline{c_i}\,\langle req_2\rangle\,.B_i'$$
$$C_i = c_i(req_2).C_i'$$

The second aspect that is mentioned in the detailed pattern description covers routing a document from one participant to another where every participant has

read-only access to the document the whole time but has to retrieve a write-token when wanting to modify the document.

The following formalization shows how such data structures can be expressed in π-calculus.

$$\begin{aligned} Gen_D &= !(\nu\ read, lock, x)\overline{create}\langle read, lock\rangle.D_{unlocked}(x) \\ D_{unlocked}(x) &= \overline{read}\langle x\rangle.D_{unlocked}(x) \\ &\quad + (\nu\ write, ul)\overline{lock}\langle write, ul\rangle.D_{locked}(x, write, ul) \\ D_{locked}(x, write, ul) &= \overline{read}\langle x\rangle.D_{locked}(x, write, ul) \\ &\quad + write(y).D_{locked}(y, write, ul) \\ &\quad + ul.D_{unlocked}(x) \end{aligned}$$

Gen_D is a generator that can create an infinite number of memory cells. Every participant wanting access to this memory cell needs the *read* and *lock* channels.

As soon as a participant locks the cell, this participant gets the *write* channel that he can use in order to modify the contents. He has to unlock the cell before anyone else can retrieve a *write* channels.

Every time the cell is locked, new names for the *write* and *unlock* channels are created. That way it is ensured that only the participant who has currently locked the cell can write on it and can unlock it.

Let us now assume a scenario where a participant A sends a document to a set of recipients B_1 ... B_n. In an inter-leaved parallel routing manner every participant is asked to modify the document.

$$\begin{aligned} A =&(\nu\ h)(\Pi_{i=1}^{n}\overline{b_i}\langle read, lock\rangle.\overline{h} \mid \{h\}_1^n.A') \\ B_i =&b_i(read, lock).lock(write, unlock).read(doc).\tau_{B_i}.\overline{write}\langle doc\rangle.\overline{unlock}.B' \end{aligned}$$

This formalization does not specify how A has received the channels *read* and *lock*.

3 Formalizing the Interaction Patterns using Petri Nets

Petri nets are bipartite graphs consisting of places and transitions that are connected via directed edges. Places can contain tokens that can be consumed and produced by transitions. Transitions and places can be assigned to different actors, which is graphically represented by swim lanes.

Tokens represent control flow as well as data flow. Tokens passed from one actor to another will represent messages in our formalizations.

The send / receive pattern can be modeled as shown in figure 1.

Now imagine that several instances of the interaction take place at the same time. In this case more tokens flow through the net. However, there is no information about which tokens belong to the same instance (if we use simple place / transition nets). Therefore, these simple nets do not support correlation.

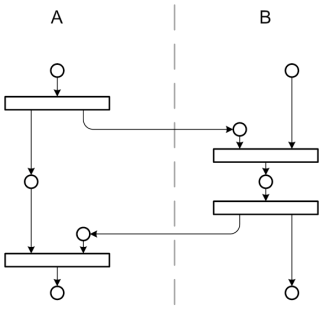


Fig. 1. Send / receive as state / transition net

Figure 2 illustrates how the one-from-many receive pattern in the basic version (stop condition = success condition = n messages received) would be modeled using Petri nets.

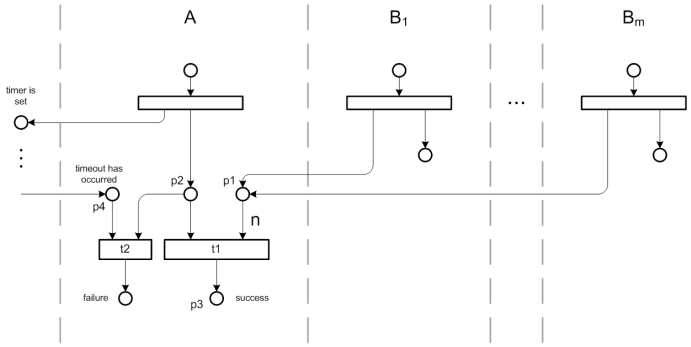


Fig. 2. One-from-many receive

Transitions have to be enabled before they can fire. I.e. n tokens have to be available in place p1 and one token in p2 before t1 can consume these tokens and produce a token on place p3. Once t1 has fired t2 cannot fire any more because there is no token in p2.

We have to stress the *can fire* because t1 is *not forced* to fire. It could happen that t1 could fire but does not and then the time out occurs (a token is produced in place p4) and t2 fires.

This problem also comes up in the racing incoming messages pattern. This pattern defines that the continuation is activated *as soon as* the first message arrives. However, in the case of Petri nets the transition is not forced to consume the first token but could also consume the second instead.

There is a solution to this problem though: Timed Petri nets introduce a notion of temporal ordering of tokens.

Diagram 3 shows a formalization for the request with referral pattern using a place / transition net.

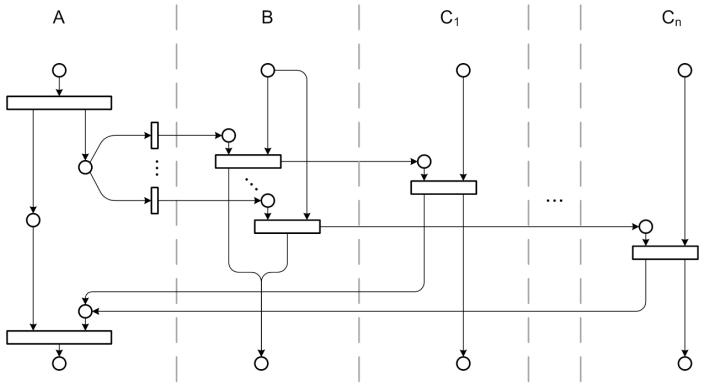


Fig. 3. Request with referral as place / transition net

Since we cannot encode anything in the token we have to indicate the third interaction partner by putting a token into the appropriate place for B. As we might face a huge number of potential interaction partners, we have to model a huge number of places and transitions. This is not feasible.

Another approach is to encode an identifier of the third interaction in the token, which is supported by Colored Petri nets. That way, we introduce something like a bus (see place pBus in diagram 4) where every potential interaction partner is connected to. Guard conditions prevent a transition from consuming tokens that were not meant to be received by the corresponding actor. Diagram 4 illustrates the idea.

We have drastically reduced the number of places and transitions compared to the previous formalization using state / transition nets. Nevertheless, there are still some drawbacks to this approach:

- We still have to model connections between every set of potential interaction partners. In the web there might be thousands or more potential interaction partners in every interaction.
- Due to the different combinations of interaction partners in an interaction the arc and guard expressions might become very complex.
- We cannot model change. Because of the static nature of Petri nets we have to know every potential interaction at design-time. This does not reflect the reality where new potential interaction partners appear and also disappear at run-time.

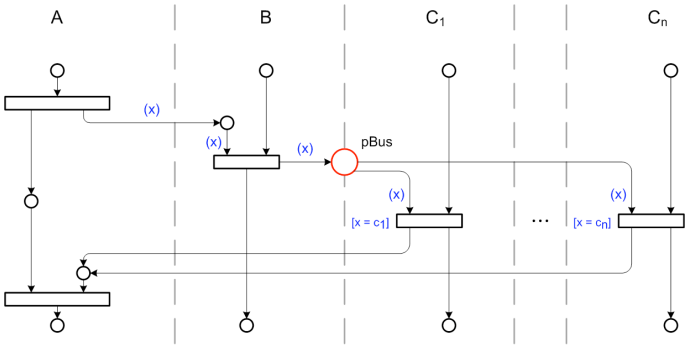


Fig. 4. Request with referral as Colored Petri net

Ten out of the thirteen service interaction patterns incorporate sending messages. In all of these cases it is explicitly stated that the receiver might not be known at design-time. Even using Colored Petri nets we have seen that modeling this mobility with the nets is not feasible when dealing with many potential interaction partners (which has to be assumed).

4 Related work

Recently several papers have been published that deal with formalizing web service choreographies ([11], [12], Busi et al: [13]). All these approaches are based on process algebras (but not the π-calculus). Like in the case of the Web Services Choreography Description Language (WS-CDL, see [14]), all papers only consider one-way- and simple request-response-interactions. This is heavily critized by Barros et al in [2].

Busi et al argue that mobility is not needed for describing service choreographies. They assume that all interaction participants are not at deign-time. They introduce their own process algebra for service orchestration and choreography and show how conformance between a set of orchestrations and a choreography can be proved.

Puhlmann and Weske have formalized all the Workflow Patterns [15] using the π-calculus in [16]. This allows for translating a huge range of service compositions into π-processes.

Puhlmann et al have already sketched in [17] how π-calculus could be used for formalizing service invocations. In this paper the π-representation of correlations has been introduced.

There has not been a formalization of the Service Interaction Patterns so far.

5 Conclusion and Outlook

In this paper we have seen that mobility is a central aspect of the Service Interaction Patterns. As these patterns cover the most common interaction scenarios

in choreographies we can conclude that we need a means to conveniently model mobility when describing choreographies.

This paper has presented how all of the patterns can be formalized using π-calculus. We have also mentioned that theoretically all patterns could be modeled using Petri Nets. However, due to the static nature of Petri nets we have to cope with huge nets in the case of many potential interaction partners.

Therefore, our final conclusion is that π-calculus is better suited for expressing the patterns (see [18] for a definition of suitability).

The formalizations presented in this paper can be the starting point for further work on a complete formal grounding of the intersection of the domains Service-oriented Architectures (SOA) and Business Process Management (BPM) using π-calculus. The very next step would be to show how the formalizations of the Service Interaction Patterns can be integrated with the formalizations of the Workflow Patterns provided in [16].

As already stated in [2] existing choreography description languages such as WS-CDL are not suitable for incorporating complex interactions like we can find them in real-world business scenarios. Therefore, the final goal of having established the Service Interactions Patterns is to create a new pattern-based choreography description language. Once this language has been specified an algorithm for translating choreography descriptions into π-processes would be the next step. That way, an unambiguous specification of a choreography is possible.

As we have seen several interaction patterns can have to be plugged together. E.g. the formalization for pattern 11 shows how send / receive and request with referral can be combined. Thus, our formalizations help to identify which pattern combinations are actually possible.

Once we have both a choreography and corresponding orchestrations available as π-processes we proceed with introducing conformance checking. I.e. we want to verify if the behavior of the individual orchestrations comply to the choreography.

Another area of interest is the investigation of soundness criteria for choreographies. A first starting point could be the soundness criteria for workflows (see [19]). However, there might be other aspects that arise only in cross-organizational business process integration settings that are not covered yet.

References

1. IBM: Web services architecture overview. (2000) http://www-128.ibm.com/developerworks/webservices/library/w-ovr/.
2. Barros, A., Dumas, M., Oaks, P.: A critical overview of the web services choreography description language (ws-cdl). BPTrends Newsletter **3(3)** (2005)
3. Barros, A.P., Dumas, M., ter Hofstede, A.H.M.: Service interaction patterns. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F., eds.: Business Process Management. Volume 3649. (2005) 302–318
4. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana., S.: Busi-

ness process execution language for web services, version 1.1. (2003) http://www-106.ibm.com/developerworks/webservices/library/ws-bpel.
5. van der Aalst, W.M.P., van Hee, K.: Workflow management: models, methods, and systems. MIT Press, Cambridge, MA, USA (2002)
6. Milner, R.: Communicating and Mobile Systems: The π-calculus. Cambridge University Press, Cambridge (1989)
7. Sangiorgi, D., Walker, D.: The π-calculus: A Theory of Mobile Processes. Cambridge University Press, Cambridge (2003)
8. Parrow, J.: An Introduction to the πCalculus. Elsevier (2001)
9. Milner, R., Parrow, J., Walker, D.: A Calculus of Mobile Processes - Part I. (1989)
10. Robben, B., Piessens, F., Joosen, W.: Formalizing correlate - from practice to π. In: Proc. of the BCS-FACS second Northern Formal Methods Workshop. (1997) 1–16
11. Brogi, A., Canal, C., Pimentel, E., Vallecillo, A.: Formalizing web service choreographies. In: Proceedings of First International Workshop on Web Services and Formal Methods, Elsevier (2004)
12. Gorrieri, R., Guidi, C., Lucchi, R.: Reasoning about interaction patterns in choreography. In: M. Bravetti et al. (Eds.): Second International Workshop on Web Services and Formal Methods, LNCS 3670, Springer Verlag (2005) 333348
13. Busi, N., Gorrieri, R., Guidi, C., Lucchi, R., Zavattaro, G.: Choreography and orchestration: A synergic approach for system design. In: B. Benatallah, F. Casati, and P. Traverso (Eds.): ICSOC 2005, LNCS 3826, Springer Verlag (2005) 228–240
14. Kavantzas, N., et al.: Web service choreography description language (ws-cdl). Technical report (2004)
15. van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow patterns. Distributed and Parallel Databases **14(3)** (2003) 5–51
16. Puhlmann, F., Weske, M.: Using the π-calculus for formalizing workflow patterns. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F., eds.: Business Process Management. Volume 3649. (2005) 153–168
17. Overdick, H., Puhlmann, F., Weske, M.: Towards a formal model for agile service discovery and integration. In: Proceedings of the ICSOC Workshop on Dynamic Web Processes (DWP 2005), Amsterdam, The Netherlands (2005)
18. Kiepuszewski, B.: Expressiveness and suitability of languages for control flow modelling in workflows. (2002)
19. van Hee, K., Sidorova, N., Voorhoeve, M.: Soundness and separability of workflow nets in the stepwise refinement approach. In: W. van der Aalst, E. Best (Eds.): Applications and Theory of Petri Nets 2003, LNCS 2679, Springer Verlag (2003) 337–356

Comparing the Capabilities of the Pi-Calculus and Extended Petri Nets Regarding the BPM-Domain

René Freude

Seminar Business Process Management II at the
Hasso-Plattner-Institute for IT Systems Engineering,
at the University of Potsdam
Rene.Freude@student.hpi.uni-potsdam.de

Abstract. Today, the BPM-domain is faced with a lot of heterogeneous and partly proprietary process modeling languages. As this situation is resulting in a lack of interoperability, as well as insufficiency in regard to appropriate analysis techniques, common formal foundations, on which application-specific solutions can build up, have to be found. This paper compares the capabilities for extended Petri nets (i.e. high-level Petri nets extended with self-modification and recursion) and the Pi-calculus to meet these demands. At first, ability for modeling stand-alone processes is evaluated. Since, for the Pi-calculus, it is claimed that all workflow patterns have already been mapped, the focus of this analysis is put on their formalization for extended Petri nets. Afterwards, feasibility for defining process orchestration and choreography is discussed.

1 Introduction

The field of Business Process Management has brought forth a lot of different languages for modeling processes (workflows), as well as distributed processes' interaction (orchestration/choreography). Thereby, many of the present approaches are building up on insufficient and proprietary formal foundations. The problems arising from this situation are multifaceted. On the one hand, it is often unachievable to define formal mappings between the concepts of different modeling languages. This lack of interoperability again complicates the exchange of process specifications among business partners and increases time and costs concerning process integration and, thus, process-orientated software integration. On the other hand, missing formal foundations are inhibiting reliable process execution within arbitrary workflow engines. At the same time, process analysis in terms of soundness often turns out to be unworkable. To bridge the these gaps, a general formal grounding has to be found, on which domain-specific process definition and process interaction languages can build up. According to the current ongoing debate [1, 2], (high-level) Petri nets and the Pi-calculus constitute the most promising candidates for implementing such a back-end concept.

The appliance of high-level Petri nets (i.e. extended with color [3], time [4] and hierarchy) to the challenges of the BPM-domain has already been investigated a lot

[5, 6]. Though, large expressive power and feasibility of many analysis techniques could be proven, there are still a couple drawbacks left. Especially some aspects concerning advanced control flow and orchestration/choreography among arbitrary process instances could not be realized sufficiently, yet. Those deficits, which will be discussed more detailed later, are leading to the idea of taking additional extensions into account. Concretely, this work concentrates on the added value of high-level Petri nets in combination with self-modification [7] and recursion [8]. In contrast to Petri-net-based modeling languages, the Pi-calculus [9], an algebra specification for mobile processes, is a rather new topic for the BPM-domain. Thus, there has been little research on this field, so far. A first proposal for the formalization of all workflow patterns [10] has been given in [11]. Moreover, in [12] an approach for defining service orchestration/choreography has been presented.

The aim of this paper is the comparison of the capabilities of extended Petri nets and the Pi-calculus. Thereby, the introduced weaknesses concerning high-level Petri-Nets are probed for feasibility with using the mentioned additional extensions. At the same time, the results are compared to the proposed Pi-solutions. As it is a proven fact, that data of arbitrary structure is both supported by colored Petri nets and the polyadic Pi-calculus [13], this topic will not be treated in more detail.

The structure of this work is as follows: Firstly, the main concepts of self-modifying nets and recursive Petri nets are introduced in section 2. Thereby, the focus is also put on compatibility issues regarding time, color and hierarchy. While the third section deals with the evaluation of coverage and practicability for modeling workflow patterns, the discussion switches to the field of orchestration and choreography in section 4. Finally a conclusion and an outlook are given.

2 Self-modifying Nets and Recursive Petri Nets

This section provides a short overview of the Petri net extensions self-modification and recursion. The major characteristics of both concepts are introduced, and finally, aspects regarding their compliance to classical high-level Petri Nets are discussed.

2.1 Self-modifying Nets

A self-modifying net is an ordinary place/transition net with a rather small extension concerning arc inscriptions.

Let $P = \{ p_1, p_2, ..., p_n \}$ be the set of places within a Petri net.

Then, an arc inscription may either be denoted by a number – as it is the case for ordinary place/transition nets – or by using a formal polynomial of the form:

$$z + \left(q_1 * p_1 + q_2 * p_2 + ... + q_n * p_n\right) \text{ with } z, q_i \in \left\{x \in N \,|\, 0 \le x < \infty\right\}$$

Finally, when applying the enabling rules of transitions, every variable p_i is substituted by the value of the marking $M(p_i)$, i.e. the number of tokens that are located at the appropriate place.

Such "marking-aware" arc inscriptions are leading to the ability of modeling transition enabling rules, which are not only related to the states of the incoming places but also to the internal state of the whole net. Figure 1 shows a demonstrative example for a concurrent system with the critical states 'p2' and 'p5'. Thereby, the transitions 'A' or 'B' may fire, if and only if the corresponding concurrent thread of control is not in the critical state. A further benefit of self-modifying nets is the support for inhibitor arcs, i.e. an arc weighting that leads to transition enabling, if and only if the according input place contains zero tokens. A possible solution for modeling such inhibitor arcs is shown in figure 2 ('A' is enabled if and only if 'p2' is empty).

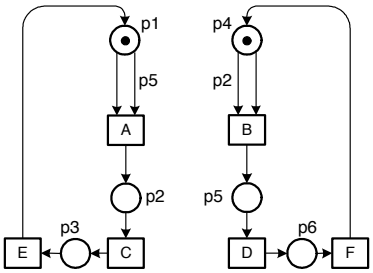


Fig. 1. A simple concurrent system

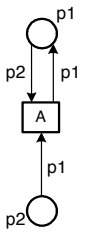


Fig. 2. Modeling inhibitor arcs

Self-modifying nets provide the expressive power of a Turing machine. However, many important soundness properties such as reachability, liveness and boundedness are undecidable. On the other hand, if that extension is only used within isolated blocks (subnets) which soundness can be proved for, soundness properties of the global net would not be affected at all.

2.2 Recursive Petri Nets

Recursive Petri nets are again based on ordinary place/transition nets. Thereby, the essential innovation is a dynamical three of execution threads, which are connected by the fatherhood relation. As illustrated in Figure 3, there exist three different types of transitions:

- An *elementary transition* ('B') fires such as defined for ordinary Petri nets.
- An *abstract transition* ('A') consumes the input tokens according to the enabling rule and starts a new thread of execution with an initial marking assigned to it.
- A *final transition* ('C') terminates the current thread of execution, as well as all descendants. Moreover, the output tokens of the abstract transition which gave birth to the thread are produced.

The extended marking of a recursive Petri net is structured as a tree: Vertices, representing the root thread and all descendant threads of execution, are associated with a certain ordinary marking. An edge, leading from a parental vertex to child

vertex, is associated with the birth-given abstract transition. One possible firing sequence for the net shown in figure 3 is presented in figure 4. Initially, only the root thread of execution is active with one token in place 'p1'. Now, the firing of transition 'B' results in the empty marking for the root thread, as well as in creation of a new child thread, which is initialized with the ordinary marking assigned to 'B'. Finally, if the final transition 'C' both has fired for the child thread and for the root thread, execution terminates. In this case, the extended marking is denoted by an empty tree.

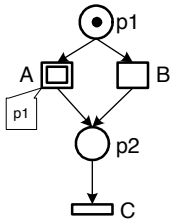


Fig. 3. A simple recursive Petri net

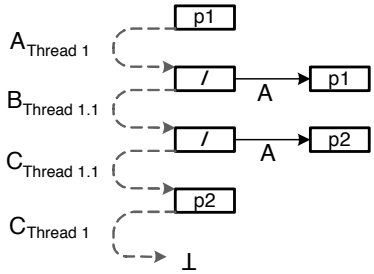


Fig. 4. The extended marking of the net in fig. 3

As a certain thread of execution mainly behaves like an ordinary Petri net and abstract transitions produce exactly one output token, boundedness is still decidable for recursive Petri Nets. The same is true for reachability [8]. Thus, assuming a workflow net for which the output place is reachable, liveness of the strongly connected counterpart (connecting the output place to the input place) can also be inferred.

2.3 Compliance with classic high-level Petri nets

Generalized stochastic Petri nets [4] define additional transition enabling constraints in the form of firing delays. As this is only leading to a more deterministic behavior concerning the resolution of transitions' concurrency but does not have any impact on the remaining execution semantics, the adoption of self-modification and recursion can be considered to be fully compliant.

The hierarchy extension of Petri nets provides feasibility for decomposition of a global net into smaller and independent subnets. This approach is a bit harder to combine with the concepts of self modifying nets and recursive Petri nets. Since there are much more interdependencies between transition execution semantics and the marking of arbitrary places (either the actual marking for self-modification or the initial marking for a recursive thread of execution), the identification of independent subnet candidates turns out to be more complex. Nevertheless, compliance in general is not affected.

Workability of colored Petri nets enriched with self-modification has already been proven in [14]. Thereby, place-related arc expressions are evaluated to the multi-sets of referenced places in the actual marking. Regarding recursive Petri nets, the color extension can be adapted easily, too. As the only additional customization, marking assignments to abstract transitions would have to become "color-aware".

3 Control Flow and Workflow Patterns

In this part of the capability analysis, the ability for modeling single workflows – i.e. a process's internal control flow – should be judged. As a common basis for this evaluation the set of workflow patterns that are defined in [10] is utilized. For the Pi-calculus, it has been claimed, that all appropriate patterns can be supported [11]. In contrast to that, there are still some challenges left concerning the usage of high-level Petri nets [5]. In particular, those patterns that are dealing with advanced synchronization aspects, multiple instance execution or spontaneous cancellation could not be realized satisfactory. In the following, the critical patterns will be formalized by using the extensions self-modification and recursion. Simultaneously, the appropriate results are compared to the proposed Pi-solutions. Thereby, the main criterions are completeness regarding the workflow pattern specification, as well as practicability and a low complexity.

3.1 Advanced Synchronization Patterns

The synchronization of optional branches that are computed in parallel is rather complex. At design time, it is not clear which branches will be activated and, therefore, have to be merged. Regarding the usage of (high-level) Petri nets, there only exist some rather workaround-like solutions that are relying on the forward-communication of activated branches. Thereby, a multi-choice control structure either uses true/false-tokens or alternative bypasses to propagate its execution outcomes. However, this approach works poorly with control cycles and, moreover, is often resulting in a cluttered net structure. On that account, a backtracking mechanism, which is based on the self-modification extension, is proposed for the relevant patterns, in the following.

The workflow net in figure 5 illustrates a simple scenario, which the subsequent studies will refer to: Depending on which transition fires in the initial state, either one of the parallel branches or both will be activated. This behavior is compliant with the Multi Choice pattern. Finally, some kind of synchronization has to be done, which results in enacting activity 'D'.

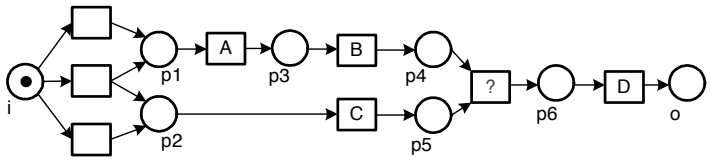


Fig. 5. Parallel execution of optional branches

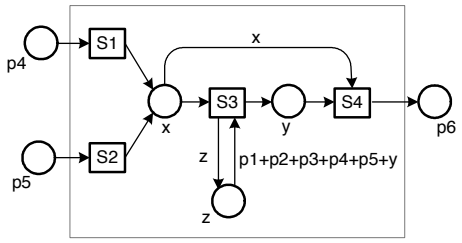
3.1.1 Synchronizing Merge

The Synchronizing Merge waits for all active branches to be finished before they are converged into a single thread.

The left part of figure 6 shows the appropriate formalization for self-modifying nets. Thereby the places 'p4' and 'p5' are exactly the same as in figure 5. If the first

active branch finishes execution, the firing of either ‘S1’ or ‘S2’ results in enactment of the synchronizing block, which then is waiting for all other active branches to be also finished. This is ensured by transition ‘S3’, which is enabled if and only if there are no tokens remaining within the synchronized part of the net. This backtracking-like behavior is achieved by the empty place ‘z’ in combination with the inscription of the outgoing arc that is leading to ‘S3’. Finally, if transition ‘S3’ hast fired and there is a token in place ‘y’, the synchronization block produces one output token for place ‘p6’ by firing transition ‘S4’. At the same time all tokens in place ‘x’, which possibly are remaining from further finished branches, are consumed.

This solution also works for a scenario, in which transition ‘D’ is either leading to the output place or, alternatively, starts a new cycle of execution that is resulting in the repetitive enactment of one of the optional branches. Moreover, boundedness and liveness could be proven by constructing a reachability graph. Thus, if the self-modification extension is only used within the synchronisation block, soundness properties of the global net are remaining decidable.



$$B = \tau_B.\overline{d_1}.O$$
$$C = \tau_c.\overline{d_2}.O$$
$$D = d_1.\tau_D.D' + d_2.\tau_D.D' + ' d_1.d_2.\tau_D.D'$$

Fig. 6. Synchronizing Merge for self-modifying nets (left) and the Pi-calculus (right)

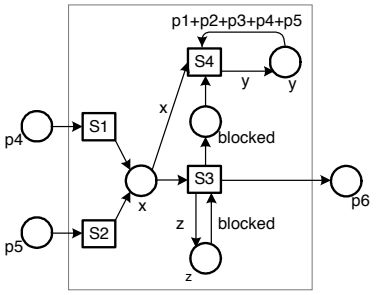
In contrast to that, the Pi-solution, which is shown in the right part of figure 6, does not support a backtracking mechanism and, therefore, is not complete regarding the pattern definition. Since that approach has no capabilities for taking possible pending activities before ‘B’ or ‘C’ into account, it only succeeds for optional branches that are containing not more than one single activity. Some kind of backwards communication might solve this problem; however, complexity would increase a lot.

3.1.2 Discriminator

The Discriminator waits for only one incoming branch to complete before it activates the subsequent activity. Possible further branches are ignored. Finally, if all activated braches have finished, it resets itself.

The left part of figure 7 shows the appropriate formalization for self-modifying nets. If the first branch has completed, the synchronization block is enacted with one token in place ‘x’. Afterwards transition ‘S3’ fires, and an output token for place ‘p6’ is produces immediately. At the same time the Discriminator switches into blocked mode and waits for completion of further braches, which will be ignored. Finally, if there are no more tokens remaining within the synchronized part of the net, transition ‘S4’ is enabled and resets the Discriminator on firing (i.e. the tokens of places ‘x’ and

'blocked' are consumed). As for the Synchronizing Merge, liveness and boundedness regarding the synchronization block could be proven.



$$B = \tau_B.\overline{d_1}.O$$
$$C = \tau_c.\overline{d_2}.O$$
$$D = (\nu h, exec)(D_1 \mid D_2)$$
$$D_1 = d_1.\overline{h}.O + d_2.\overline{h}.O$$
$$D_2 = h.\overline{exec}.h.D \mid exec.\tau_D.D'$$

Fig. 6. Discriminator for self-modifying nets (left) and the Pi-calculus (right)

Again, the corresponding Pi-approach lacks ability for backtracking. Since it is not clear, if there are some pending activities before 'B' or 'C', the Discriminator does not know, how long to wait before a resetting should take place.

3.1.2 N-out-of-M-Join

For the N-out-of-M-Join, the number of to be synchronized branches (as a subset of M optional incoming threads), is arbitrary and denoted by the constant N. Since this pattern constitutes a generalization of the Discriminator, a separate comparison for extended Petri nets and the Pi-calculus is not necessary at this point. It can be derived, that the limitation of the Pi-based Discriminator – resulting from missing backtracking support – is also existent regarding the realization of the N-out-of-M-Join. For the sake of completeness, a formalization approach for self-modifying nets is shown in figure 7.

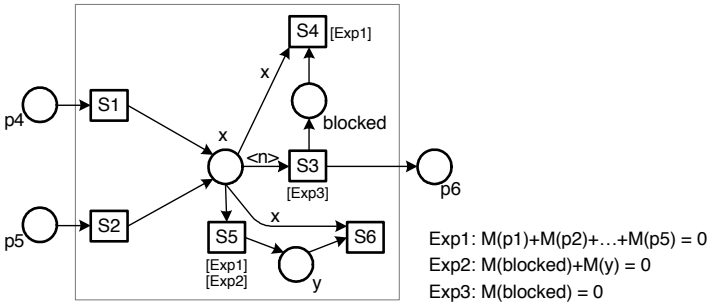


Fig. 7. N-out-of-M-Join for self-modifying nets

Different from the Discriminator, the firing of transition 'S3' (which is resulting in token generation for output place 'p6', as well as blocking mode), occurs if N branches have completed. Additionally transition 'S4' and 'S5' are resetting the block, if less than N branches have completed and there are no tokens remaining within the synchronized part of the net. The guard expressions, which have been attached to some transitions, do not belong to the notation of self-modifying nets. To

provide better clearness, the arc expressions of necessary empty control places (such as place ‘z’ in figure 6) have already been mapped to the appropriate transition enabling constraints.

3.2 Multiple Instance Patterns

Multiple instance execution of certain activities within a process in combination with supplementary thread synchronization is rather hard to handle. One the one hand, it has to be kept track of the number of instances still running; on the other hand multiple instances even might be nested. If there is more than one case in execution at the same time, different case identities have also to be taken into account. Since for the following analysis, exactly one process instance is assumed to be created for each possible case, this aggravating factor does not have to be considered.

(High-level) Petri nets lack support for modeling multiple instances when the concrete number of executions is either unbeknown or decided at runtime. Thus, the appropriate patterns will be formalized by using extended Petri nets. Because of the fact, that the Pi-solutions for handling multiple instances are fully compliant to the pattern definition, only the Petri net approaches have to be discussed in the following.

3.2.1 MI without a-priory Runtime Knowledge with Synchronization

For the formalization of multiple instances without runtime knowledge, two different approaches have been identified. The left part of figure 8 shows the realization with using self-modifying nets in combination with the hierarchy extension. If transition ‘A’, which is supposed to be executable arbitrary times, is firing, the according subnet is entered. For each firing of transition ‘Inc’ one additional instance is triggered. If transition ‘Exit’ fires right after ‘Enter’, no instances are executed.

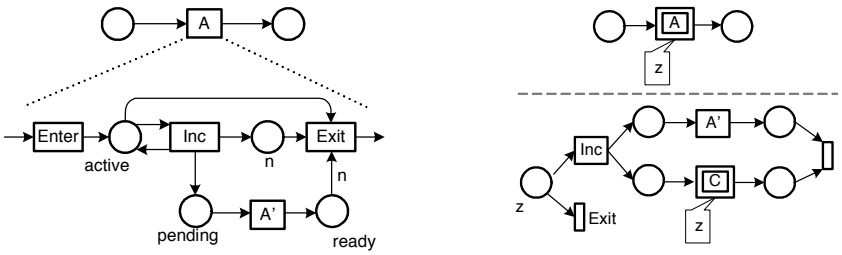


Fig. 8. MI without a-priory RT-knowledge for self modifying nets and recursive Petri nets

This realization includes the deficit, that only one activity instance can be executed at a certain time. This drawback can be avoided by using the recursive Petri net approach which is shown in the right part of figure 8. Thereby, if transition ‘Inc’ fires for a certain thread of control, creation of a new child thread and enactment of the current instance are done in parallel.

3.2.2 MI with a-priory Runtime Knowledge with Synchronization

The formalization for multiple instances with runtime knowledge, which has been done by using self-modification, is quite similar to the last solution. As shown in figure 8, an extra transition 'Run' has been added that may fire, if the number of instances that are to be executed has been determined.

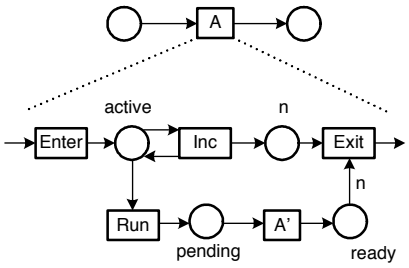


Fig. 8. MI with a-priory RT-knowledge for self modifying nets

3.3 Cancellation Patterns

As cancellation might occur spontaneously, it has to be considered for all those parts of the process, which extraordinary termination should be enabled for. While the Cancel Activity pattern deals with termination of a single activity (what at the same time results in termination of the related thread), appliance of the Cancel Case pattern is leading to the termination of the whole workflow instance.

The Pi-solutions are relying on the triggering of cancellation events, which all appropriate processes provide receptors for. In that way, either a single process or all processes that are belonging to a certain case are triggered and, thus, terminated.

Regarding (high-level) Petri nets, especially the Cancel Case Pattern is difficult to formalize. As some kind of vacuum cleaner for all remaining tokens has to be implemented, the net structure would rather explode. The usage of a final transition as defined by recursive Petri nets would easily provide the demanded behavior and, moreover, is even less complex than the Pi-counterpart.

4 Orchestration and Choreography

Today, there is a steady trend from stand-alone applications toward large integrated system landscapes. Thus, process modeling languages do not only need to provide feasibility for modeling single workflows, but also have to meet the demands for the specification of processes' interaction. Although there has been little research so far, process orchestration and choreography constitutes the field of application, for which the advocates of the Pi-calculus stress the advantages in comparison to Petri-net-based languages. In [12], a model for asynchronous and synchronous interactions between Pi-processes, as well as a concept for handling correlations between arbitrary processes by exchanging names is proposed. The following example shows an appropriate communication scenario for a server 'A' and a client 'B'. In the initial

state, the server is waiting for requests. Thereby, the according channel ‘w’ is known by the client. Thus, one can say that the client holds a reference to the server. By using that channel, the client firstly sends a name ‘c’ to the server, which again acts as a reference for the subsequently reply.

$$A = w(c).(\nu x)(\bar{c}\langle x\rangle).0 \qquad B = (\nu c)(\bar{w}\langle c\rangle.c(x)).\tau_{CI}.0$$

This structural behavior, which is denoted by the term mobility, is not achievable for the usage of ordinary Petri nets. Since arcs between places and transitions are static, all possible connections between partners involved within the interaction have to be modeled in advance. The self-modification extension would not improve this situation. Although arc weightings might dynamically change at runtime, the connection paths are remaining static.

At this point the color extension for Petri nets could be used, to allow the modeling of processes’ correlations at the type level. Thereby, colored tokens could imitate the exchange of names that afterwards act as references. Figure 9 shows the appliance to the introduced interaction example.

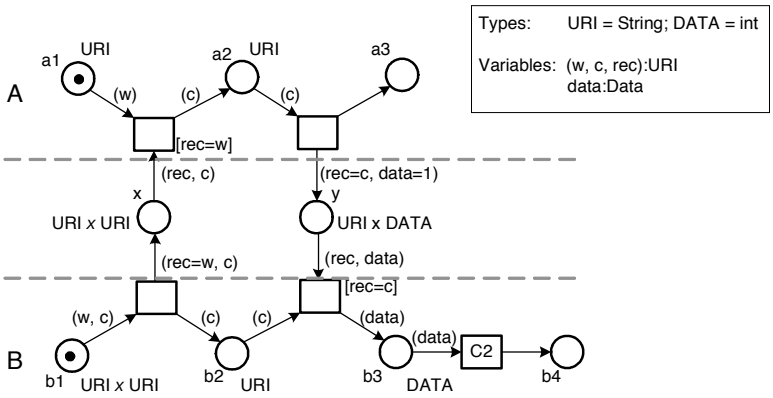


Fig. 9. Handling of correlations by using colored Petri nets

At instantiation time, each subnet ‘A’ and ‘B’ has to be initialized with a starting token that both contains the own address, as well as all references to other processes, that are considered to be known. These references might also be exchanged during communication. Thereby, each interaction step is done by sending a token, which contains an identifier for the addressee in the first place. Guard expressions that are attached to all incoming transitions are deciding the rooting behavior.

Obviously, this approach can not be considered to be practical at all. One the one hand, the resulting net turns out to be highly complex; on the other hand all process instances have to be connected to each other at the instance level, anyway. Moreover, more advanced interaction scenarios are hard or impossible to realize. Even a simple broadcast would lead to serious difficulties.

5 Conclusion

In this paper, the capabilities of extended Petri nets (i.e. high-level Petri-nets extended with self-modification and recursion) and the Pi-calculus have been compared. Thereby, it turned out that, depending on the scope of evaluation, each of the concepts has certain strengths but, at the same time, also includes a couple of weaknesses.

When considering the modeling of control flow within a certain process, extended Petri nets provide the most applicable foundations. Almost all workflow patterns, which could not be realized for (high-level) Petri nets so far, have been formalized by adopting the semantics of self-modifying nets. Recursive Petri nets may offer some added value when dealing with multiple instances, nevertheless, this extension plays a minor role in this regard. At the same time, it has been proven, that self-modification does not always imply elimination of decidability for soundness properties such as liveness or boundedness. If the usage of this extension is restricted to a certain isolated block – as it has been done for the formalized patterns – and soundness for that block can be evidenced on the basis of a reachability graph, the soundness properties of the global process can still be analyzed by decomposition into safe building blocks or the creation of a global reachability graph.

Though feasibility and completeness regarding the workflow patterns has also been claimed for the Pi-calculus, some difficulties in terms of advanced synchronization aspects have bee identified (see section 3.1). Since the corresponding pattern formalizations do not support the backtracking for possible pending activities, some application scenarios cannot be covered. The weaknesses, identified for the Pi calculus, are mostly resulting from loose and event-based couplings between single Pi-processes. This structural property, on the other hand, is leading to a huge expressive power in regard to the definition of orchestration and choreography. Even though colored Petri nets might be also used to model correlated communication between distributed processes, after all, the appropriate solutions turn out to be unpractical when the number of communication partners is arbitrary.

Taking the evaluation results into account, a universal formal foundation for business process languages cannot be proposed. However, depending on the concrete field of appliance, both concepts provide a lot of potential to act as some kind of backend-model for domain-specific languages and, therefore, may contribute to a more homogenous and interoperable BPM-landscape. In this manner extended Petri nets could constitute the formal grounding for the specification of single workflows and processes' internals respectively, while the Pi-calculus-based concepts could be established for the definition of public processes, i.e. interaction contracts regarding distributed processes.

References

1. van der Aalst, W.M.P.: Pi calculus versus petri nets: Let us eat "humble pie" rather than further inflate the "pi hype". (http://is.tm.tue.nl/research/patterns/ download/pi-hype.pdf (May 31, 2005))

2. van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M., Russel, N., Verbeek, H.M.W., Wohed, P.: Life After BPEL?. In: Bravetti, M., Kloul, L., Zavattaro, G. (eds.): WS-FM 2005, Lecture Notes in Computer Science, Volume 3670. Springer-Verlag, Berlin (2005) 35-50
3. Jensen, K.: An Introduction to the Theoretical Aspects of Coloured Petri Nets. In A Decade of Concurrency, Reflections and Perspectives. In: Bakker, J.W., Roever, W.P., Rozenberg, G. (eds.): REX School/Symposium 1993, Lecture Notes in Computer Science, Volume 803. Springer-Verlag, London (1993) 230-272
4. Ajmone Marsan, M., Balbo, G., Conte, G. et al.: Modelling with Generalized Stochastic Petri Nets. Wiley series in parallel computing. Wiley, New York, 1995
5. van der Aalst, W.M.P., ter Hofstede, A.H.M.: Workflow Patterns: On the Expressive Power of (Petri-net-based) Workflow Languages. In: Jensen, K., (eds.): Fourth Workshop on the Practical Use of Coloured Petri Nets and CPN Tools 2002, Volume 560, University of Aarhus, Arhus (2002), 1-20
6. Mulyar, N., van der Aalst, W.M.P.: Patterns in Colored Petri Nets. In: BETA Working Paper Series, University of Technology, Eindhoven (2005)
7. Valk, R.: Self-Modifying Nets, a Natural Extension of Petri Nets. In: Ausiello, G., Böhm, C. (eds.): Proceedings of the Fifth Colloquium on Automata, Languages and Programming 1987, Lecture Notes in Computer Science, Volume 62, Springer-Verlag, London (1987) 464-476
8. Haddad, S., Poitrenaud, D.: Theoretical Aspects of Recursive Petri Nets. In: Donatelli, S., Kleijn, H.C. (eds.): Proceedings of the 20th international Conference on Application and theory of Petri Nets 1999, Lecture Notes in Computer Science, Volume 1639, Springer-Verlag, London (1999) 228-247
9. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, Part I/II. Information and Computation 100 (1992) 1–77
10. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.: Workflow patterns. Distributed and Parallel Databases 14 (2003) 5–51
11. Puhlmann, F., Weske, M.: Using the Pi-Calculus for Formalizing Workflow Patterns. In van der Aalst, W., Benatallah, B., Casati, F., eds.: BPM 2005, volume 3649 of LNCS, Berlin, Springer-Verlag (2005) 153–168
12. Overdick, H., Puhlmann, F., Weske, M.: Towards a Formal Model for Agile Service Discovery and Integration. In Proceedings of the ICSOC Workshop on Dynamic Web Processes (DWP 2005), Amsterdam, December 2005
13. Milner, R.: The polyadic π–Calculus: A tutorial. In Bauer, F.L., Brauer, W., Schwichtenberg, H., (eds.): Logic and Algebra of Specification, Springer-Verlag, Berlin (1993) 203–246
14. Valk, R.: Extending S-invariants for Coloured and Self-modifying Nets, Universität Hamburg, Hamburg, (1993)

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.

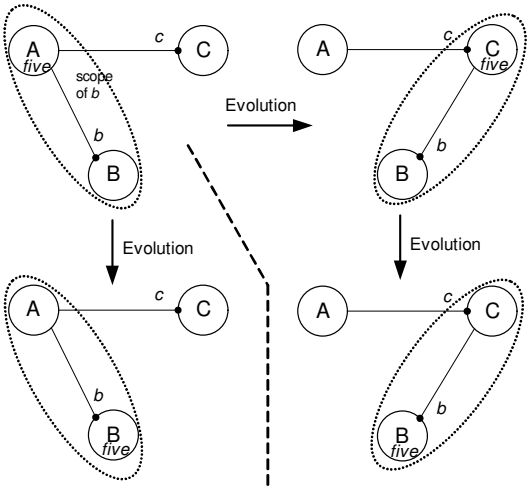


Fig. 1. Pi-Calculus Example of a System of Processes

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.

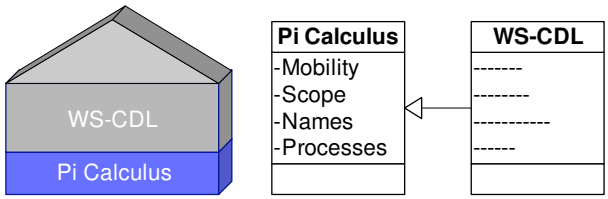


Fig. 3. Pi-Calculus is the basis for WS-CDL

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.

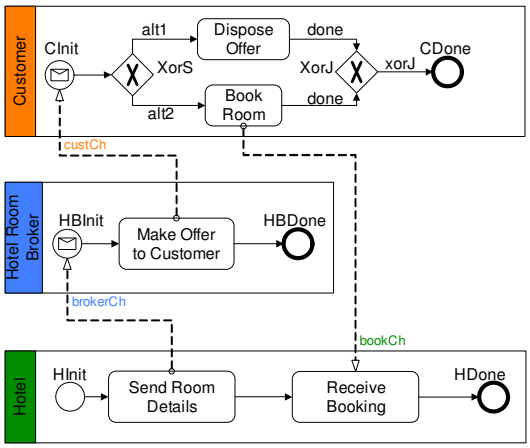


Fig. 4. Simple choreography example

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 $bookCh_z$.

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

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>
```

Semi-automated service composition

Anna Ploskonos

Hasso-Plattner-Institute for IT Systems Engineering, Postfach 900460,
14440 Potsdam, Germany
anna.ploskonos@student.hpi.uni-potsdam.de

Abstract. Semi-automated service composition indicates that a composition system and human can work together to compose web services for given requests. The composition system should be able to analyze a partial workflow created by the user, notify the user about issues that have to be resolved in the current situation and suggest the user what actions could be taken next. Humans make the decision regarding the service composition. In this paper we consider interactive workflow composition tool developed by Jihie Kim, Yolanda Gil, and Marc Spraragen, and compare with other works which use semantic description based on Web Service Modeling Ontology (WSMO) or Web Ontology Language (OWL).

1 Introduction

Web services become an essential technique that allows reusing components and getting new application by composing web services which describe scientific or business applications.

Today web service composition is a complex task because a huge amount of web services are available on the web, and it is already difficult to make an appropriate composition manually. Another difficulty is made by dynamic business environment which demands creating and updating of web service composition on fly. The composition system should immediately detect changes and make appropriate updates in the workflow. The lack of de facto standard for semantic service specification is also a reason of the composition problem because a lot of web services are developed by different organizations using different conceptual models. It requires a utilization of relevant semantic information in creating web service composition. Thus, the web service environment is highly complex and it is not practicable to generate the service composition in an automatic way. Usually, the highly automated method suits for generating a skeleton of the service composition which will be specified by users later. The semi-automated composition system allows users to make web service composition and supports them with suggestions. The system takes into account user's preferences and requirements, and based on them makes suggestions how to combine web services. Thus, the system allows the user to make the decision regarding the service composition.

By making a service composition the user may use following strategies: 1) top-down selection of components, for instance, the user doesn't have explicit description of the desired result, they may start making a composition from abstract model and then specify it; 2) result-based selection of components, the user has explicit description of the desired result and would like to simulate the situation that leads to this desired result; 3) situation-based of components, the user has only description of initial states and wants to get a simulating model that describes possible results.

Thus, the goal of semi-automated service composition system is to support users in creating service composition by giving a possibility to filter, select web services and by providing intelligent suggestions. The service composition system should be able to analyze a partial service composition created by the user, notify the user of issues that have to be resolved in the current situation and suggest the user what actions could be taken next.

This paper is organized in the following manner. In Section 2, we describe a motivating example for current research. In Section 3, we give the background – several existing tools for semi-automated service composition. In Section 4, we discuss one of these tools in detail which called Composition Analysis Tool developed by Jihie Kim, Yolanda Gil, and Marc Spraragen. And finally, Section 5 concludes the work. Throughout this paper, we use the example we described in Section 2 to illustrate some concepts used in different techniques.

2 Motivating example

The common example for web service composition is online shopping, the core services range from product search, ordering, payment and shipment.

As a working example we take book buying domain. The user has a goal - buying a certain book. First of all she/he wants to make a composition from available services on the web such as book finding service, book ordering service, payment service, shipment service and currency exchange service. The book finding service is responsible for searching a book given book name. The result will be a price in USD. When the user wants to see the price in EURO the currency converter service has to be instantiated. The currency converter service is responsible for transformation the price in USD into the price in EURO. The book ordering service is responsible for ordering a book given book name, author and quantity of required books. The payment service makes bank transaction given cost. The outcome is a receipt of the operation. The shipping service looks for appropriate delivery company given seller', buyer's addresses and makes delivery order to deliver the book. We would like to mention that our working example is intentionally made simpler in order to keep simplicity of the presentation.

3 Related work

In the following we give an overview on several existing tools for semi-automated service composition. First of all we consider required framework for service composition system. The abstract model of the framework for service composition system is proposed in [1] and illustrated in Figure 1.

According to this work the composition system should contain following components: translator, process generator, evaluator, execution engine and service repository. The translator translates between the external languages used by users to express what they want in easy manner and the internal languages, for example, logical programming languages used by the process generator. The process generator makes a process model (plans) by composing the available services from the service repository to fulfill the user's request. The process model contains a set of selected services with the control and data among them. The evaluator evaluates all plans using non-functional attributes and proposes the best one for execution. The execution engine executes the plan and returns the result to the requestor.

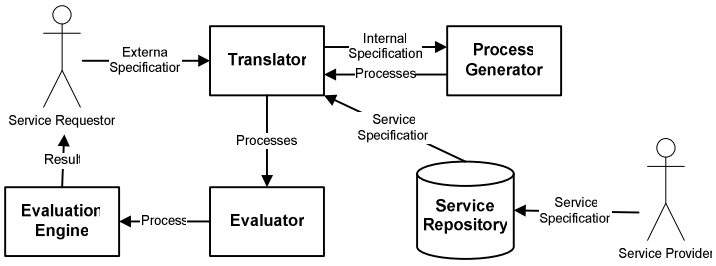


Figure 1. Framework of service composition system.

Semi-automated service composition indicates that the process generator and human can work together to generate the composition of web services for given requests. There are a lot of developed approaches used by the process generator to compose web services. Most of them are related to AI planning and deductive theorem proving. In this paper we consider such approaches:

- Semi-automatic service composition in IRS-III (Internet Reasoning System);
- Semi-automatic service composition using OWL;
- Composition Analysis Tool.

The next two sections describe each of these first two techniques in turn. The CAT is described in section 4 in more detail; creators use another way of presentation the semi-automated service composition.

3.1 Semi-automatic service composition in IRS-III

This approach is detailed described in [2]. Semi-automated service composition tool in IRS-III is a graphical tool that supports users during the process of designing the service composition. IRS-III is a framework for Semantic Web Services based on WSMO specification. The IRS-III automatically transforms programming code into a web service and supports capability-driven service discovery and invocation.

The core of this approach is WSMO description that combines such components: goals, web services, ontologies and mediators. The goals are defined as pre-condition which describes the state of the desired information space of a given web service and post-condition which describes the state of the world after execution of a given web service. The web service descriptions are used for representation functional behaviours of web services such as how web services communicate – choreography view and how they are composed – orchestration view. Ontologies are described in terms of concepts, axioms, relations and instances. The mediators play the main role in the service composition; they specify interoperability mechanisms and link goals, web services and ontologies. More information about WSMO can be found in [3].

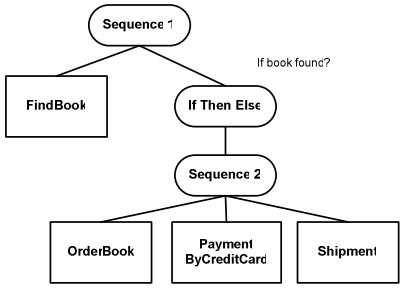


Figure 2. Example of a composition tree for book buying domain.

The composition tool uses web composition model defined as WSMO orchestration and as two types of components: control components (concurrent, sequence, while, if then else) and service components. Let's consider a composition model for book buying example. The composition tree for this example is depicted in Figure 2. There are two sequence controls. First sequence combines finding book component and IfThen-Else control which means if found book, then should be performed next sequence. The next sequence contains of ordering book, payment book and shipment. Finally, the payment operation charges the card and the shipment operation arranges sending the book from the bookstore to the client's address.

By making a service composition the user can: 1) add/remove goals; 2) add/remove control operations; 3) select mediators; 4) receive recommendations according to the automatic match of inputs and outputs goals; and 5) call discovery features. Ones the service composition is defined, the composition tool instantiates orchestration engine for executing this workflow.

3.5 Semi-automatic service composition using OWL

Semi-automated service composition using semantic descriptions is presented by Sirin and others [4]. This approach works based on functional and non-functional attributes presented by OWL classes. OWL-S service's description has three parts: service profile, process model, and grounding. The service profile specifies input and output types, preconditions and effects. The process model describes how the service works, like BPRL4WS. And the grounding specifies information for executing the services by mapping from OWL-S to WSDL.

The composition system regarding this approach has following components: inference engine, composer and graphical interface, though which users can establish their preferences for the workflow and make the service composition by selecting components. The inference engine stores service advertisements and process requests and performs the role of OWL reasoner. OWL reasoner matches two services when an output parameter of one service is the same OWL class or subclass of an input parameter of another service. If more than one match is found, the system filters the services based on the non-functional attributes. The composer generates the service composition by communicating with inference engine and presents users the possible choices at each step.

4 Composition Analysis Tool

In this section we consider interactive workflow composition tool called as Composition Analysis Tool (CAT) created by Jihie Kim, Yolanda Gil, and Marc Spraragen. The description of this approach could be found in [5], [6], [7]. The idea is the user specifies input data for overall composition and desired results, and makes the service composition by selecting components and establishing links between them. The CAT checks the workflow created by the user on consist of errors and suggests next steps.

This approach uses knowledge-based representations of components that describe the relations and constraints among components by the task ontology and the domain ontology. Description logic is utilized in order to reason about these ontologies. The tool exploits errorscan algorithm to find errors in the service composition and to generate suggestions.

Let's look at how the system works. The system combines knowledge base, errorscan, interaction manager and planner. The interaction manager plays a role of the mediator between users and the system. The knowledge base consists of descriptions of the task ontology and the domain ontology, the explanations of these terms comes soon. So, the user makes steps (add component or remove component, add link or remove link) towards to the creation of the service composition; the interaction manager passes this workflow to the errorscan, and receives a list of errors and suggestions how to fix these errors. The user selects appropriate suggestion and the process of the service composition continues until completion the service composition without errors. The planner receives the workflow created by the user as initial states, and the

goals, which the user wants to achieve, and returns the plan with a set of tasks and links among them. The CAT architecture is presented in Figure 3.

In the next sections we will discuss the knowledge base, the errorscan algorithm and the planner in detail.

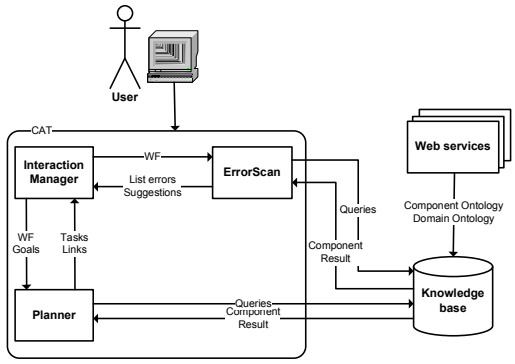


Figure 3. CAT architecture.

4.1 Knowledge Base

Knowledge-based descriptions of components consist of task ontology and domain ontology. The task ontology is utilized to describe abstract types of operations and services and built based on case frames, where verbs are qualified by cases that reflect their linguistic usage. For instance, the book buying domain can be presented as a set of atomic task types such as "OrderBook", "FindBookPriceInUsd", "PaymentBy-CreditCard", "DeliveryBook", "CurrencyConverterFromUsdToEuro" and so on.

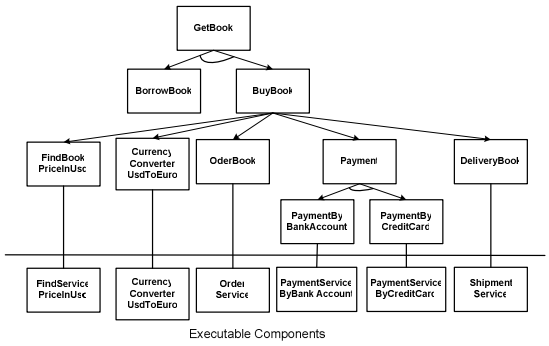


Figure 4. Example of the task ontology.

Each atomic task type is related to a web service as executable component. Atomic tasks can be combined into abstract class, for example, "GetBook" task type consists of "BorrowBbook" task type and "BuyBook" task type. The partial task ontology for

book buying domain is depicted in Figure 4. The more information about how to build the task ontology can be found in [8].

The domain ontology is utilized to specify data type in WDSL description. Input and output parameters from WSDL are mapped to data types in the domain ontology. The data types are presented in the domain ontology as Loom classes, this approach of presentation is described in [9]. The example of the domain ontology for book buying domain is presented in Figure 5.

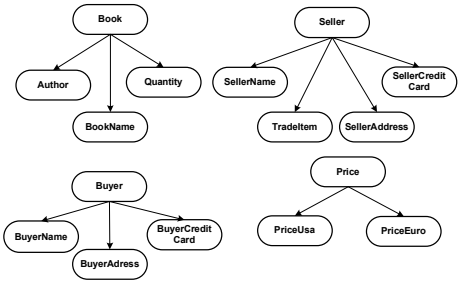


Figure 5. Example of the domain ontology.

After that we establish relationships between the task ontology and the domain ontology by relating inputs and outputs. Thus, each operation is represented with a task type in the task ontology and its input and output parameters are represented with data types in the domain ontology. Regarding our example the task type "FindBook-PriceInUsd" has "BookName" as the input data type and "PriceUsd" as the output data type from the domain ontology.

Note that because the system has the task ontology that describes high-level task types as well as specific tasks that are mapped to services, users can start making service composition from a high-level description of what they want without knowing the details about available operations.

The system refers to the knowledge base (KB) via the following queries:

- components(): returns a set of available components defined in the KB,
- data-types(): returns a set of data types defined in the KB,
- input-parameters(c): returns input parameters of component c,
- output-parameters(c): returns output parameters of component c,
- executable(c): returns false if c is not an executable component,
- range(c, p): returns a class defined as the range of parameter p of c,
- subsumes(t1, t2): returns true if class t1 subsumes class t2 in the KB,
- component-with-output-data-type(t): returns a set of components c where c ∋ components(), ∃ p ∋ output-parameter(c) and subsumes(range(c, p), t), t is a data type,

– component-with-input-data-type(t): returns a set of components c where c ∋ components(), ∃ p ∋ input-parameters(c) and subsumes(range(c, p), t), t is a data type,

Examples:
input-parameters(FindBookPriceInUsa) = {BookName};
range(FindBookPriceInUsa, PriceUsd) = Price;
subsumes(GetBook, BuyBook)= true;
component-with-input-data-type(PriceUsd) ={CurrencyConverterFrmUsdToEuro}.

4.2 ErrorScan algorithm

This section shows how the error algorithm is used in helping users construct a service composition. A workflow (W) is considered as a tuple <C,L,I,G> where C is a set of workflow components, L is a set of links, I is a set of initial-input components, G is a set of end-result components. Each service composition should meet workflow properties: tasked, satisfied, grounded, justified, consistent and unique. Tasked workflow means that the workflow contains one or more end-results. Satisfied — all input parameters of all components are provided by output parameters of other components, or by default values, or as user inputs. Grounded — all components are executable, there are not abstract components. Justified —all output parameters of all components are linked to other components or to end results. Consistent — each link connects an output of one component to the input parameter of another component, where the output is subsumed by the input. It means that the input data type should be more general in the domain ontology. Unique, there is no link or component is redundant with any other one. The workflow is considered as complete if it is satisfied and tasked, and the workflow is correct if it is complete, grounded, justified, consistent and unique.

The process of service composition is guided by workflow properties. The errorscan algorithm utilizes workflow properties to find errors in the service composition and refers to the knowledge base to find appropriate suggestions how to fix errors. Thus, the input for the errorscan is a partial workflow and the output is a list of errors and suggestions. First, the errorscan checks whether the workflow is tasked or not. If it is not tasked, the system makes suggestions from possible end-results. Then the system checks each component within workflow for the purpose of justified, grounded and satisfied. If the component is not justified, the system suggests remove the components and its links or add a link to another component that is already justified. If the component is not grounded, the system proposes to choose the component form executable components. If the input parameter of the component is not satisfied, the errorscan returns from the knowledge base or from the workflow a list of components that have outputs that subsumed by this input or suggests to use parameter with default value or enter a value manually. After that the system makes verification for each link within workflow for the purpose of consistent and unique. If the link is not consistent, the system proposes to remove the link, or fix the link by interposing and linking to appropriate component. If the link is not unique, the errorscan suggests remove the link. Hence, if the service composition meets all of these requirements, it can be considered as complete.

4.3 Sample case

Now we describe how could be used this approach in making service composition for book buying domain. For example, the user wants to find a certain book and the price has to be in euro. She/he defines the input parameter as book and output parameter as price in euro for overall composition. These paramagnets are not linked yet as shown in Figure 6. In the first composition the system found that the output parameter of the initial-input component is not justified. As was mentioned before justified property means that the output parameter of the component should be linked to another component or to an end-result. In this case the system would suggest regarding the errorscan algorithm to link to another component. The component should be found in the knowledge base by satisfying the rule – output parameter of one component is subsumed by input parameter of another component or it's the same subclass in the domain ontology. The errorscan proposes to add a component "GetBook" while it suits to output parameter of the initial component. And there is also another error in the composition. The input parameter of end-result component is not satisfied (an input parameter of one component should be provide by another component, by default values, or as user inputs). The errorscan would suggest adding component "CurrencyConverterFromUsdToEuro" because of it suits to input parameter of the end-result component. This workflow is tasked while there is an end-result for overall composition.

Figure 6. Service composition: input and output parameters for overall composition.

In figure 7 shows that our user has chosen "GetBook" from privies suggestions. In this composition the system according to the errorscan algorithm detects such errors: the end-result of the composition is not satisfied, the output parameter of the component "GetBook" is not justified and the component "GetBook" is not grounded. In case of grounded property the system suggests specifying component "GetBook" to "BuyBook" or "BorrowBook". If the user chooses "BuyBook" the errorscan proposes to specify component until getting a grounded component "FindBookPriceInUsd"

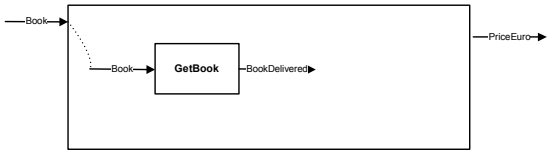


Figure 7. Service composition: is not grounded, not satisfied and not justified.

In the next step the user has chosen "FindBookPriceInUsd" and added a link between two parameters "PriceInUsd" and "PriceInEuro" as shown in the Figure 8. The

system detects that link from the component “FindBookPriceInUsa” to the end-result is not consistent while the output parameter is not subsumed by the input parameter in the domain otology. In this case the errorscan would propose to add a component “CurrencyConverterFromUsdToEuro” and two links.

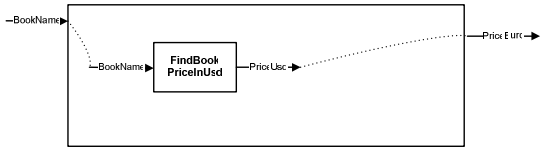


Figure 8. Service composition: the link is not consistent.

After adding the component “CurrencyConverterFromUsdToEuro” the all properties are satisfied. The result of well defined service composition is presented in Figure 9 that meets all requirements for correct and complete service composition.

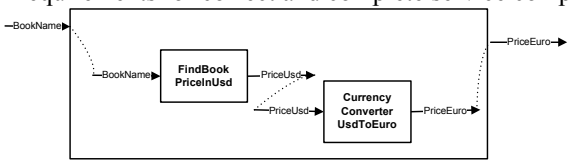


Figure 9. Service composition: correct and complete.

The user can continue such process of combining web services until all the expected results are achieved, all necessary input data are provided, there are not inconsistent links, and all components become grounded.

4.4 Planner

Semi-automated service composition requires user interactions; however, it may become exhausting work if a large number of tasks are needed. For this purpose, it is useful to utilize automated planning techniques to generate a skeleton of the service composition.

The workflow without inconsistent links and unjustified components created by the user is passed to the planner as planning problem and returned as a set of tasks with links among them. If the planner reaches more than one possible task, the planner will return the most abstract class of possible candidates. After that the user has to specify abstract classes by selecting appreciate ones. Accordingly, the system allows users making the final decision regarding the service composition.

5 Conclusions

In this work we have discussed semi-automated service composition on the assumption of such system should be able to analyze a workflow created by the user, notify about what issues have to be resolved in the current situation and suggest what actions could be taken next. As the example of semi-automated service composition tool we considered CAT. Its main components are the knowledge base, the errorscan algorithm and the automatic planner. The key element of this approach is the knowledge base; its advantage is the division into the task ontology and the domain ontology that describe high level as well as specific level. Users can start making service composition from a high-level description of what they want without knowing the details, the system will assist users in making the service composition and leads them to desired results. The process of service composition is guided by workflow properties like: tasked, satisfied, grounded, justified, consistent and unique which should be proofed. The idea of using automated panning techniques is effective as well because it can reduce unnecessary user's iteration within the creation of service compositions which consist of a lot of task by generating a sketch of created workflow. The disadvantage of this approach is usage only merging input and output parameters without control structures like concurrent, while, if then else. The CAT doesn't utilize existing standards for semantic specification such as OWL or WSDL.

We have discussed other approaches for semi-automated service composition: semi-automatic service composition in IRS-III and semi-automatic service composition using semantic descriptions based on OWL. The first work is based on WSDL specification, the process of the composition regarding this approach is guided by mediators. The user has possibility to add or remove goals; to add or remove control operations; and select mediators. The system gives recommendations according to the automatic match of inputs and outputs goals. Comparing with CAT this approach provides control structures of the workflow as was shown in the section 3.1.

The last considered approach is semi-automatic service composition using semantic descriptions based on OWL. The process of the composition within this approach is guided by OWL-reasoner taking into account functional and non-functional attributes presented by OWL-classes.

Each described tool has two following main components: knowledge base and inference engine. The challenge for the knowledge base is rich ontology which will help the reasoner to find better answers for requests. The suggestions given by the inference engine can be improved by taking into account past user activities. By using such information the inference engine can reorder the proposed choices or presents a composition similar to the previous ones.

References

1. Jinghai Rao and Xiaomeng Su: A Survey of Automated Web Service Composition Methods. In Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition, SWSWPC 2004, San Diego, California, USA (2004)

2. Farshad Hakimpour, Denilson Sell, Liliana Cabral, John Domingue and Enrico Motta: Semantic Web Service Composition in IRS-III: The Structured Approach. 7th IEEE International Conference on E-Commerce Technology (CEC 2005), IEEE Computer Society 2005, ISBN 0-7695-2277-7 (2005) 484-487
3. WSMO, "Web Service Modeling Ontology–Standard", http://www.wsmo.org/2004/d2/, 2005
4. Evren Sirin, Bijan Parsia, and James Hendler: Filtering and Selecting Semantic Web Services with Interactive Composition Techniques. IEEE Intelligent Systems, (2004) 19(4): 42-49
5. Jihie Kim, Marc Spraragen, and Yolanda Gil: A Knowledge-Based Approach to Interactive Workflow Composition. Proc. Int'l Conf. Automated Planning and Scheduling, Workshop Planning and Scheduling for Web and Grid Services, AAAI Press (2004)
6. Jihie Kim, Marc Spraragen, and Yolanda Gil: An Intelligent Assistant for Interactive Workflow Composition. In proceedings of the 2004 International Conference on Intelligent User Interfaces (IUI), Madeira Islands, Portugal, (2004)
7. Jihie Kim and Yolanda Gil: Towards Interactive Composition of SemanticWeb Services. In Semantic Web Services - 2004 AAAI Spring Symposium (2004)
8. Yolanda Gil and Jim Blythe: How can a structured representation of capabilities help in planning? In AAAI 2000 Workshop on Representational Issues for Real-world Planning Systems. Austin, TX (2000)
9. Robert MacGregor and Mark It. Burstein: Using a description classifier to enhance knowledge representation. IEEE Expert: Intelligent Systems and Their Applications (1991) 6(3): 41 - 46

Mixed-Initiative Use Cases for Semi-Automated Service Composition: A Survey

Jan Schaffner

Hasso-Plattner-Institute for Software Systems Engineering,
Prof.-Dr.-Helmert-Str. 2-3,
14482 Potsdam, Germany,
jan.schaffner@student.hpi.uni-potsdam.de

Abstract. Semi-automated service composition with mixed initiative interactions, where both user and machine jointly contribute to the creation of composed services, is currently subject to intensive research. We propose a definition of the term "semi-automated composition", closing the gap that a comprehensive definition is not available at present. We give an overview over recent research approaches by presenting four different semi-automated service composition tools. As the main contribution of this paper, we introduce three mixed initiative use cases characteristic for semi-automated composition, which we have extracted and generalized from the presented approaches and then extended. Based on these use cases and additional distinctive properties, we give a qualitative evaluation of the presented approaches.

1 Introduction

Web services have been established as a promising approach to provide value added functionalities across organizational borders. Businesses use Web services as a mean to invoke intra-enterprise processes as well as processes including multiple business parties. The actual activities in these processes are Web service operations.

Business processes can therefore be modeled as compositions of Web services. At present, these compositions are created manually: A domain expert creates a static process model which can be translated into an executable language (i.e., WS-BPEL [1]).

That fact that specialists are required to create composed services derives from the complexity of this task:

At design time, the modeler has to anticipate all possible cases that shall be handled by the process he is working on: All imaginable alternative paths must be specified as well as all possible failures must be considered. The modeler has to interpret the names, interfaces and - if applicable - textual descriptions of the services he or she uses in the composition in order to understand their capabilities and nonfunctional properties. This is a prerequisite for the delicate task of correctly defining both the control and data flow among the services. Due

to this complexity it is likely that the modeler introduces faults into the service composition, making conventional service compositions prone to errors.

Furthermore, conventional service compositions are inflexible, because they contain concrete service instances that are to be called at runtime. This is problematic as services may have become unavailable when a composition is enacted.

Handcrafted service compositions are therefore seldom optimal as they contain tradeoffs. They are likely to become complex and inflexible, and are thus difficult and expensive to maintain.

In recent years, the above reasons have been serving as a rationale to introduce automated planners. These systems create executable plans for the individual cases at runtime, which opposes the idea of creating composed services that cover as many cases as possible. The plans are produced in a fully automatic fashion, based on domain knowledge (i.e., ontologies) and semantic service descriptions.

While automated planners are able to reduce complexity, inflexibility and error-proneness akin to the creation of composed services, several drawbacks can be identified: Automated planning relies on the availability of complete formal representations of the domain knowledge and the "state of the world".

The task of formally specifying a domain in sufficient fidelity so that it can be used for automated planning presents a tremendous challenge. If the domain knowledge is incomplete, an automated planner might not be able to produce a plan. In contrast, a human planner can draw upon his experience with a specific domain when he or she creates a composed service.

In a planning problem, the state of the world is represented by the initial state, which serves as a starting point for automated planners to find a path to a supplied goal. It is impossible to encode the complete world state in the initial state of a planning problem. Therefore, matchmakers will always plan with limited information about the world state, which can result in failing to deliver a plan.

The incorporation of the matchmaking technologies as used by automated planners into a semi-automated modeling tool for creating service compositions has several advantages: On the one hand, the problems of complexity, inflexibility and error-proneness of the created service compositions can be reduced or eliminated by building new mixed initiative use cases on top of the Semantic Web technologies used by automated planners. On the other hand, the modeler can rely on his or her experience with composing services. The modeler also has the opportunity to incrementally learn from the scenario and to refine his or her goals while developing the plan. This opposes the problem of planning with incomplete information faced by fully automated planning environments.

Moreover, the fact that fully automated service composition methods do not require a human in the loop poses an organizational and juridical impediment: It may be desirable that a concrete person is responsible for a particular business process. As this lowers the industry acceptance of automated planning techniques, their transition from research to industry is progressing slowly.

For the above reasons, semi-automated service composition is a strongly researched topic at present. However, a general definition of the term "semi-automated composition" is missing. There is also no overview of existing approaches available, neither do we have common understanding of the functionality that is characteristic for semi-automated composition. The goal of this paper is to fill this gap and to address these shortcomings.

The remainder of this work is organized as follows: In section 2 we will give a definition of the term "semi-automated composition". Section 3 will give an overview over current research efforts in the field of semi-automated service composition. In section 4 we will introduce three mixed initiative use cases that enable semi-automated modeling tools to overcome the problems of error-proneness, complexity and inflexibility described above. Section 5 presents an evaluation of the presented existing approaches based on the mixed initiative use cases they support as well as additional criteria. Section 6 concludes the paper.

2 Semi-Automated Composition: A Definition

The purpose of this section is to introduce the term "semi-automated composition" and to give a working definition.

The terms "automated composition" and "semi-automated composition" are often used in recent research papers when the authors refer to the creation of composed services. However, there are different perceptions about the meaning of the term. It is probably due to the absence of a clear definition for this term that it is often used at random. A broad definition would be that every composition approach that requires user intervention at some point is a semi-automated approach. We are convinced that a more distinctive definition is necessary. We will give a definition in the remainder of this section.

Ponnekanti and Fox present SWORD [2], "a developer toolkit for Web service composition". In fact, SWORD is a framework providing automated planning for Web service composition. Still, the authors claim to present a semi-automated composition approach, which is probably due to the fact that the user can choose to create a persistent representation of a particular service composition (i.e., a plan) after it has been generated by the system.

In SWORD and similar frameworks, the underlying rule engine creates the service compositions independently from the user on the basis of the available formalized domain knowledge. The usage of the term "semi-automation" is therefore unsuitable when addressing approaches that involve manually reviewing automatically generated service compositions.

The definition of the term "semi-automated composition", which will be used throughout the remainder of this paper, is as follows:

> A framework or approach for creating composed services is called semi-automated, if it is committed to augmenting human planning skills

rather then to controlling the planning process. The user drives the composition process and has the possibility to cede control to system for specific (and limited) tasks.

An interaction between different entities, be it humans or intelligent systems, is called mixed initiative interaction, when all participants contribute at all points in time what is best suited to solve the overall problem. The term "mixed initiative" has been coined by the AI community and is often used in conjunction with semi-automated composition. Allen [3] introduces four different levels of mixed initiative. According to his classification, the semi-automated composition approaches presented in this paper reside on the lowest level of mixed initiative, called "unsolicited reporting".

3 Existing approaches

The purpose of this section is to give an overview of current research efforts regarding semi-automated service composition. Four approaches will be presented according to their main characteristics.

3.1 Web Service Composer

Sirin, Parsia and Hendler [4] present a prototypical implementation of a composer for Web services. Their tool allows creating executable compositions of Web services that are semantically specified with OWL-S [5].

The created service compositions can in turn be stored as OWL-S "process models". Process models are a part of OWL-S ontologies which is normally used to encode the choreography for a described service. Well-known control constructs from the area of Workflow Management can be used within OWL-S process models. It is therefore a suitable format for representing composed services.

The focus of their work is on filtering the list of available services at each composition step and thus helping the user to select the appropriate services.

In order to create a composed service, the user follows a backward chaining approach. He or she begins with selecting a Web service that has an output producing the desired end result of the composition from a list of all available services. Next, the user interface presents additional lists connected to each OWL input type of the service producing the end result. In contrast to the first composition step, these lists do not contain all available services: They contain only those services that generate an output compliant to the particular input type they are connected to. An output of a service A is compliant to an input of a service B, if their types are exactly the same or if the output of A subsumes the input of B (i.e., the input of B is a specialization of the output of A). If a service is selected from the list of compliant services, this service's inputs must again be produced by selecting services producing compliant outputs. This is repeated until the user decides at one point to provide the inputs that are not connected

to a compliant service by entering them as input values (or connecting them to compliant services that have no input parameters).

Creating the composed service by forward chaining (i.e., starting with the first activity in the process instead of the last one) is planned but not implemented in their prototype.

In addition to filtering on the compliance of the services in terms of their inputs and outputs, the user can apply further filtering based on the nonfunctional properties of the services. This only works for services that adhere to a specific OWL-S "service profile" (i.e., they implement the service profile). Once the user has selected a service profile, the system renders an UI element which allows him or her to provide values for the nonfunctional properties that are specified for the selected service profile. The user can then apply the filter, thereby further restricting the set of services that are presented for the current composition step.

Additional to its composing functionality, Web Service Composer can also execute the composed services: The services that can be selected must be specified in OWL-S and a grounding for WSDL must be provided. Therefore, the tool can invoke the services in the composition and pass the data between the services according to the user-specified control flow.

3.2 CAT

Kim, Spraragen and Gil introduce CAT (Composition Analysis Tool) [6], a tool which illustrates their approach to interactive workflow composition.

The focus of their work is to assist the user in the creation of computational workflows. The authors' work is not directly related to service composition. However, we can conceive a computational workflow as a service composition. The activities of the workflow are represented by services that realize data transformations.

The authors have developed their own knowledge base format, which they use to semantically describe the components that can be used in a workflow and their input and output parameters: "Component ontologies" describe hierarchies of components, from abstract-level components to executable components. An abstract component represents a common set of features that applies to all components of that type. "Domain ontologies" semantically specify the data types which can serve as inputs and outputs of the components described in the component ontologies.

In CAT, the user can add components to the composition at any time. There is no need for the user to follow a strict backward or forward chaining composition. The "end result" of the composition can be specified by declaring outputs produced by components as the end result (or as a part of it). Control flow in CAT is described by explicitly linking inputs and outputs of different services together. Values of input parameters can also be default values from the respective ontologies or values entered by the user.

Instead of filtering the set of services that can be included in a composition, CAT provides a list of suggestions about what to do next. These suggestions

resolve errors and warnings, which are also displayed. The idea is that consequently applying suggestions will produce a "well-formed" workflow as a result. The authors therefore introduce a set of properties that must be satisfied by the composition in order to be well-formed. These properties ensure that

- the composition has an end result,
- all components' inputs are satisfied
- all components have been specialized to executable components
- all components produce outputs relevant for producing the end result
- for all links between components there is a "subsumes"-relation between the output of one component and the input of the other component
- the composition does not contain redundant links or components.

Depending on whether these properties are satisfied or not, the ErrorScan algorithm (which is also provided in [6]) determines which suggestions are presented to the user. Possible suggestions are

- adding an end result
- removing a component
- inserting a link to or from another component
- entering the value for an input
- taking the default value for an input from the component ontology
- specializing a component
- removing a link
- adding a new component before a link

CAT uses heuristics to determine the ordering of the suggestions, so that more recent and more severe errors are displayed before warnings that do not necessarily have to be resolved in order that the workflow is well-formed. It is noteworthy that the suggestions in CAT have the property of being corrective or additive: Applying a suggestion never causes more errors than it resolves.

3.3 PASSAT

Myers et al. present PASSAT (Plan-Authoring System based on Sketches, Advice, and Templates) [7], an interactive tool for constructing plans. PASSAT is not directly concerned with the creation of composed services, but its concepts can be mapped into the context of service composition.

PASSAT is based on hierarchical task networks (HTN) [8], while the model has been extended to realize some concepts that are outlined below. In HTN planning, a task network is a set of tasks (or service calls) that have to be carried out as well as constraints on the ordering of these tasks. Moreover, it consists of a set of constraints that must be valid before the execution of the tasks and information about how the tasks instantiate variables. Because the variables (partly) describe the state of the world before and after the execution of a specific task, the constraints on these variables can be used to express preconditions and effects.

The HTN based approach naturally imposes top-down plan refinement as the planning strategy the user must adhere to: The user can start by adding tasks to a plan and refine them by applying matching HTN templates. A template consists of a set of subtasks that replace the task being refined, as well as the preconditions and effects of applying individual tasks and the entire template. It is noteworthy that the user has the possibility to override unmatched constraints when applying a template. This is especially desirable when comprehensive domain knowledge (i.e., a collection of templates) cannot be provided. Task refinement is repeated until the plan contains no activities that can be further expanded.

A core feature of PASSAT is its automated planning mode, which allows the user to have the system expand all remaining tasks, applying the templates that are currently available to the system.

PASSAT also features an "advice" mechanism that allows the user to specify high-level policies for the overall plan being created. These policies are global constraints that restrict the set of actions that the user can undertake when developing a plan. However, they can be relaxed and overridden and need not to be necessarily satisfied to reach the overall goal. The automated planning mode also takes these policies into account when it selects the templates for refining the open tasks.

Opposing the strict top-down refinement approach implied by the use of HTN networks, PASSAT provides a "plan sketch facility": This allows the user to freely arrange tasks that need not to be necessarily fully specified and that can reside on different layers of abstraction (regarding the template hierarchy). After the user has outlined a plan sketch, the system tries to find possible expansions by applying matching templates. The user can then choose one of these expansions to be included in the plan and return to the normal planning mode.

PASSAT also informs the user about open tasks and outstanding information requirements in order for the plan to be completed. Therefore, it presents the user with an agenda of actions such as "expand task", "instantiate variable" and "resolve constraint".

The system helps the user to choose from the applicable templates at a given composition step by keeping track of past user experience: A statistic about how often a template has been applied in plan refinement is encoded in the templates.

3.4 IRS-III

Hakimpour et al. introduce Internet Reasoning Service (IRS) [9], a Semantic Web Services framework. One of their implementations, IRS-III, includes a tool that supports a user-guided interactive composition approach by recommending component Web services according to the composition context.

Their approach uses Web Services Modeling Ontology (WSMO) [10] as the language to semantically describe the functionality of the Web services. In IRS-III, Web services are represented by WSMO "goals":

WSMO introduces the concept of goals to represent the objectives of users when consulting a Web service. A goal is a subset of a Web service's capabili-

ties that is of particular interest for the user, namely the service's outputs and effects. According to Hutter, this reflects the so-called goal driven approach in AI planning [11].

Similar to Web Service Composer [4], the user starts with adding the goal (i.e., a Web service) that produces the desired end result of the composition. The first goal can either be selected from a list containing all goals, or by searching for an appropriate goal. The inputs of this goal must then be fed by other goals or values entered by the user. Like in [4], the available goals at each composition step are filtered: Only the goals that produce outputs that deliver the desired input for the downstream goal can be selected.

The tool also features the execution of the composed services. During the execution, the orchestration engine queries the user to provide values for the inputs that have not been assigned goal or a value at design time.

In IRS-III it is possible to introduce WSMO goal-to-goal mediators into the composition. This is necessary, when two goals are to be connected that have been specified by different parties. In such cases it cannot be ensured, that the same ontologies and thus the same semantic descriptions for the inputs and outputs are used by the different parties. However, if two types in different ontologies describe the same concept, the user can specify a mapping between them in a mediator.

The tool also allows if-then-else control operators to be added to the service composition.

4 Mixed initiative Use Cases

In this section, we will describe three use cases characteristic for semi-automated service composition approaches that enable them to overcome the problems of complexity, inflexibility and error-proneness akin to conventional service compositions, as described in section 1. We extracted these use cases from the approaches presented in section 3. We generalized and extended them according to our own research findings.

Figure 1 shows the use cases that are going to be developed throughout the remainder of this section. Before investigating these use cases in detail, we will refresh our understanding of how to semantically describe the capabilities of a service.

4.1 Prerequisites

Services can be *information-providing*, *world-altering* or both. The execution of information-providing services results in a change of the information space at a given point in time. In OWL-S ontologies [5], the *inputs* and *outputs* of a service describe the data transformation that is accomplished by a service. If a service has world-altering capabilities, the preconditions and effects describe a part of the state of the world before and after its execution. Abbreviated,

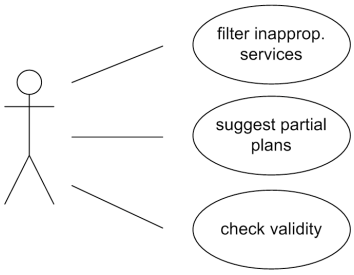


Fig. 1. Mixed initiative Use Cases

this information is called the IOPEs of a service. We will use this terminology throughout the remainder of this paper.

While service compositions are usually seen as a set of activities among which an ordering relation exists, we can also conceive them as a set of states and transitions. In doing so, the transitions denote the individual activities (i.e., the services or service calls).

The execution of a service results in a change of the state. The information space and the state of the world in a given state in a service composition depend on the IOPEs of the services that precede this state. A transition (i.e., a service call) from a state A to another state B is allowed only if the inputs and preconditions of the service describing the transition are satisfied in state A. The other way round, this transition implies that the outputs and effects are valid in state B. The outputs and effects that become valid at this point are added to the outputs and effects of the preceding states. These are all the outputs and effects that have either been produced by upstream service executions or that have been provided by the "environment" (e.g., data which is available to all services in a composition per definition). This is also depicted in figure 2. The circles represent states, the rectangles represent transitions (i.e., service calls) and the dotted lines represent the outputs and effects that make up the information and world state. Please note that an effect has the power to negate other effects that may hold prior to the execution of the service that produces the effect.

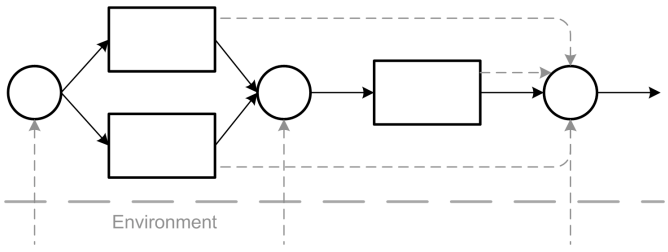


Fig. 2. A state depends on the outputs and effects of all precedent services

A large number of service compositions can be modeled without preconditions and effects. Every composition which has the goal to realize a data transformation can be described only using inputs and outputs. However, if we want to take preconditions and effects into account when creating composed services, we have to ensure that they can be evaluated later at runtime.

4.2 Filter Inappropriate Services

A major problem with creating composed services is that the number of activities that can be selected might be extremely high, depending on the domain. For instance, SAP's Enterprise Service Repository today contains more than 500 services, and growing [12]. As users of a tool for creating service compositions cannot oversee such a vast amount of available options, it is desirable to filter the set of available services. Such filtering can be done based on semantic service descriptions.

When creating composed services, users select services and add them to the composition. In a given state, it is possible to filter the selection according to semantic descriptions: It is desirable that services requiring inputs and preconditions that are not satisfied in a given state will be filtered, effectively reducing the number of choices presented to the user.

While filtering based on the services' IOPEs restricts the set of presented services to those which are compatible with the current state, the set can be further restricted by filtering based on the nonfunctional properties of the services.

Nonfunctional properties do not only offer a possibility to record juridical relevant information like a publisher's name and address, but also quality indicators for services. Such indicators can be measures that address the performance (in terms of response time), error rate or robustness of a service, as well as issues like scalability, reliability, geographical coverage, invocation cost and many more.

When creating service compositions, the user may find himself in a situation where more than one available service offers the functionality that is needed to go to the next state. At this point, the editor should allow the user to assign values to the nonfunctional properties of the presented services. These values are then evaluated by the matchmaker and only those services that both provide the desired functionality and comply with the user-specified nonfunctional properties are presented for selection.

A technical issue that has to be resolved is the fact that it is unlikely that all the semantic service specifications for the services providing equivalent functionality contain the same set of nonfunctional properties. Possible strategies depend on the language concepts of the used semantic specification framework (e.g., OWL-S [5], WSMO [10] or WSDL-S [13]), and on whether or not the editor incorporates the concept of abstract services.

An abstract service represents a set of service capabilities (i.e., the functionality a service provides). Abstract services in composed services can be used to realize a late binding of the concrete services at runtime: The engine that executes the process can discover all currently registered services that implement the functionality specified in the abstract service.

As specifications for abstract services are generalizations of concrete services, it would make sense to annotate them with a set of nonfunctional properties that is common to all concrete services. That way, the editor could offer a selection of abstract services for which the values of their nonfunctional properties can then be assigned by the user.

The use case of showing the applicable services for moving to the next state can be further enhanced by ordering the list of possible services according to the "degree of match" that the matchmaker returns for a service: The services that require exactly the inputs and preconditions that hold in the given state (i.e., an "exact match") should be presented first. Further ordering of the list can be based on the minimal distance between the respective concepts in the taxonomy. This distance can be translated into a classification of the "goodness" of the match, according to Li and Horrocks [14].

Another (and probably more accurate) possibility to order the list of applicable services would be to consult ratings of how often the user has selected the particular services. As this does not directly involve service semantics, this will not be further detailed here. However, it would be conceivable to incorporate a rating facility in the semantic service descriptions. The editor could update the descriptions in order to maintain a nonfunctional property such as "user rating".

4.3 Suggest Partial Plans

Automated planners will always plan according to an algorithmic planning strategy, such as for example forward- or backward chaining of services, or both. Human planners will in contrast not always behave according to this schema when they model composed services. Users might have a clear idea about some specific activities that they want to have in the process, without a global understanding how the whole will fit together as a process.

A possible user behavior is to start modeling the composed service by adding some activities and chaining them together, and then continue with an activity with unsatisfied inputs and preconditions representing some state later in the composition. In such and similar cases, it might be desirable for the user to let the editor generate valid service chains that connect two unrelated activities. This is depicted in figure 3.

Connecting two unrelated activities in a composition constitutes a standard planning problem, given the state after the execution of activity A (the initial state), the state before the execution of activity B (the goal state), the set of activities, as well as the set of all possible states and the set of all possible state transitions. The two latter sets can be derived from the semantic specifications of the activities (i.e, the services). The problem is therefore handed over to an automated planning engine.

A special case of the "Suggest Partial Plans" use case is when activity B (i.e., the activity to connect to) is the last activity in the composition. When this function is executed, the editor finishes the service composition.

If there are multiple possibilities to get from activity A to activity B, the editor should present the alternatives. In advanced scenarios it might also be

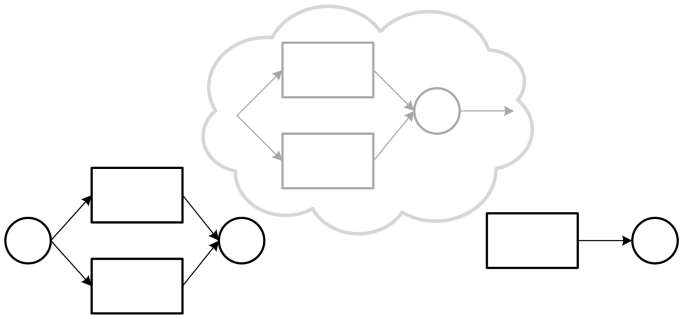


Fig. 3. A partial plan capable of connecting two unrelated services

useful if the editor presented a rating of the alternatives based on the aggregation of a nonfunctional property such as cost, for example.

In any case, the generated service chain should be editable by the user.

4.4 Check Validity

Because the human planner has full control over the modeling of the business process in a semi-automated environment, it is natural that errors are likely to be introduced into the composition. It is therefore necessary to provide the possibility to check the overall process validity.

Like the use cases that have been defined earlier in this section, the validity check can be realized using Semantic Web technology. A simple validity check would be to verify that the inputs and preconditions are satisfied for each activity in the service composition.

Such validity checks could be executed by the user at the end of the modeling process. However, in order to better support the user in creating composed services, validation should be interleaved with the actual modeling of the composed service: The user should be informed about unresolved issues in an unobtrusive way.

Unresolved issues arise from activities in the composition which violate one ore more aspect of a set of desirable properties for well-formed workflows, which has been introduced by Kim, Spraragen and Gil [6]. A workflow is well-formed, if

- one or more activities are contributing to the composition's "end result",
- the inputs and preconditions of every activity are satisfied,
- every activity is either an "end result" or produces at least one output or effect that is required by another activity,
- it does not contain redundant activities.

These properties should be checked after each user action so that the list of unresolved issues can be updated. To further assist the user, the editor could even suggest appropriate actions to resolve the open issues. Such actions include

adding or removing activities to or from the composition as well as adding or removing control flow between two activities.

A problem that arises when suggesting actions is that the suggested action can introduce new errors to the composition. In doing so, a suggested action could produce more errors than it resolves.

5 Evaluation

In the following, we will present an evaluation of the semi-automated composition approaches presented in section 3 based on the supported mixed initiative use cases as well as additional criteria.

5.1 Supported Mixed initiative Use Cases

Table 1 gives an overview of the mixed initiative use cases that are supported by the semi-automated composition approaches presented in section 3.

	Web Service Composer	**CAT**	**PASSAT**	**IRS-III**
Filter Inappropriate Services	Considers only inputs and outputs	N/A	N/A	Considers only inputs and outputs
Extensions	Filtering on NFPs, Ordering by match goodness	N/A	N/A	N/A
Suggest Partial Plans	N/A	N/A	HTN template expansion	N/A
Extensions	N/A	N/A	High-level policies for composition	N/A
Check Validity	N/A	Evaluates "well-formedness"	Tracks open information requirements	N/A
Extensions	N/A	Suggests fixes (ErrorScan)	Prioritized agenda	N/A

Table 1. Supported mixed initiative use cases

Web Service Composer filters the list of services that can be included in the composition at each composition step. This realizes the use case "Filter Inappropriate Services" that was presented in section 4 of this paper. However, the realization of this use case in Web Service Composer is restricted in two ways:

First, the tool only considers inputs and outputs, i.e., the mere data transformation that services realize. The preconditions that must be satisfied before the execution of the services and the effects that the executions of the services have on the state of the world are not taken into account.

Second, the selection of appropriate services is done per input of a downstream service that must be satisfied, which is due to the strict backward chaining approach imposed by the tool. This means in consequence that the plans constructed with the tool are not always optimal. For example, when one service operation delivers two outputs each of which satisfies a different input of a downstream service, this services operation has to occur twice in the composed service.

Web Service Composer supports two extensions of the use case "Filter Inappropriate Services" that have also been identified in section 4: First, the tool can further restrict the set of filtered services according to user-specified values of nonfunctional properties that are common to that set. Second, the list of filtered services which is presented to the user is ordered according to the goodness of match: Services that exactly produce a necessary input for a downstream service (i.e., an exact match) are ranked higher than services that produce outputs that subsume the necessary inputs.

IRS-III also supports the use case "Filter Inappropriate Services". While IRS-III underlies the same restrictions for that use case, none of the extensions specified in section 4 is realized.

PASSAT is the only tool of those included in this survey that partially supports the use case "Suggest Partial Plans". PASSAT is a tool for interactive plan authoring based on HTN networks. The user can invoke an automated planning mode to expand open tasks in the plan. This can be seen as a specialization of the use case "Suggest Partial Plans" in the sense that partial plans can only be generated from the current state to a state in which the composition is finished, i.e. all tasks can be executed. However, this realization of the use case is restricted in the way that the user must have completed the plan on a high level of modeling - otherwise the task network cannot be expanded.

In section 4 we have described a possible extension of the "Suggest Partial Plans" use case: If there is more than one alternative for a partial plan, a ranking of user-specified nonfunctional properties should determine the order in which the alternatives are presented to the user. In PASSAT, the user can specify high-level policies (e.g., "maintain an overall cost total of less than $ 100") which are also taken into account when automated template expansion is performed. This can be seen as a realization of that extension, as the alternative for a template expansion that conforms best to the specified policies will be presented to the user.

PASSAT also supports the use case "Check Validity", as it interleaves a checking mechanism with the actual planning process: After each user action the system updates an agenda showing open information requirements that must be satisfied in order to have an executable plan. As an extension to this mechanism, PASSAT orders the agenda according to user-specified criteria.

Another, more thorough realization of the use case "check validity" can be found in CAT. Here, the tool checks at each composition step if the composition complies with a set of properties that describe the "well-formedness" of the composition. In case these properties are violated, the system consequently presents a list of warnings and errors. As an extension of this use case, the authors present an algorithm that presents the user with suitable suggestions for next steps based on the evaluation of the well-formedness criteria. The applicability of CAT has been shown in the domain of seismic hazard analysis; however, it remains unclear why the authors opted for developing their own correctness criteria for computational workflows rather than building upon more established approaches to verify workflow correctness, such as the soundness criteria introduced by van der Aalst [15]. Also, the authors do not describe how their notion of well-formedness relates to the soundness criteria for workflows.

5.2 Additional Evaluation Criteria

The mixed initiative use cases of a semi-automated composition approach are its most important characteristic. However, there are more criteria that allow further distinction among such approaches. In the remainder of this section, these criteria will be presented and applied to the semi-automated composition approaches presented in section 3. Table 2 gives an overview of how we evaluate the presented approaches according to these criteria.

	Web Service Composer	**CAT**	**PASSAT**	**IRS-III**
Imposed planning strategy	Backward chaining	None	Top-down refinement	Backward chaining
Modeling environment	Graphical	Textual	Textual	Graphical
Knowledge base	OWL-S	Non-standard	Non-standard	WSMO
Reasoning	Output-input subsumption	Output-input subsumption	Takes complete IOPEs into account	Output-input subsumption
Control constructs	Not provided	Not provided	Not provided	if-then-else construct
Compositions are executable	Tool acts as Web service client	No	No	Orchestration engine
Output format	OWL-S process model	Non-standard	Non-standard	Non-standard

Table 2. Evaluation according to additional criteria

An important criterion for the user who created composed services is the way in which composed services can be modeled with the system he or she utilizes.

As human planners are likely to feel constrained when they are forced to adhere to an algorithmic planning strategy, the tools should give the users maximum freedom in modeling their compositions.

Web Service Composer and IRS-III impose a strict backward chaining planning strategy to the user. The user has to start with the last activity in the composition, i.e., the activity producing the desired end result. The inputs of this activity are then recursively satisfied until the first activity in the composition (e.g., a user input) is reached. Due to the strict backward chaining approach only the last activities of compositions created with Web Service Composer and IRS-II can determine the end results, which is also problematic.

Another criterion that is highly important to the user of semi-automated composition tools is whether a graphical user interface is provided. Web Service Composer and IRS-III provide the user with a graphical user interface, while CAT and PASSAT come with mere textual modeling environments. Especially for complex compositions, the user can hardly oversee the causal relations between the activities.

Semi-automated service composition approaches reason over domain knowledge that is specified in ontologies. In order to support the maintainability of the composed services that are created using semi-automated composition tools, standardized formats should be used for the ontologies. Because the formal specification of domain knowledge presents a tremendous challenge, organizations have to rely on available ontologies that have been created by other parties as building blocks for assembling their domain knowledge. Web Service Composer and IRS-III build upon open formats such as OWL-S and WSMO. Additionally, IRS-III allows the use of WSMO mediators in the compositions, which eases the process of integrating ontologies from different parties. CAT and PASSAT in contrast are building upon proprietary formats for encoding domain knowledge.

When service capabilities and functionalities are specified in ontologies, the data transformation realized by a service's execution can be specified as well as the change in the state of the world that the execution of a service implies. Three semi-automated service composition tools out of the four presented in this survey only reason on the inputs and outputs of the services that can be included in the compositions (i.e., the data transformation that the services effect). However, a large number of possible applications (i.e., the set of computational workflows) can be described only using inputs and outputs. PASSAT is the only approach among those presented here that explicitly supports constraints on the state of the world.

When modeling composed services, we naturally expect the possibility to model control flow between the individual activities. Here again, three tools out of the four presented do not provide control constructs. This is probably related to the fact that most tools reason only on the inputs and outputs of the services that can be included in a composition: If preconditions and effects are not considered, the control flow of a composition derives implicitly from the data flow. Being the only approach of those investigated in this paper that supports

preconditions and effects, PASSAT lacks a notion of explicit control flow. IRS-III in contrast provides a basic if-then-else operator.

A semi-automated service composition tool should somehow ensure that the composed services being created with it can be executed. This can be done either by directly proving the user with an execution environment or by exporting the compositions into an executable format.

Web Service Composer allows to directly execute composed services by calling the individual services via the WSDL interface that is provided in the groundings of the OWL-S ontologies used. As the tool acts as the Web service client for all calls, it does not support complex choreographies. IRS-III in contrast comes with an orchestration engine on which the composed services can be enacted, allowing the user to specify choreographies that include more than two parties.

In addition to its Web service execution functionality, Web Service Composer is able to store the composed services as OWL-S process models. Together with an OWL-S grounding, the service compositions are executable on platforms other than Web Service Composer. CAT, PASSAT and IRS-III are unable to store composed services in an open format. The de-facto standard format for executable service compositions is WS-BPEL [1]. It is striking that none of the tools offers WS-BPEL export functionality.

6 Conclusion

In this paper, we have developed a definition of the term "semi-automated composition", filling the gap that a comprehensive definition is not available at present.

We have also given an overview of recent research approaches towards semi-automated composition by presenting four different semi-automated service composition tools.

As the main contribution of this work, we have developed three mixed initiative use cases for semi-automated composition, which we have extracted and generalized from the presented approaches and then extended.

Based on these use cases, we have given a qualitative evaluation of the approaches presented. Additionally, we have developed a number of distinctive properties that are characteristic for semi-automated approaches and applied them to the existing ones.

Our results show that no semi-automated modeling tool known to us is complete in terms of the functionality it provides: None of the presented existing approaches covers all the mixed initiative use cases that have been developed in section 4 of this paper.

While the use cases "Filter Inappropriate Services" and "Check Validity" are both realized in two out of the semi-automated composition approaches, the use case "Suggest Partial Plans" is only addressed in one of them: PASSAT [7] partly implements this use case with its feature of automated plan expansion. Plans in PASSAT are represented as Hierarchical Task Networks (HTN). The user can invoke an automated planning mode that expands any open tasks within a plan.

The presented approaches could hardly be extended in order to provide the missing functionality in terms of the presented mixed initiative use cases. Three out of the four tools presented are only capable of reasoning over input and output types, which only covers the class of information-providing services. In contrast, the described mixed initiative use cases also consider the class of world-altering services. Another problem with extending the functionality of the existing approaches is that all of them use different ontology formats to represent domain knowledge, making it difficult to integrate them.

The ontology format underlying a semi-automated composition approach is also important when we consider the fact that the task of formally specifying domain knowledge is very delicate. It will be necessary for organizations utilizing this kind of technology to use ontologies that have been created by other parties. It is therefore necessary to build upon standards. Two of the four tools presented implement a proprietary ontology format, which hinders the exchangeability of domain knowledge between organizations.

Another problem common to all presented existing approaches is their weak usability. Two of four approaches provide mere textual interface. The tools with graphical user interfaces are prototypical proof-of-concept implementations. None of the four presented approaches provides a user interface that would enable domain experts to create composed services without training. Thus, the industrial applicability of semi-automated service composition technology is limited today.

References

1. Arkin, A., et al.: Web Services Business Process Execution Language Version 2.0. Technical report (2005) http://www.oasis-open.org/apps/org/workgroup/wsbpel/.
2. Ponnekanti, S.R., Fox, A.: SWORD: A Developer Toolkit for Web Service Composition. In: Proceedings of the Eleventh International World Wide Web Conference (WWW02), 7-11 May, 2002, Hawaii, USA, ACM (2002)
3. Allen, J.: Mixed Initiative Interaction. IEEE Intelligent Systems **6** (1999) 14–16
4. Sirin, E., Parsia, B., Hendler, J.: Filtering and Selecting Semantic Web Services with Interactive Composition Techniques. IEEE Intelligent Systems **19** (2004) 42–49
5. Martin, D., et al.: OWL-S: Semantic Markup for Web Services. Technical report (2003) http://www.daml.org/services/.
6. Kim, J., Spraragen, M., Gil, Y.: An intelligent assistant for interactive workflow composition. In: IUI '04: Proceedings of the 9th international conference on Intelligent user interface, New York, NY, USA, ACM Press (2004) 125–131
7. Myers, K.L., et al.: PASSAT: A User-centric Planning Framework. In: Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space, Houston, TX, USA, AAAI (2002)
8. Tate, A.: Generating Project Networks. In: Proceedings of the Fifth Joint Conference on Artificial Intelligence, Cambridge, MA, USA, Morgan Kaufmann Publishers (1977) 888–893

9. Hakimpour, F., Sell, D., Cabral, L., Domingue, J., Motta, E.: Semantic Web Service Composition in IRS-III: The Structured Approach. In: CEC, IEEE Computer Society (2005) 484–487
10. Roman, D., Lausen, H., Keller, U.: D2v1.2. Web Service Modeling Ontology (WSMO). Technical report (2005) http://www.wsmo.org/TR/d2/v1.2/20050413/.
11. Hutter, M.: Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability. Springer, Berlin, Germany (2004)
12. Richardson, B.: The View from SAPPHIRE: No Desktop Left Behind (2005) http://www2.cio.com/analyst/report3603.html.
13. Akkiraju, R., et al.: Web Service Semantics - WSDL-S. Technical report (2005) http://lsdis.cs.uga.edu/projects/METEOR-S/WSDL-S.
14. Li, L., Horrocks, I.: A software framework for matchmaking based on semantic web technology. In: Proceedings of the Twelfth International World Wide Web Conference (WWW2003), 20-24 May, 2003, Budapest, Hungary, ACM (2003)
15. van der Aalst, W.M.P.: Verification of workflow nets. In: ICATPN '97: Proceedings of the 18th International Conference on Application and Theory of Petri Nets, London, UK, Springer-Verlag (1997) 407–426

Automated Workflow Planning in Agent-Based Semantic grids

Volker Gersabeck

Hasso-Plattner-Institute
for Software Systems Engineering
University of Potsdam
volker.gersabeck@hpi.uni-potsdam.de

Abstract. In some workflow based grid applications the failure of the execution engine causes the failure of the whole application. This paper presents two approaches of how to distribute the execution and thus the planning of the workflow. Using an example grid application, a peer to peer architecture and a clustered architecture based on agent technology are explained and discussed showing their advantages and disadvantages.

Introduction

Sciences like High Energy Physics, Earthquake engineering, or Global Climate Change have a certain need for a huge computational infrastructure. This need can be covered by a distributed platform for large-scale computations, data and remote resource management, also called a grid [10].

A main challenge in grid systems is the composition of applications exploiting all the power of the resources in order to provide the results as fast as possible. As applications need several hours up to some days [2], time is an essential factor in the development of the applications. Usually one application is executed only a few times, which leads to the usage of workflow execution engines with a smaller development time for the application itself. The application can be described by a workflow and executed by the engine. The workflow then consists of the application logic plus the grounding on the resources to be used during execution.

The act of modelling the application logic and doing the grounding is also called planning of the workflow. Such a workflow consists of a final piece of data that is to be produced by the application using several tasks arranged in a partial order. If we call the final piece of data the goal of the application and each task a planning operator we have the requirements for a planning problem that can be solved using AI planning techniques. The inputs and outputs of each operator need to be specified not only syntactically but also semantically in order to automate the planning of the workflow. As many scientists are specialists in their field of investigation but not necessarily in programming, automated workflow planning helps them to create their applications in less time with better results.

The automated workflow planner needs semantic descriptions of each operation and each resource. Thus, the whole grid system is described semantically and we are

now speaking about the Semantic grid [5], analogous to the Semantic Web [1]. Another trend in the grid community is to call each operator or each task of the logical workflow a service, while the notion of a resource describes a piece of data or a physical resource needed to execute a service. In this paper I will follow this trend and use the words service and resource as described.

In a highly distributed environment like a grid it is quite usual that some nodes fail to provide a job, especially with the extremely long execution times of grid applications (up to some days). Thus, the workflow execution engine needs to deal with these failures and from time to time has to do a re-planning, i.e. exchange the failed resource by another one capable of doing the required job in case of failure. As already mentioned, the execution of an application takes a lot of time; it is now obvious to do the grounding of some parts of the workflow closer in time to its execution. This procedure, also called just-in-time planning [6], prevents failures and also some of the re-planning.

By using re-planning the grid system tolerates failures of some resources. But if the node running the workflow execution engine fails, the whole application might fail. To prevent this, a distributed engine is needed. If one part of the engine fails, the other parts are able to replace it and rescue the execution of the application.

Dealing with failures and the fact of the grid being an open distributed system show the need for autonomous flexible behaviour of the workflow execution engine. A software agent is capable of flexible, autonomous action in some environment in order to meet its design objectives [16]. Thus, it is obvious that integrating agent technology in grid systems helps solving some upcoming problems. Foster et. al. [9] also suggests bringing both technologies together in order to let them profit from each other.

After presenting some related projects two architectures of distributed workflow execution engines will be explained: a peer to peer architecture and a clustered architecture. Using an example grid application from high-energy physics, automated workflow planning and execution will be shown in both architectures. By counting the necessary messages the approaches are compared and discussed. Finally a conclusion shows which approach promises better results.

Related Work

There are some projects that deal with the combination of multi-agent systems and grid systems. Patel et. al. [13] for example presents architecture for an Agent-Based Virtual Organization realizing the management of a grid system. The authors represent each node of the grid as a “Service Provider” offering its services to the organization. One “Virtual Organization Manager” is central to each virtual organization. It deals with discovery, contract negotiation, trust, and quality of service issues. Each part, i.e. each service provider, the virtual organization manager, the discovery service, etc. is represented by one agent respectively. The service requester is represented by one service provider.

The presented project concentrates more on issues like quality of service and trust than on planning and failure handling. Thus, there are not many hints on how to distribute planning and how to handle failures. But they show how agent technology can be exploited within a grid infrastructure.

Although there are differences between a grid and Semantic Web Services, it is possible to adapt certain features of agent-based Web service infrastructure to an agent-based grid. Buhler et. al. [4] presents a service composition engine consisting of many Service Agents each representing a certain service. Each service agent knows its possible successors, i.e. those service agents having as input at least a part of the output of this service agent. In order to compose a service, the engine starts the search by sending composition request messages to some agents, who then forward this composition message to their successors and so on until the requested result can be composed of the output of some service agents. The found composition of services can then be executed to provide the result to the requester.

The authors mentioned that their system scales adequately although each service agent represents only one service. Because the number of services to be invoked in a grid is very high (over a thousand jobs [6]), it is doubtful whether this approach can be adapted as is to a grid. This will be discussed later on.

Another approach, which is based on peer-to-peer and agent technology, is shown in [17]. Each service provider and the service requestor are represented by one agent. The service provider agents can exchange intermediate data and control information between them making a central coordinator unnecessary during the enactment of the workflow. The service requestor knows the workflow at request time, which means that it does not create the workflow within the presented framework. It only has to find, select and contract service providers for each task of the workflow. For each task it creates one negotiation agent that negotiates with possible service providers for this task. A coordinator agent assures the overall quality of service by coordinating all negotiation agents.

As the workflow planning is not automated there are no hints on how to do this in a decentralized manner. But the part of service selection and especially the part of workflow enactment might be adaptable to a grid. The fact that each service provider is able to send intermediate data immediately to the service provider who needs this data reduces the management overhead and the network traffic. The authors also presented a mechanism for automated exception handling and re-planning within the involved agents. This mechanism seems to be adaptable, too.

The authors of [12] have developed the most complete workflow management system for a grid using automated planning techniques. In their current paper they present some future aspects on workflow management in grid. They reason about a decentralized system based on Agent technology. Although a concrete architecture of such a system is not shown I will refer to their work as I based my architecture on some of their ideas.

Architecture of an Agent-Based Grid

The previous chapters showed that there is a need for a decentralized workflow execution engine. This chapter discusses different possible architectures that realize the distribution taking into consideration the requirements that grid systems dictate.

Cluster vs. Peer-to-Peer

There are two possible architectural patterns for decentralized systems: a peer-to-peer architecture and a cluster architecture. In contrast to a client-server architecture, a pure peer-to-peer architecture has no central instance like a server. The parts of the peer-to-peer system have equal capabilities of executing application logic. Thus, the software running on each peer should realize the same application logic.

A computer cluster is a group of loosely coupled computers that work together closely so that it might be viewed as though it were a single computer. The grid system would then consist of several clusters capable of executing a workflow using its cluster nodes. The clusters can then be organized in peer-to-peer manner. The organization of the cluster nodes is transparent to the other clusters. Thus, different approaches are possible: a client-server architecture as shown in the diagram below or a peer-to-peer architecture. In the latter case it is far more difficult to give the impression of being a single computer. A server has full control of each cluster node and is a single point of entrance to the cluster. That way it is possible to act as a unit to the other clusters and distribute the workload to the cluster nodes.

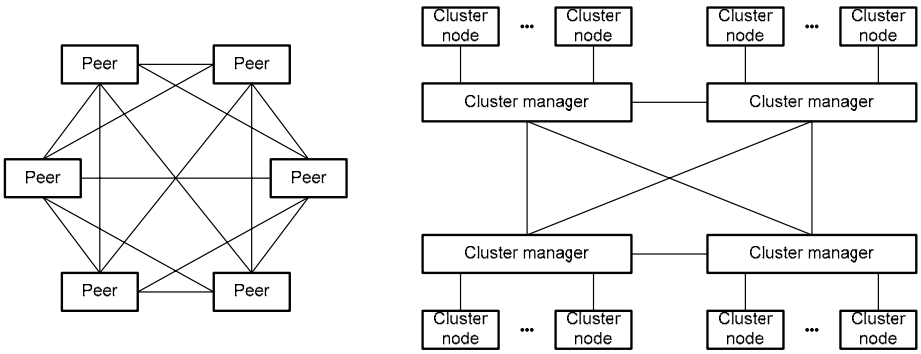


Fig. 1. Peer-to-Peer (left) and Cluster (right) architectures

Applying a peer-to-peer architecture to a grid comes with some obstacles. As there is no central instance for doing the grounding of the workflow, the peers have to negotiate about it. As each negotiation needs time and the number of jobs to be grounded is high, over a thousand or even a million jobs, the time for negotiation cannot be ignored. Another fact is the maintenance of the network. Each peer needs to check whether its neighbours are still alive, which results in additional network traffic. Although the diagram above shows a possible immediate communication between each peer, in a network with over thousand peers, this is almost impossible. Thus,

each network package needs to pass several peers until it reaches its destination. But as network bandwidth is valuable for grid applications, a communication overhead created by the infrastructure is not desired.

The second approach of distributing the workflow engine is clustering the grid as shown in figure 1. In here the cluster managers have to decide on who is responsible for which part of the workflow. As there are not that many cluster managers the negotiation and the communication efforts are still in acceptable range. Then, each cluster manager needs to do the final grounding of the jobs in its workflow parts. Because a central instance managing the job execution exists in a cluster, a late grounding of the jobs is possible without a time consuming negotiation.

Although at first sight the cluster architecture seems to be the better solution, there are two crucial points: the partitioning of the workflow and the clustering of the grid. As the first point is related to the planning of the workflow and depends also on the clustering of the grid, it will be discussed in the next chapter. Following, some approaches of clustering a grid are shown.

Different approaches of clustering

Maybe the most obvious approach of clustering the grid is to group the nodes that belong to the same LAN, because the communication within a LAN should be much faster than communication over the internet. As there will be a lot more communication between the nodes of one cluster than between two clusters, this might make the communication faster. A disadvantage is the combination of nodes by chance. There is no guarantee that there will not be more communication between clusters, as different jobs might need different resources of which some are not available within the current cluster. Thus, some jobs need to be executed by one cluster and others by a second. At least control information needs to be exchanged between the two clusters in order to enact the different jobs.

A second approach is clustering by resource types. This means to group together the nodes with similar capabilities of computational power and amount of storage. This way, the cluster manager has a huge range of possible resources to do the grounding, in order to maximize the parallelism of the application. The example application presented in the next chapter shows why this is a major advantage. Because the workflow execution should be tolerant to failures, it is recommended to have more than one cluster per resource type, such that the failure of an entire cluster can be recovered by another cluster of the same resource type.

Both approaches have their advantages. In order to show how it is possible to maximize the parallelism, clustering by resource type will be used in the next chapter, which deals with planning and execution of a sample grid application.

Planning in an Agent-Based grid

Before presenting different aspects of workflow planning in grid systems, an example application is shown to give the reader an impression of how a typical application for grid systems looks like.

An example application: particle decay simulation

The particle decay simulation is needed to verify a theory of particle decay against measurement results that were collected by a detector. In the shown case the theory depends on one argument *a*. By executing the simulation with different values for this argument the best value for the argument should be obtained. The workflow below shows the execution of the simulation for 100 different values of the parameter *a*.

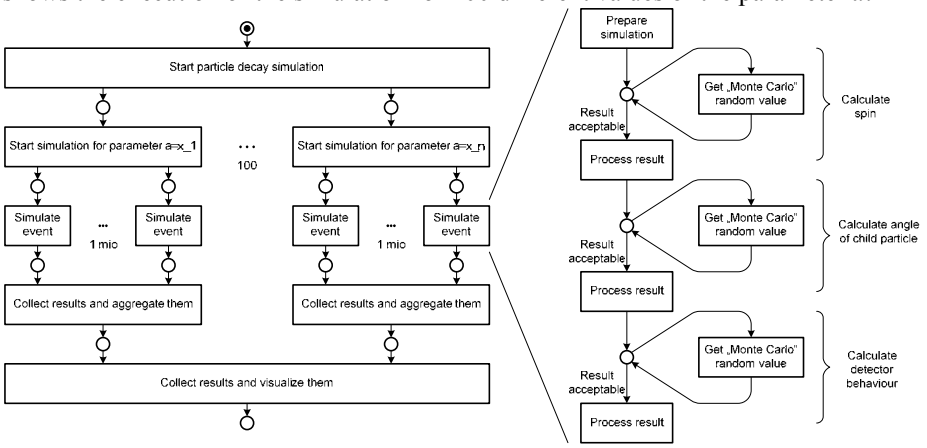


Fig. 2. Workflow of particle decay simulation (left) and simulation of one event (right)

Each simulation then consists of a certain number of events to be simulated. In this case one event is the act of decay of one particle in two pieces. Both pieces will go away from each other in an angle of 180°. Thus, it is sufficient to calculate the angle of one piece by the simulation. As this angle depends on the spin of the original particle, the spin also needs to be considered in the simulation. In the third step of the simulation of one event, the behaviour of the detector calculated. In all three steps a "Monte Carlo Technique" [7] is used to sample random variables. As this technique is not deterministic in its runtime behaviour, i.e. the number of loop iterations is variable, and therefore the overall runtime of the event simulation cannot be calculated beforehand.

The number of events in one simulation is likely to be around 1 million. Thus, the number of generated random values in the whole application is at least 300 million. Each simulation produces several GBytes of data that need to be stored temporarily and aggregated for evaluation later on. The overall execution time of such an application for state-of-the-art high energy physics experiments would be around one day, when 100 machines with about 3 GHz processors and 2-4 GByte of main memory were used.

These figures show the difference between an application consisting of Semantic Web Services and one for a grid system. In a grid system execution times of several hours up to some days are usual while an application of Semantic Web Services has execution times that are in the range of seconds. Thus, the planning of the workflow for grid applications could take some time when this leads to significant shorter execution times.

Planning in Peer to Peer architecture

An abstract workflow shows the application logic. An example of an abstract workflow is shown in figure 2. In contrast, a concrete workflow also contains tasks like data transport from one location to another or the task of storing a piece of data at a certain location. In a distributed workflow engine the task of enacting the next cluster or peer might appear in the concrete workflow, too. As in a peer-to-peer architecture the services are bound to a resource, planning the abstract and the concrete workflow needs to be done in one step.

In the peer-to-peer architecture planning of the workflow needs to be done in a distributed manner, as there is no central node that could do the planning. In addition, there is no central registry of the peers that would be necessary in order to know the provided services and resources. A distributed planning approach for Web services is shown in [4] (see also section related work). Adopting this approach to the grid system results in the following message sequence.

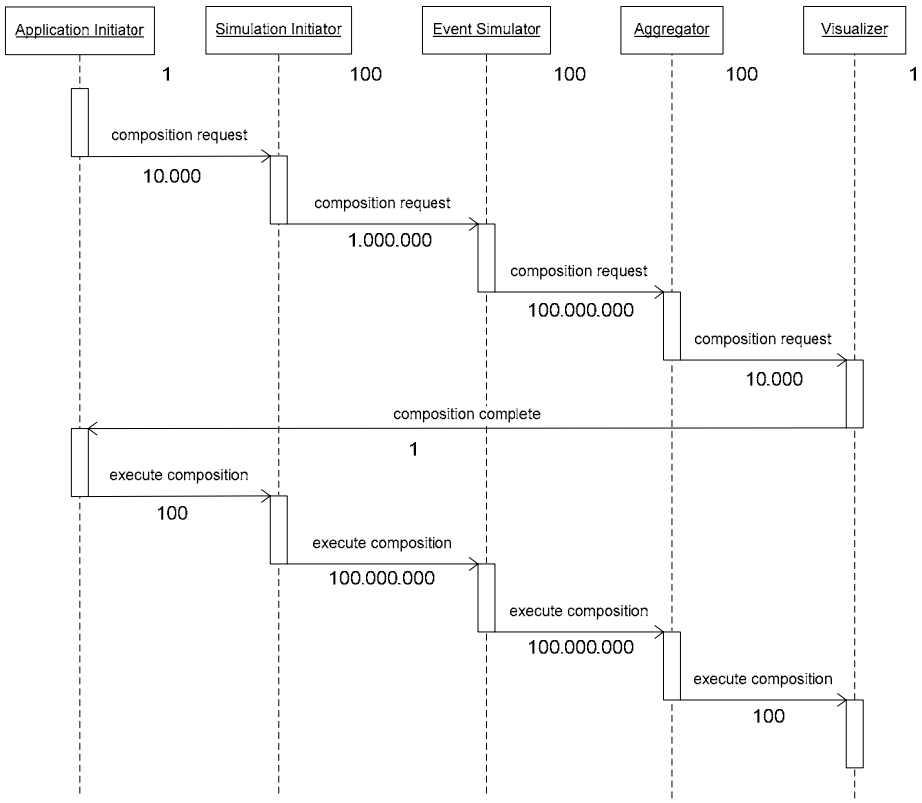


Fig. 3. Message Sequence in Peer to Peer architecture for example application

The numbers in the diagram say how often a service exists within the grid and how often a message will be send. The system consists of a hundred machines each hosting the same implementation of the services. In order to start the execution of the sample

application the application initiator needs to send requests for each of the hundred simulations to each of the hundred service providers. The simulation initiator then sends one request to each event simulator. This is already an optimization, as only one request is sent for the million executions of the event simulator. Then each event simulator sends one request to each aggregator, who has to choose one path out of all incoming requests. Thus, each aggregator chooses a hundred paths (one for each simulation) and send a request to the visualizer. The visualizer then has to choose a hundred paths out of the incoming requests, one for each simulation, and returns a control message indicating completeness of the workflow. Then the execution can be started by sending one message for each task to be started.

In this sample application there are many optimizations possible. In order to keep the algorithm general and not optimized for this application, no further optimizations were included.

The sequence shows that there are quite a lot of messages (around 300 mio) to be sent in order to execute the example application. And there is not yet any optional implementation of the services. In addition, if one service fails the whole application will fail because no monitoring is included. One possibility would be to send a message back to the initiating service when the service has finished. This way, the initiating service could choose another service, if its successor service fails. But this would double the number of messages during execution. This monitoring cannot be done by the successor service, because this service does not know when its predecessor started with execution and thus also does not know when it should be finished.

Another difficulty with monitoring and re-planning is the fact that each service must know an alternative service it can call, when the desired service fails. But storing each alternative in the workflow would drastically inflate its size. This would lead to slower message transportation, as within each message at least the remaining workflow needs to be included because the services need to know which other service they have to call afterwards.

Planning in Clustered architecture

In the cluster architecture there exist two possibilities of workflow planning. Either there will be one node doing a central planning or the cluster managers do a decentralized planning as the service descriptions are distributed. In the first case a traditional approach of workflow planning using a central knowledge base can be applied as described in [2]. Peer [14] gives an overview of different AI planning techniques, that could be used.

Although there are already some first steps done in order to distribute the act of planning the workflow, in this paper only central planning is considered. The following diagram shows the message sequence of the planning and execution of the example application in the clustered architecture.

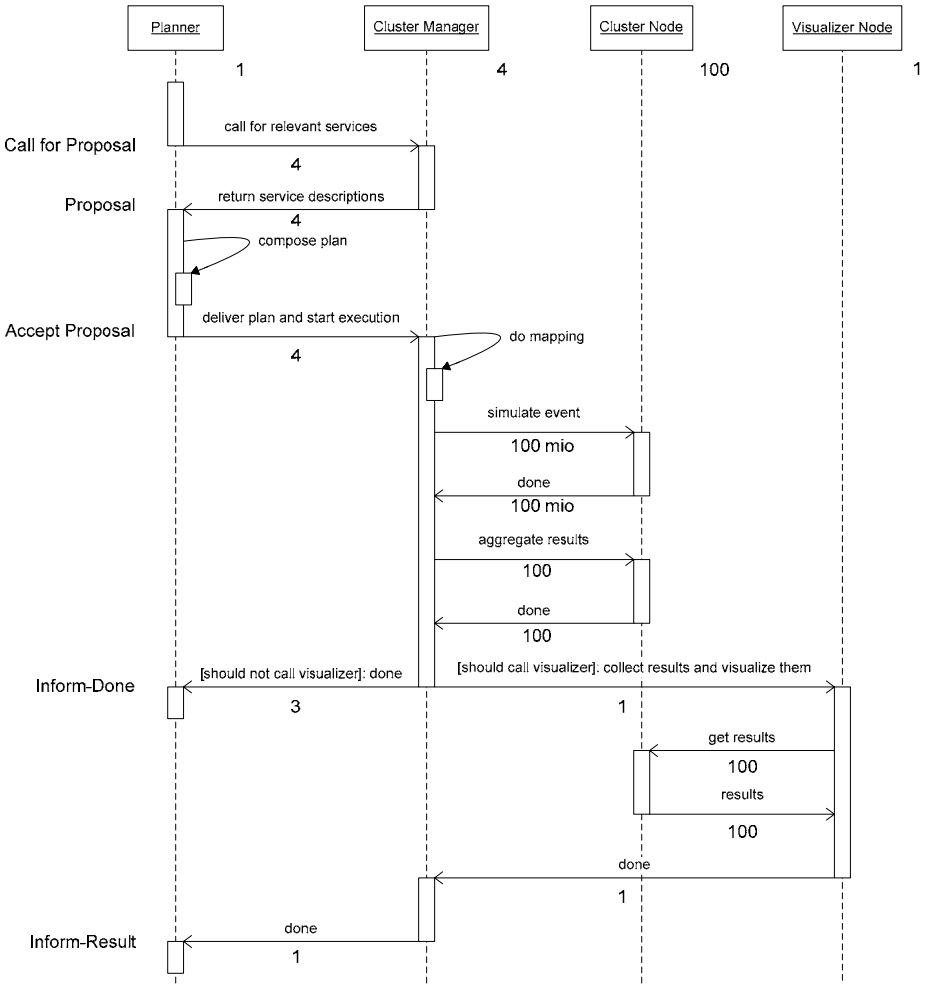


Fig. 4. Message Sequence in clustered architecture for example application

Again the numbers show how often a message is sent and how many instances of each participant exist. The Planner and the Cluster Manager can be realized using Agent Technology. Both use the FIPA Contract Net Interaction Protocol [8] for their communication. The different message types are shown on the left side of the diagram.

The planner first collects available service descriptions, which it uses to compose an abstract workflow. When the abstract workflow is composed the execution can start by delivering the workflow to the cluster managers. This step already contains the choice of which cluster should execute which tasks, which is also the first step in planning of the concrete workflow.

In order to execute the abstract workflow the cluster managers have to map it onto the cluster nodes. This includes finding concrete instances of the logically described tasks and data of the abstract workflow. When the concrete resources are chosen, it

becomes clear where additional data or service transports are needed. These tasks are then added to the concrete workflow.

As mapping a part of the abstract workflow to the cluster nodes is the same as mapping a whole workflow by a central workflow engine the same tools [15] could be used by the cluster manager. The optimizations shown in that paper can also be applied.

Dealing with failures

The main aspect why workflow planning should be distributed is the handling of failures. If a central workflow execution engine would fail, the whole application would fail. In a distributed system the failure of one component can be compensated by the others.

The interactions shown in the Peer to Peer architecture do not allow failure handling because no Peer would discover a failure of another Peer. Adding monitoring to the protocol would increase the already high number of messages. One thing is handling occurred failures while preventing failures is another. This can be done by mapping the tasks to the resources as late as possible, also called just in time planning [6]. As in the peer to peer architecture there is no difference between a service and a resource, the mapping is done by choosing the service. Thus just in time planning, and therefore preventing failures, is not possible in a peer to peer architecture.

Failure handling in a clustered architecture is easier, because inside each cluster there might exist a monitoring service. This service can detect the failure of a resource, trigger the re-planning of this task and thus recover the execution.

The failure of a cluster manager is somehow more complex. Either the same approach as in peer-to-peer systems can be applied, i.e. each predecessor cluster monitors the execution of its successor clusters. Or all cluster managers maintain the network by sending messages from time to time. If then a cluster has not sent any messages for a while, the other cluster managers assume a failure of that cluster. This failure is then recovered by choosing another cluster for the execution of the affected part of the workflow.

Just in time planning is also possible in the clustered architecture. The cluster manager does not need to map all tasks immediately but can do this step by step. Singh et. al. [15] presents different possibilities for dividing the mapping. That paper also shows how much time can be saved by doing just in time planning. Thus, just in time planning not only prevents failures but also reduces the overall execution time.

Conclusion

Comparing the two approaches by counting the messages exchanged might not be the only possible measurement. But it already indicates a performance problem in the peer to peer approach. Furthermore, considering that the shown peer to peer approach did not include just in time- and re-planning, the clustered architecture clearly promises better results. The fact that in the clustered architecture the already existing tools for workflow planning and execution can be included is another advantage.

Using the shown clustered approach enables the workflow execution engine to be distributed. Thus, several workflow based grid applications can be recovered when a part of the execution engine fails without loosing the already produced data. Applications no longer need to be restarted due to failures. This saves valuable time the scientists can use for their investigations.

References

1. Berners-Lee, T., Hendler, J. and Lassila, O. *The Semantic Web*. Scientific American, vol 284, no. 5, pp. 34-43, May 2001.
2. Blythe, J., Deelman, E. and Gil, Y. *Planning and Metadata on the Computational grid.* American Association for Artificial Intellgience, 2003.
3. Blythe, J., Deelman, E. and Gil, Y. *Planning for workflow construction and maintenance on the grid*. ICAPS 03 workshop on web services composition.
4. Buhler, P., Greenwood D. and Reitbauer, A. *A Multiagent Web Service Composition Engine*. ICSOC 2005
5. De Roure, D., Jennings, N. R. and Shadbolt N. R. *The Semantic grid: Past, Present, and Future*. Proceedings of the IEEE, Vol 93 No. 3 March 2005
6. Deelman, E. et al. *Pegasus: Mapping Scientific Workflows onto the grid*. In Across grids Conference 2004, Nicosia, Cyprus, 2004.
7. Eidelman, S. et al., *Monte Carlo Techniques*. In Physics Letters B592, 1 (2004)
8. FIPA Contract Net Interaction Protocol Specification: http://www.fipa.org/specs/fipa00029/index.html
9. Foster, I., Jennings, N. R. and Kesselman C. *Brain Meets Brawn: Why grid and Agent Need Each Other*. In Proceedings of 3rd International Conference on Autonomous Agents and Multi-Agent Systems, pp. 8-15, New York, USA, 2004.
10. Foster, I. and Kesselman, C. (eds.) *The grid: Blueprint for a New Computing Infrastructure (2nd Edition)*. Morgan Kaufmann, 2004.
11. Gil, Y. et. al. *Artificial Intelligence and grids: Workflow Planning and Beyond*. IEEE Intelligent Systems, special issue on E-Science, Jan / Feb 2004.
12. Lee, C. A., Michel, B. S., Deelman, E. and Blythe, J. *From Event-Driven Workflows Towards A Posteriori Computing*. In Future Generation grids as part of the Springer LNCS series, expected 2006.
13. Patel, J. et al. *Agent-Based Virtual Organisations for the grid*. Int. Journal of Multi-Agent and grid Systems, 2005.
14. Peer, J. *Web Service Composition as AI Planning – a Survey*. Second, revised version, Technical Report, Univ. of St.Gallen, 2005
15. Singh, G., Kesselman, C. and Deelman, E. *Optimizing grid-Based Workflow Execution*. In CS Tech report. 05-851. 2005, University of Southern California.
16. Wooldridge, M. *Agent-based Software Engineering*. In IEEE Proceedings on Software Engineering, 144(1), pages 26--37, February 1997
17. Yan, J., Yang, Y., Kowalczyk, R. and Nguyen, X. T. A Service Workflow Management Framework Based on Peer-to-Peer and Agent Technologies.

Case Study: Web Service Composition Framework

Sergey Smirnov

Hasso-Plattner-Institute for IT Systems Engineering
P. O. Box 90 04 60, 14440 Potsdam, Germany
sergey.smirnov@student.hpi.uni-potsdam.de

Abstract. This paper discusses the approach for automated service composition proposed by Pistore, Traverso, Bertoli and Marconi. The approach is based on using planning techniques to solve the automated composition problem. The authors aim to provide not only a theoretical framework, but also a tool which is capable of solving real composition tasks. The main framework assumptions are explained in this work. They are followed by a detailed description of a service composition mechanism. To illustrate how the approach works the reference example is introduced. Finally, pros and cons of the approach are discussed.

1 Introduction

The current trend in the software industry is to assemble applications from platform independent software components called web services which are available in the Internet. Universal Description, Discovery and Integration (UDDI) [1], Web Services Description Language (WSDL) [2] and Simple Object Access Protocol (SOAP) [3] define standards for service discovery, description and messaging.

An automated composition of existing web services is the promising technique to support business-to-business and enterprise application integration. The planning approach has been used in several research projects for solving the automated service composition problem [9]. The services which are available, *component services*, define the planning domain. The composition requirement is interpreted as the planning goal. The application of a planning algorithm is used to generate plans. The plan is a *composite service* interacting with the component services in order to achieve the planning goal.

This paper performs a case study and an analysis of the approach proposed by Pistore, Traverso, Bertoli and Marconi in [5]. Their research group has been working at the automated composition problem for several years. During this period their approach has evolved. The approach analyzed in this paper is the development of its predecessor introduced in [4, 6]. The introduction of the knowledge level allows avoiding the restrictions of the first approach.

The research group concentrates on the development of a framework which can have the industrial applicability and can be effectively implemented. In order to achieve this goal they work with the standard languages for business process

modeling and execution, such as Business Process Execution Language for Web Services (BPEL4WS). They also created tools which implement the approach. These tools can be used to perform the composition task.

This paper describes the assumptions accepted in the approach, steps which have to be executed in order to perform the service composition. The advantages and disadvantages of the framework are discussed.

The related concepts and specifications are introduced in Section 2. Section 3 describes the framework model and its underlying assumptions – the basis of the approach. Section 4 starts with the introduction of the reference example. The section aims to describe the approach step by step. The example illustrates each step. Finally, section 5 discusses the advantages and disadvantages of the framework.

2 Preliminaries

In this section the *BPEL4WS processes* and *state transition systems* are introduced. The main goals of the BPEL4WS specification are clarified and the types of BPEL4WS processes are named. The role of BPEL4WS processes in the approach is explained. The state transition system concept is defined and its role in the framework is discussed.

2.1 BPEL4WS Processes

Different research projects place emphasis on different aspects of the service composition problem. The intention of the framework authors is to develop the method for automated service composition which is compatible with the current industrial standards. This requirement implies that the services must have descriptions in the format accepted by the industry. The authors are of the opinion that BPEL4WS process descriptions meet this requirement in the best way. They assume that BPEL4WS process descriptions are able to provide enough information to create the planning domain. That is why the information from the BPEL4WS processes is used as the starting point of a composition.

BPEL4WS specification aims to ease the process integration in the intra-corporate and in the business-to-business spaces [8]. It supports a description of stateful behavior of Web Services. BPEL4WS distinguishes two types of business process descriptions: *abstract* and *executable*. The abstract process describes a behavior visible to the other parties involved in a business interaction. It determines the service partners, the process internal variables and operations. The executable process models the actual behavior of the participant in the interaction.

The framework discussed in this paper uses abstract processes of component services for obtaining information about the planning domain, while the plan is transformed into the executable process. Thus, the latter can be executed on the standard engine, such as the Active BPEL Open Engine or the Oracle BPEL Process Manager.

2.2 State Transition Systems

As it has been already discussed the component services represent the planning domain, while the composite service represents the plan. At the beginning the planning domain and the plan are in one of their initial states. The performing of an action causes the evolution of the planning domain (as well as the plan) from one state to a new one. Several types of actions can be distinguished. Messages sent from the composite service to the component services represent *input actions*, while messages received from the component services by the composite service – *output actions*. There are also internal actions of the services that are not observable for the environment. The internal actions of the component services are denoted with τ. The composite service can perform actions without communication with the component services – *private actions*.

The planner is able to work with the planning domain modeled as a *state transition system* (STS).

Definition 1 (state transition system)
A state transition system Σ is a tuple (S, S^0, I, O, A, R, L) where:

- S is the finite set of states;
- $S^0 \subseteq S$ is the set of initial states;
- I is the finite set of input actions;
- O is the finite set of output actions;
- A is the finite state of private actions;
- $R \subseteq S \times (I \cup O \cup A) \times S$ is the transition relation;
- $L: S \rightarrow Prop$ is the labeling function.

The transition relation defines the rules of the system evolution. The labeling function associates to each state the set of properties *Prop* holding in that state.

The planning domain is modeled as a STS. The output of the planner is the plan also modeled as STS Σ_c. Therefore, the planner has to create such a composite service Σ_c that "controls" the component services Σ, interacting with them, in order to satisfy the composition requirement.

3 Framework Assumptions

The discussed approach is based on the services interaction model. The model has a great impact on the framework architecture. That is why specific model features such as asynchronous interaction between services and their nondeterministic behavior are crucial for the approach understanding.

Web services have an **asynchronous nature**: the interaction between two parties is message-based. The message-based interaction between the domain and the plan is

reflected in the framework. The support of the message-based interaction by the framework is the milestone of the approach.

The framework assumes that the behavior of web services is **partially observable** and **nondeterministic**. The first source of nondeterministic behavior is the presence of nondeterministic transitions in the framework (there can be a nondeterministic choice between two transitions). The asynchronous framework also does not provide information about when internal transitions take place within the service component. This is the second source of nondeterministic behavior.

The nondeterministic behavior of services leads to the necessity to formalize the requirements with **extended goals**. The authors propose to use the EAGLE language [7] for this purpose. However, a temporal logic can also be applied.

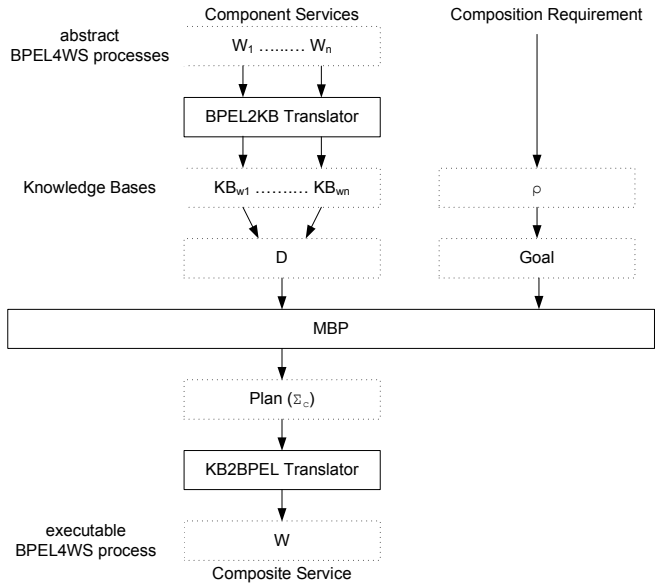


Figure 1. The general schema of the approach

4 Approach Description

The general schema of the approach is presented in Figure 1. Given the abstract BPEL4WS processes the knowledge-level models of the component services are created. The composition requirements are formalized as the goal. After this the composition problem is reformulated as the planning problem. The knowledge-level models of the component services together with the composition goal form the

planning problem. The planner constructs a plan describing the composition needed. Afterwards the plan is translated into the executable BPEL4WS process.

4.1 Example

The *P&S* example consists in providing a purchase and ship service by combining two independent existing services: a producer service *Producer* and a delivery service *Shipper*. Therefore, a user, also described as a service, may directly ask the composite service *P&S* to purchase a given item and deliver it at a given place (user's home). The certain protocols specify the interactions with the existing services. The Business Process Modeling Notation (BPMN) is used to show the interaction between the participants in Figure 2. The *P&S* has the goal to sell home-delivered goods (i.e., to reach the situation when the user has confirmed an order and the service has confirmed the corresponding suborders to the producer and the shipper), interacting with *Shipper*, *Producer*, and *User* according to their protocols. An example of the successful interaction could be the following:

1. *User* requests *P&S* for an item to be delivered to user's home;
2. *P&S* requests *Producer* for the size, cost, and production details of the item;
3. *P&S* requests *Shipper* for the delivery cost of an object of the defined size to the user's home;
4. *P&S* sends *User* an offer with the overall cost (plus an added cost for *P&S*);
5. *User* sends a confirmation of the order, which is dispatched by *P&S* to *Shipper* and *Producer*.

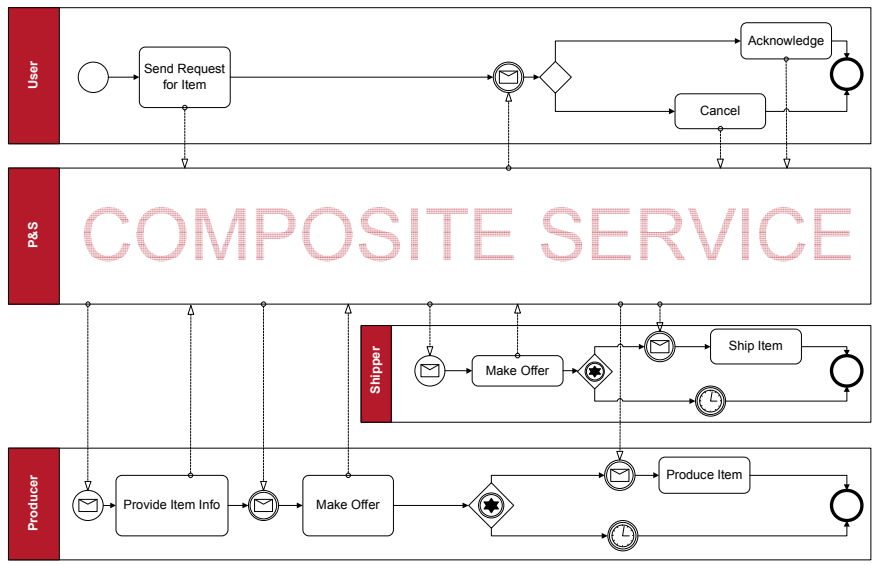


Figure 2. The P&S example with the component services (BPMN notation)

4.2 Component Services as Abstract BPEL4WS Processes

The planner is able to operate with the knowledge-level models of component services. Thus, knowledge-level models have to be extracted from the abstract BPEL4WS processes. The approach proposes that this operation must be automatic. This suits the idea of automatic composition and helps avoiding manual operations. That is why a formal method for the translation is proposed. The formal method is used by the corresponding tool to perform the translation.

The method extracting knowledge level information from the BPEL4WS processes is restricted to a subset of processes: there is a support for all BPEL4WS *basic* and *structured activities* (like *invoke*, *reply*, *receive*, *sequence*, *switch*, *while*, *flow* and *pick*), *assignments* and a limited form of *correlation*.

The fragment of the Shipper abstract process is presented here.

```
<process name="Shipper" abstractProcess="yes" … >
  . . .
  <variables>
    . . .
    <variable name="customer_size"
              type="tns:CustomerSize"/>
    <variable name="customer_loc“ type="tns:Location"/>
    <variable name="offer_delay" type="xsd:integer"/>
    <variable name="offer_cost" type="xsd:integer"/>
  </variables>
  . . .
</process>
```

In the fragment the variables are declared. This information will be modeled later at the knowledge level.

4.3 Knowledge Level Modeling

As it has already been said the discussed approach is the extension of its predecessor. The introduction of the knowledge-level model of component services is the main step in the framework evolution. The component service model on the knowledge level is called a *knowledge base*.

Definition 2 (Knowledge Base)
A knowledge base *KB* is a set of propositions of the following form:

- $K_v(x)$ where x is a variable with an abstract type;
- $K(x=v)$ where x is an enumerative variable and v is one of its possible values;
- $K(x=y)$ where x and y are two variables with the same type;
- $K(x=f(y_1, \ldots, y_n))$ where $x, y_1, \ldots, y_n$ are variables with an abstract type and f is a function compatible with the types of $x, y_1, \ldots, y_n$.

$K_v(x)$ is a proposition meaning that the value of variable x is known. $K(p)$ means that the proposition p is true.

The knowledge base KB is consistent if it does not contain contradictory knowledge propositions. Knowledge base *KB* is *closed under deduction* if it contains all the propositions that can be deduced from the propositions in *KB*.

The knowledge base of a component service is obtained from the variables, functions and types of the service defined in the abstract BPEL4WS process. The example of the KB for *Shipper* is the following:

```
KB={K(pc = waitAnswer),
Kv(customer_size),Kv(customer_loc),
K(offer_cost = costOf(customer_size, customer_loc)),
K(offer_delay = delayOf(customer_size, customer_loc)),
Kv(offer cost), Kv(offer delay)}
```

Now it is possible to explain at the knowledge level when the transition can be executed and what the transition execution means for the particular *KB*. The transition is a triple (C, a, E), where $C = (c_1 \wedge \ldots \wedge c_n)$ are conditions, a is its firing action and $E = (e_1; \ldots ; e_n)$ are its effects. Each transition has a firing action a. Action a denotes the execution of the action on the component service: it includes instantiation of appropriate variables on the side of the component service. However, the action can represent the interaction between the services. That is why we need to denote the same action with a_c to represent the execution of this action (with the instantiation of its own variables) on the side of the other interaction participant. To continue the concepts of *knowledge base restriction with condition* and *update of knowledge base with effects* are needed.

The restriction of a knowledge base *KB* with a condition *C*, denoted with *restrict(KB, C)*, is performed adding to *KB* the knowledge obtained from *C* and closing under deduction.

The update of a knowledge base *KB* with an effect *E* denoted with *update(KB, E)*, consists in performing the following steps: for each assignment in *E* remove from *KB* the knowledge on the modified variable, add the knowledge derived from the assignment and close the *KB* under deduction.

Now, it is possible to define when the certain transition t and a corresponding action a_c are applicable to the knowledge base KB, denoted with $applicable[t](KB, a_c)$:

- if a_c is an input action $a_c=i(x_1, \ldots, x_n)$ then the following conditions must hold: $K_v(x_1),\ldots, K_v(x_n) \subseteq KB$ and $restrict(KB, C)$ is consistent;
- if a_c is an output action $a_c=o(x_1, \ldots, x_n)$ then the $restrict(KB, C)$ must be consistent;
- if $a_c = a = \tau$ then $restrict(KB, C)$ must be consistent.

If the transition t and the action a_c are applicable to the knowledge base *KB* then it can be executed. The execution is denoted $KB' = exec[t](KB, a_c)$ and defined in the following way:

- if $a_c = \tau$ and $t=(C, \tau, E)$ then *KB* is restricted with *C* and updated with *E;*
- if $a_c = i(x_1, \ldots, x_n)$ and $t=(C, i(y_1, \ldots, y_n), E)$ then *KB* is restricted with *C* and updated with $y_1 := x_1,\ldots, y_n := x_n$ and *E;*

- if $a_c = o(x_1, \ldots, x_n)$ and $t=(C, o(y_1, \ldots, y_n), E)$ then KB is restricted with C and joined with $\{K_v(y_1), \ldots, K_v(y_n)\}$ and updated with $y_1 := x_1, \ldots, y_n := x_n$ and E.

The execution of a transition increases the knowledge base with the information in the conditions and in the effects of the transition.

4.4 The Goal

The composition requirements represent goals in the planning problem. The goals are formulated in the EAGLE language. The EAGLE language was designed to specify goals. It provides a set of modal operators of different strength, e.g. *TryReach*, *DoReach* and *Fail*. The nondeterministic service behavior makes us to use such operators: we can not be sure that it is possible to achieve the goal (i.e. the specific state of the system). The modal operators allow us to describe the other desirable states if the specified state can not be reached. The goal for *P&S* example can be formulated in the following way:

```
TryReach
     user.pc = SUCC ∧
      producer.pc = SUCC ∧ shipper.pc = SUCC ∧
      user.offer_cost =
       addCost(producer.costOf(user.article),
       shipper.costOf(producer.sizeOf(user.article),
       user.location))
```

After the goal has been formulated in the EAGLE language its knowledge-level representation has to be generated. It is done by flattening the functions in the goal: introducing auxiliary variables until only the basic propositions are left. The knowledge-level representation of the example mentioned above will be the following:

```
TryReach
  K(user.pc = SUCC) ∧ K(producer.pc = SUCC) ∧
  K(shipper.pc = SUCC) ∧
  K(user.offer_cost = goal.added_cost) ∧
  K(goal.added_cost = goal.addCost(goal.prod_cost,
goal.ship_cost)) ∧
  K(goal.prod_cost = producer.costOf(goal.user_art)) ∧
  K(goal.ship_cost = shipper.costOf(goal.prod_size,
goal.user_art)) ∧
  K(goal.user_art = user.article) ∧
  K(goal.prod_size = producer.sizeOf(goal.user_art)) ∧
  K(goal.user_loc = user.location)
```

The knowledge-level representation of the goal is the knowledge base, called the *knowledge base of the goal*.

4.5 Knowledge Level Planning

The planning problem at the knowledge level is formed by the knowledge base of each component service and the knowledge base of the goal. However, the planner is able to work with the planning problem modeled with a STS. That is why a relation between the concepts of the knowledge level and a STS of components has to be defined.

Definition 3 (Knowledge-level Planning Problem)
The planning problem Σ is an STS (S, S^0, I, O, A, R, L) defined as follows:

- the set of states S are all the possible KB defined on the set of typed variables $X=\bigcup_{i=0...n} X_i$ and on the set of typed functions $F=\bigcup_{i=0...n} F_i$ where $X_1, \ldots, X_n$ and $F_1, \ldots, F_n$ are the variables and functions of the component services, while X_0 and F_0 are those of the composition goal;
- $S^0 \subseteq S$ is the set of initial states corresponding to the initial knowledge bases $KB^1_0, \ldots, KB^n_0$, obtained from the initial assignments of the component services;
- I is the set of input actions $i(x_1,\ldots, x_n)$ such that $i(y_1, \ldots, y_n)$ is an input action in a transition of a component service and $x_1, \ldots, x_n$ are goal variables with the same type of service variables $y_1, \ldots, y_n$;
- O is the set of output actions $o(x_1,\ldots, x_n)$ such that $o(y_1, \ldots, y_n)$ is an output action in a transition of a component service and $x_1, \ldots, x_n$ are goal variables with the same type of service variables $y_1, \ldots, y_n$;
- A is the set of private actions $x_0 := f(x_1,\ldots, x_n)$ where f is a goal function and $x_0, x_1,\ldots, x_n$ are goal variables compatible with the type of f;
- R is the set of transitions $r = \langle s, a_c, s' \rangle$, with $s, s' \in S$, such that:
  - if a_c is an input, output or τ action, then there exists a $t = (C, a, E)$ in the sets of transitions of the component services such that $applicable[t](s, a_c)$ and $s' = exec[t](s, a_c)$;
  - if a_c is a private action of the form $x_0 := f(x_1,\ldots, x_n)$, then $K_v(x_1), \ldots, K_v(x_n) \in s$ and $s' = update(s, x_0 := f(x_1,\ldots, x_n))$;
- L is the trivial function associating to each state the set of propositions that hold in that state.

If the planning problem Σ is constructed it is possible to use the planning techniques to obtain a plan Σ_C. The model based planner (MBP) [10] is used to create the plan Σ_C – a controller for Σ.

4.6 The composite service as the executable BPEL4WS process

The plan Σ_C which is the output of the planner is modeled at the knowledge level. It models the interaction of the plan with the planning domain and the operations on the goal variables.

The tool provided by the framework allows translating the plan Σ_c to the executable BPEL4WS process. This process will describe the behavior of the composite service. Considering the *P&S* example the composite service – *P&S* service – is shown in Figure 3.

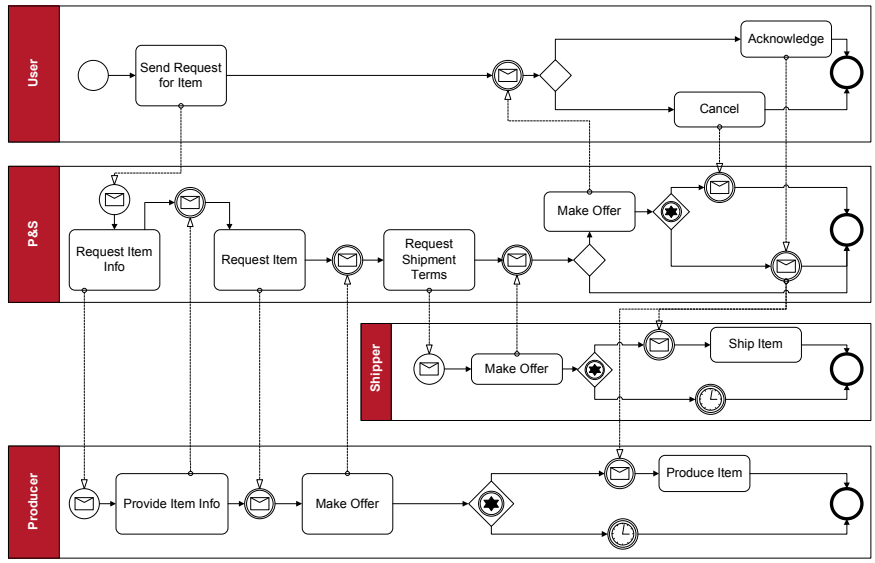


Figure 3. The P&S example with the component services and the composite service (BPMN notation)

5 Discussion

The main goal of the authors was to design a framework that can perform an automated composition of services described in industrial standard languages for business process modeling. To enable automatic composition the tools supporting the framework have been developed. To describe web services BPEL4WS was chosen. This leads to certain advantages and disadvantages of the framework.

The tools provided by the framework enable automated information translation from one representation to another, e.g. from BPEL4WS to KB. The whole set of tools allows performing all the translations in an automatic manner in a reasonable amount of time. This addresses the task of the automated composition and can be treated as the core advantage of the approach.

The choice of BPEL4WS for describing services and their interaction puts restrictions on the component services used in the composition. The BPEL4WS processes describe the behavior of services and contain syntactical descriptions of the

services. They do not provide any semantic information which is needed to construct knowledge-level models of services – knowledge bases. However, services are only described as BPEL4WS processes. That is why the names of operations and variables in the BPEL4WS processes have to be the source of the semantic information.

During the extraction of semantic information from the BPEL4WS process the certain meaning is assigned to each variable, according to its name. This can be illustrated by *Shipper* process and *KB* for this service: the variable *customer_size* is declared in the BPEL4WS process and there is a corresponding variable *customer_size* in the KB. This means that if there are two variables of the same type and name in different processes they will be given the same meaning at the knowledge level.

This approach assumes that variable names are consistent among different services. Therefore, to be able to use several independent services in the composition we should agree on the names. These agreements can not be always achieved. That is why the framework can not be used for a composition of any web service described with a BPEL4WS process. The approach can be useful in the environment where for creation of process descriptions administrative regulations exist, e.g. in an organization.

To remove the restriction discussed above an ontology can be used providing semantic descriptions of services. Together with BPEL4WS processes the ontology gives enough information for creating knowledge bases. Several languages exist for describing ontologies: Web Service Modeling Language (WSML) [11] or Web Ontology Language (OWL) [12]. They can be used in the framework to describe semantics of services.

6 Conclusions

In this work a case study of the web service composition framework has been presented. At the beginning the concept of state transition system has been introduced and the BPEL4WS processes have been discussed. The roles of STS, abstract and executable BPEL4WS processes in the framework have been explained. The main assumptions of the framework have been clarified. Afterwards the approach has been described. An example has been used to illustrate the work of the approach on each step. Finally the advantages and disadvantages of the approach have been discussed.

Being able to provide automatic composition of web services the framework has certain restrictions. The lack of semantic information in the descriptions of component services does not allow using any services described with BPEL4WS. The description of services used in the composition must be consistent in the sense of variable names and types. The introduction of an ontology may aid solving this problem.

References

[1] UDDI Version 3.0.2, http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm, 2004.

[2] Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl, 2001.

[3] SOAP Version 1.2. http://www.w3.org/TR/soap/, 2003.

[4] Pistore, M., Traverso, P., Bertoli, P., Marconi, A.: Automated Synthesis of Composite BPEL4WS Web Services, In Proceedings of IEEE International Conference on Web Services (ICWS) , icws , 2005, 293-301.

[5] Pistore, M., Marconi, A., Bertoli, P., Traverso, P.: Automated Composition of Web Services by Planning at the Knowledge Level, In Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), AAAI Press, 2005.

[6] Pistore, M., Traverso, P., Bertoli, P.: Automated Composition of Web Services by Planning in Asynchronous Domains. In Proceedings of the Fifteenth International Conference on Automated and Planning Scheduling (ICAPS), AAAI Press, 2005, 2-12.

[7] Dal Lago, U., Pistore, M., and Traverso, P.: Planning with a Language for Extended Goals. In Proceedings of American Association for Artificial Intelligence (AAAI), AAAI Press, 2002.

[8] Business Process Execution Language for Web Services version 1.1, ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf, 2003.

[9] Martinez, E., Lesperance, Y.: Web Service Composition as a Planning Task: Experiments using Knowledge-Based Planning. In Proceedings of International Conference on Automated and Planning Scheduling (ICAPS), AAAI Press, 2004, 62-69.

[10] Bertoli, P., Cimatti, A., Pistore, M., Roveri, M., Traverso, P.: MBP: a Model Based Planner. In Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), AAAI Press, 2001.

[11] Web Service Modeling Language. http://www.wsmo.org/wsml/, 2005.

[12] Web Ontology Language. http://www.w3.org/2004/OWL/, 2004.

Modeling Variability in State Machine Based Process Family Architectures for Automotive Systems

Kay Hammerl

Hasso Plattner Institute for Software Systems Engineering
at the University of Potsdam,
D-14482 Potsdam, Germany
Kay.Hammerl@hpi.uni-potsdam.de

Abstract. Nowadays' demand for process-oriented software in many domains on the one hand and the necessity for diversification of software products for fulfilling customers' needs and thereby ensuring a greater market penetration on the other hand, lead to the usage of process family engineering techniques. This paper describes, how variabilities of a process family can be modeled using UML 2.0 State Machines. For this purpose first a mapping of generic process elements to State Machine elements is provided. Following, in the main part, it is shown, how some of the most relevant variability mechanisms can be ported to this type of diagram and how the variabilities can be represented by using only lightweight UML extension mechanisms.

1 Introduction

Nowadays process oriented software gains more and more importance. Most innovations and profits of many business domains, for example the automotive sector, e-businesses or the ERP (Enterprise Resource Planning) sector, are based on this field. Furthermore in those sectors often products are produced which occur in several variable forms. This helps to really fulfill the customers' needs concerning functionality, which, as a result, leads to greater market penetration. So both sides could be pleased: the customer gets a personalized system (and maybe saves money, because he only gets the features he wants) and the enterprise gets more buyers. But dealing with personalized systems can be very expensive in the creation process, unless one is able to benefit from the fact that all those systems have more similarities than differences. The solution therefore is creating a product line - a mechanism well known from the manufacturing domain but relatively new to the software business. The authors of [1] state that empirical studies have shown that using software product lines can lead to serious organizational benefits such as

- achieving large-scale productivity gains
- improving time-to-market
- maintaining market presence

- sustaining unprecedented growth
- improving product quality
- increasing customer satisfaction
- achieving reuse goals
- enabling mass customization
- compensating for an inability to hire software engineers

To benefit from the idea of software product lines some advanced software development techniques like product family engineering have to be used. But the former research on this topic concentrated mostly on software built out of static diagrams like class or component diagrams. As a result the existing approaches are not able to serve the needs of process oriented software like the mentioned automotive, e-business and ERP systems.

For modeling software for embedded automotive control units, [2] reveals that a combination of UML 2.0 Activity Diagrams and State Machines is most suitable. As a consequence it is reasonable to research if these modeling techniques can be used in the case of product family engineering of process oriented software, shortly called process family engineering. Due to that the goal is to investigate if the differences of a process family can be modeled using the mentioned diagram types and how this could be done. The ability of Activity Diagrams to model variability has been already shown in [3] but for State Machines this topic was still open and will now be discussed in this paper.

This paper is structured as follows: Section 2 shows, how processes can be modeled with UML 2.0 State Machines [4], by mapping generic process elements to State Machine elements. In section 3 then some of the most cited variability mechanisms are explained and it is analyzed, if and how these mechanisms can be used in UML 2.0 State Machines concerning the utilization as well as the representation in the model. In section 4 a small example of a motor control unit process family architecture including State Machine variability is presented. Finally, section 5 gives a short summary and reveals open issues for future research.

2 Mapping from generic process elements to State Machine elements

One of the main goals of software product family engineering lies in preparing software for change by identifying the commonalities and variabilities within a family of products [5]. In [6] a recent definition for variability in software is presented:

> Software variability is the ability of a software system or artifact to be efficiently extended, changed, customized or configured for use in a particular context.

This variability of a software product family has to be described in a product family architecture which "acts as a reference architecture for the members of the

product family" [3] and specifies, which variability mechanism should be used at each variation point, the "location at which change can occur in a software product line artifact" [7]. Now, to investigate the use of State Machines in process family engineering, it is necessary to find rules and notations for adapting the concept of variability mechanisms. But before the porting of relevant variability mechanisms to UML 2.0 State Machines is describable, another step has to be taken, namely the definition how common generic process elements should be modeled in this type of diagram.

The most atomic element of a process is a *process step*. Since such a step is always an action or an activity, there exist two possibilities for embedding it into State Machines. Either it can be invoked in the do-activity of a Simple State or in an effect of a transition. Option one is in this case the better choice because of several reasons: First the diagram is more comprehensible for the viewer and second according to [8]

> effects should not be used as a long transaction mechanism. Their duration should be brief compared to the response time needed for external events. Otherwise, the system might be unable to respond in a timely manner.

The reason for why effects have to be short is that firing a transition is a run-to-completion step and thus cannot be interrupted and as a consequence no other events can be handled while processing this step. So, in State Machines process steps should be modeled as a Simple State with a do-activity invoking the corresponding activity.

The concept of having *subprocesses, encapsulated subprocesses* and their corresponding *interfaces* could be easily transfered to State Machines. The element Submachine State is used for encapsulating subprocesses. It defines an interface with its entry and exit point connection point references and allows including arbitrary submachine state machines (which would be the subprocesses) as long as they fit to the specified interface. Arguably, one could enlarge the interface by the entry, exit and do activity of the Submachine State for specifying a certain behavior the referenced submachines should implement. But since those are not known in the referenced submachine, this cannot be done. Moreover, specifying activities as an interface belongs to the activity view of a system rather than to the state machine view. Therefore, the variability should be modeled in an Activity Diagram with variability mechanism extensions and then, this diagram should be referenced in the do activity of a Simple State.

3 Variability mechanisms in UML 2.0 State Machines

In [3] and [9] variability mechanisms for generic processes are presented. This section will explain them briefly and analyze if and how these mechanisms can be used in UML 2.0 State Machines [4]. For every mechanism portable to State Machines the variation point and the corresponding variants will be identified. In this context, variation point means the invariant environment of the varying

process part in which different variants (varying process parts) could be embedded. A variability mechanism then describes the way how variants are bound to a variation point.

The variability mechanisms discussed in this paper can be categorized in three categories. At first there are basic variability mechanisms, which are all orthogonal to each other and build a foundation for other mechanisms derived by them. Second and third there are mechanisms, which are either derived by restriction or by combination of one or more (basic) variability mechanisms respectively. Figure 1 shows this classification.

Basic
- Addition / Omission / Replacement
- Encapsulation
- Parameterization
- Data Type Variability

Derived by Restriction
- Extension
- Inheritance

Derived by Combination
- Strategy Pattern

Fig. 1. Categorization of Variability Mechanisms

3.1 Addition, Omission, Replacement of Subprocesses and Process Steps

These basic variability mechanisms allow for the addition of a new process step or a whole subprocess at arbitrary positions of the process. Vice versa arbitrary process steps/subprocesses can be omitted. Last but not least any process step/subprocess can be replaced by another. As a result the **variants** used with these mechanisms can be any state (Simple State, Composite State, Submachine State) in case of State Machines, but with the restriction that an explicit entry into or an explicit exit from a Composite State has to be modeled with an entry or an exit point respectively and not with a transition leading from outside directly to a substate or vice versa.

The three mechanisms will be described in prose and additionally in a formal manner. For this purpose, some sets should be defined in advance[1]: Let V be the set of all Vertices (States, Pseudostates, ConnectionPointReferences) and T

[1] In order to define these sets and also later the mechanisms, the OCL (Object Constraint Language) notation [10] used in the UML specification [4] has been partly adopted. So, e.g. an attribute of an UML element can be accessed by using a "." (element.attributeName). This notation was preferred to introducing more functions for accessing the attributes, because it makes the formulas more easy to read

the set of all Transitions of a given state machine. Then we define $I(s \in V) = \{t \in T \mid t.target = s\}$ (the set of incoming transitions of s) and $O(s \in V) = \{t \in T \mid t.source = s\}$ (the set of outgoing transitions of s). Furthermore:

- $CP_{entry}(s \in V) = \{c \in V \mid c.kind = \#entryPoint \wedge c \in s.connectionPoint\}$: The set of all entry points of a Composite State.
- $CP_{exit}(s \in V) = \{c \in V \mid c.kind = \#exitPoint \wedge c \in s.connectionPoint\}$: The set of all exit points of a Composite State.
- $CPR_{entry}(s \in V) = \{c \in V \mid c.entry- > notEmpty() \wedge c \in s.connection\}$: The set of all connection point references to entry points of a Submachine State.
- $CPR_{exit}(s \in V) = \{c \in V \mid c.exit- > notEmpty() \wedge c \in s.connection\}$: The set of all connection point references to exit points of a Submachine State.
- $I_{CS}(s \in V) := I(s) \cup \{t \in T \mid t.target \in CP_{entry}(s)\}$: The set of all incoming transitions of a Composite State s (including transitions to entry points of s).
- $O_{CS}(s \in V) := O(s) \cup \{t \in T \mid t.source \in CP_{exit}(s)\}$: The set of all outgoing transitions of a Composite State s (including transitions from exit points of s).
- $I_{SMS}(s \in V) := I(s) \cup \{t \in T \mid t.target \in CPR_{entry}(s)\}$: The set of all incoming transitions of a Submachine State s (including transitions to entry connection point references of s).
- $O_{SMS}(s \in V) := O(s) \cup \{t \in T \mid t.source \in CPR_{exit}(s)\}$: The set of all outgoing transitions of a Submachine State s (including transitions from exit connection point references of s).

Addition This mechanism allows for the addition of a new state to a state machine. This state can be inserted between two vertices, if those are connected by a transition. The **variation point** therefore is defined by at least two vertices. 'At least', because it is feasible to insert one state between more than two vertices at the same time, if there exist transitions from different vertices to one common vertex. Figure 2 shows the simplest use case of this variability mechanism: When state A is completed, transition a fires and B is entered. In this situation an additional state C could be easily inserted by setting it as the destination of transition a and creating a new transition a' which leads from C to B.

This simple mechanism also works if there are more than one transition between the two states. Figure 3 shows such a case. Depending on the signal which arrives, A is left and a proper action is performed while moving on to B (or C respectively after adding it). The same would hold for a case in which the both shown transitions would not leave from one single state A but instead one from a state *A1* and the other from a state *A2*.

So, the general proceeding for adding a new state C between a not empty set of existing vertices A and one existing vertex B is to select transitions which have a vertex of A as source and B as destination and relocate those in such way, that the new state C is set as destination. Then a new transition must be created with source C and destination B. All other transitions which may start or end

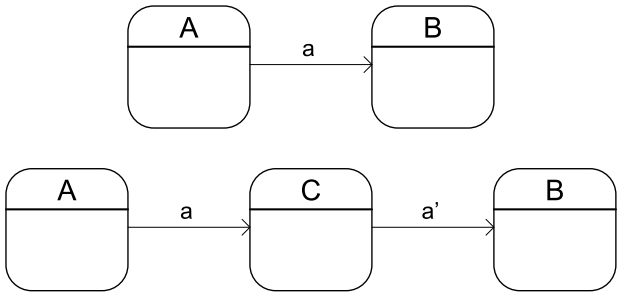


Fig. 2. Adding a state C

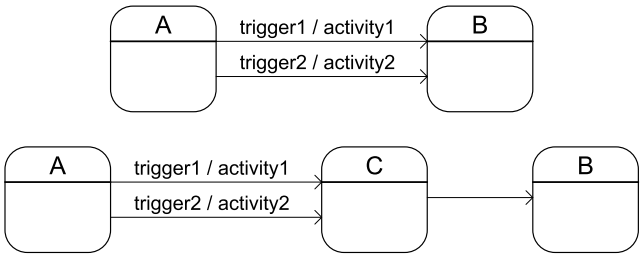


Fig. 3. Adding a state C (complex case)

in either a vertex of A or B are kept the way they are. This works for adding Simple States as well as for adding Composite States and Submachine States if the default entry and exit is used. If one or more Connection Points (in case of a Composite State) or one or more Connection Point References (in case of a Submachine State) should be used for entering the new state, this information has to be provided alongside the information where to insert it. Furthermore for every existing exit point of C a transition to B has to be created to prevent an ill-formed state machine.

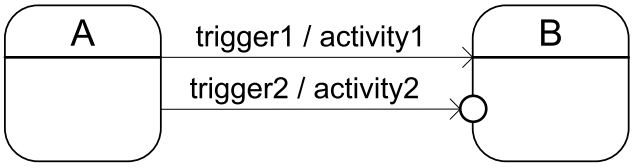


Fig. 4. Addition Problem

Using this algorithm has the prerequisite that all transitions coming from A and ending in B have exactly the same destination. For cases like the one shown

in figure 4 this does not work, because it is impossible to say where B should be entered after adding C to the state machine. This seems to be a limitation of the mechanism at the first sight, but a closer look reveals that the entry point of B is actually a state itself (a so called *pseudo state*). Let us name it D and we see that the first transition leads from A to B and the second from A to D. So in fact we have two variation points instead of one. Accordingly, a solution for such occasions is to add C multiple times.

Formally specified: Adding a state s_{new} between a set of vertices A and one vertex s_{target} is applicable iff $|A| \geq 1 \wedge \forall s \in A : \exists t \in T : t.source = s \wedge t.target = s_{target}$. If the precondition holds, let $T_1 = \{t \in T \mid t.source \in A \wedge t.target = s_{target}\}$. Now, every transition which should not be affected by the Addition has to be removed from T_1. Another possibility would be to start directly with a defined set T_1 of transitions instead of specifying the set A. Then, of course, the precondition is as follows: $|T_1| \geq 1 \wedge \forall t \in T_1 : t.target = s_{target}$. If s_{new} is a Simple State or a Composite / Submachine State without Connection Points / Connection Point References then $\forall t \in T_1 : t.target = s_{new}$ and a new transition t_{new} with $t_{new}.source = s_{new} \wedge t_{new}.target = s_{target}$ will be created and added to T. Adding Composite States or Submachine States with Connection Points or Connection Point References respectively need an additional mapping function $f : T_1 \rightarrow CP_{entry}(s_{new}) \cup s_{new}$, which assigns every transition from T_1 an entry point from s_{new}. The Addition then is defined by $\forall t \in T_1 : t.target = f(t)$ and $\forall c \in CP_{exit}(s_{new}) \cup s_{new}$ a new transition t_{new} with $t.source = c \wedge t.target = s_{target}$ has to be created. For adding Submachine States it is the same but with CPR_{entry} and CPR_{exit} instead of CP_{entry} and CP_{exit}.

Omission In this mechanism the **variation point** is defined by all sources of incoming transitions and the destination of the outgoing transition(s) of the state which should be omitted. It is very important that all outgoing transitions of the omissible state have the same destination. Otherwise it is impossible to create a valid state machine automatically. This fact can easiest be described by the example in figure 5: C is left, when either the event *b1* or the event *b2* occurs. Now, C should be omitted. At first the question arises, where to connect the transition leaving A. Should it lead to *B1* or to *B2*? This question can not be answered automatically. A possible solution (at first glance) seems to be to create two transitions starting from A, one leading to *B1* the other to *B2*, with each having the conjunction of the event that triggers the leaving of A (a) and the event *b1* respectively *b2* as trigger. But since it is impossible in UML State Machines to concatenate two events this 'solution' is not realizable. Moreover, even if it were possible, closer inspection would show, that this proposition does not work and could lead to deadlock situations. This is due to the fact that events which force a transition from one state to another can have a strong connection with the state which is left. Imagine the case that in the do-activity of C an email is sent to a customer and afterwards the system waits for either a return

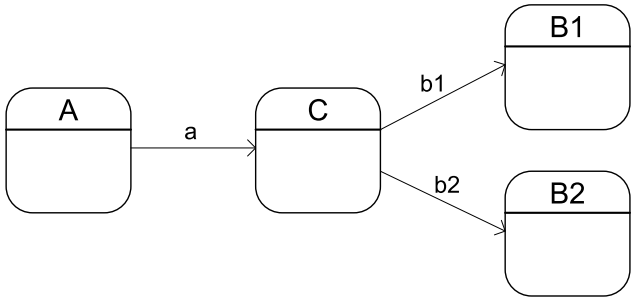


Fig. 5. Omission impossible

email (*b1*) or a incoming phone call (*b2*). Now, without having C sending the email first, neither *b1* nor *b2* would arrive and A could never been left.

Reading the figures 2 and 3 bottom-up gives an impression of the way Omission works, because it can be seen as the inversion of Addition. Generally this variability method is applicable to a state C, if C has an arbitrary number of incoming transitions (which need not to have the same source vertex) and an arbitrary number of outgoing transitions which must have the same destination (let us call it B). Omitting this state C means to set B as destination of all transitions formerly leading to C and forgetting about all the transitions which leave from C. But deleting all those transitions from the state machine also leads to the deletion of all the events that formerly triggered them. Referring to the send-email example from above this seems pretty reasonable, but what if an occurrence of an event is a precondition for entering B (B must not be entered without the event occurrence)? In that case Omission can not be used and instead C has to be replaced with a state without do-activity using the variability mechanism Replacement. It should be noted here, that no formal rules can be specified for deciding, if an event "belongs" to the state which is left due to the event occurrence or to the state which is entered afterwards, because it depends on the semantic context. Therefore, the designer has to choose the proper mechanism accordingly.

Formally specified: A Simple State s_{omit} can be omitted from a State Machine, iff $\forall t_1, t_2 \in O(s_{omit}) : t_1.target = t_2.target$. Since all transitions from $O(s_{omit})$ lead to one single vertex, let $s_{target} = t.target, t \in O(s_{omit})$. Then $\forall t \in I(s_{omit}) : t.target = s_{target}$ and $T = T - O(s_{omit})$. A Composite State s_{omit} can be omitted, iff $\forall t_1, t_2 \in O_{CS}(s_{omit}) : t_1.target = t_2.target$. If this holds, the Omission is defined by $\forall t \in I_{CS}(s_{omit}) : t.target = s_{target}$, with s_{target} defined like in case of omitting a Simple State, and $T = T - O_{CS}(s_{omit})$. The same rules hold for omitting Submachine States, but O_{SMS} and I_{SMS} have to be used instead of O_{CS} and I_{CS}.

Replacement For Replacement the **variation point** can be defined as the union of all sources of incoming transitions and all targets of outgoing transitions of the state that should be replaced. The utilization of this mechanism is rather simple: The state to be replaced is deleted from every transition where it is either source or target and instead the new state takes the place of it. If at least the replacing state is a Composite or Submachine State, additional information is needed, which maps the entry and exit points of the replaced state on entry and exit points of the replacing state. Furthermore, one should be aware of the fact that all events which formerly triggered the leaving of the replaced state remain untouched during replacement. Therefore, it is necessary to ensure that those events are not correlated with the replaced state and still can occur after the replacement.

Formally specified: A Replacement of a state s_{old} by another state s_{new} is defined as $\forall t \in O(s_{old}) : t.source = s_{new}$ and $\forall t \in I(s_{old}) : t.target = s_{new}$, if both s_{old} and s_{new} are Simple States. If only s_{old} is a Composite State or a Submachine State, then the same holds but additionally the transitions in and out of entry or exit points have to be redirected. So instead of $O(s_{old})$ and $I(s_{old})$, $O_{CS}(s_{old})$ and $I_{CS}(s_{old})$ or $O_{SMS}(s_{old})$ and $I_{SMS}(s_{old})$ have to be inserted in the formula above. If both states are Composite States, then additionally two mapping functions $f_1 : s_{old} \cup CP_{entry}(s_{old}) \rightarrow s_{new} \cup CP_{entry}(s_{new})$ and $f_2 : s_{old} \cup CP_{exit}(s_{old}) \rightarrow s_{new} \cup CP_{exit}(s_{new})$ have to be specified, which map all possible entry and exit points of s_{old} to entry and exit points of s_{new}. With those two functions the Replacement is described as follows: $\forall t \in O_{CS}(s_{old}) : t.source = f_2(t.source)$ and $\forall t \in I_{CS}(s_{old}) : t.target = f_1(t.target)$. For every other possible combination of the state types of s_{old} and s_{new}, the domain and the co-domain of f_1 and f_2 have to be changed accordingly and the proper set of incoming and outgoing transitions of s_{old} has to be taken (I,I_{CS}, I_{SMS}, O,O_{CS}, O_{SMS}).

3.2 Encapsulation of Varying Subprocesses

This mechanism describes the insertion of variant-specific subprocess implementations into a fixed subprocess interface. Contrary to the Extension mechanism described later on, in this case there has to be an implementation to this interface at any time.

In State Machines encapsulated subprocesses consist of a submachine state and a corresponding submachine which the submachine state includes. So the variability mechanism Encapsulation can be easily adapted for State Machines. The **variation point** is a submachine state. Its entry- and exit-points define an interface. Therefore, the corresponding submachines, which provide different implementations, are the **variants**.

3.3 Extension / Extension Points

This mechanism is used to extend a (sub)process at predefined points. These so called Extension Points allow to include additional optional behavior.

The **variation point** in this case is the Extension Point, which can be mapped to a submachine state regarding State Machines. By default the included submachine is a submachine with no behavior, but, depending on the favored process, another submachine could be selected instead from a set of compatible submachines (the **variants**). Therefore, the Extension mechanism is derived from the Encapsulation mechanism by restricting the default case.

3.4 Parameterization

Parameterization allows for the creation of variant subprocesses according to the values of certain parameters. Generally, one parameter can be seen as a variation point and its assignable values are the variants. For the subprocess this variability mechanism has the consequence, that all possible process parts have to be present in the subprocess.

In [7] (page 188 ff.) an approach for using Parameterization in State Machines is presented: First a state machine including all variabilities has to be designed. The different variant-specific transitions, actions and activities in this overall state machine are marked with Boolean guard conditions (according to the feature to which they belong), which means that they can only fire/execute if their guard condition is true. By setting the parameters to true or false variant subprocesses are generated.

This proposal seems reasonable but has the problem that marking actions and activities with guard conditions is not UML-conform. One could possibly argue that all occurring actions and activities in State Machines are of type 'Behavior' and therefore can have preconditions which could be interpreted in some way as guards. But as a hurt precondition is a sign of an error in the system, this workaround cannot be used. As a consequence, Parameterization can only be used in UML 2.0 State Machines in the following manner: At first the parameters have to be defined in the class the parameterized state machine belongs to. Then those parameters can be used in guard conditions of this state machine. In contrast to the suggestion of [7] the parameters need not to be boolean and therefore allow a more flexible parameterization. The problem with non-guardable actions and activities can be solved by introducing some more states and then, instead of guarding the do-activities, guarding the transitions to those states.

For Parameterization the **variation point** is a certain parameter and the **variants** are the possible values of this parameter. So, both the variation point and the variants are specified outside of the state machine and can therefore not be seen directly in the diagram. There, only the use of a parameter in a guard condition is apparent. This then marks a secondary variation point. Furthermore, the Parameterization variability mechanism disregards if a certain combination of parameter assignments leads to a valid and deadlock-free state machine. So it is necessary to keep track of allowed or unallowed parameter combinations in another place.

3.5 Data Type Variability

With Data Type Variability the type of the processed data could be changed. Since UML State Machines only show the behavioral aspect of a system, this variability mechanism can not be adapted. But as it is possible to include UML Activity Diagrams into a State Machine, for example in the do-activity of a state, Data Type Variability could be modeled there.

3.6 Inheritance

As stated in [3], Inheritance on its own is not a variability mechanism, because no variants are bound to variation points. But nevertheless it is a utility needed by the Strategy Pattern, which is explained in the next subsection. How Inheritance for UML State Machines works and what modifications to an inherited state machine are allowed is explained in [7] (page 181 ff.).

3.7 Strategy Pattern

The Strategy Pattern is used to assign different variable subprocesses, which are created by inheritance from an initial abstract process implementation, to a certain variation point. The mapping of this variability mechanism to State Machines is comparable with the variability mechanisms Extension/Extension Point and Encapsulation of Subprocesses. The **variation point** also is a submachine state and the variants are corresponding compatible submachines. The differences are that all **variants** are derived from one abstract submachine and that always one of the variants has to be chosen. Therefore, the Strategy Pattern can be seen as a combination of Encapsulation and Inheritance.

3.8 Notation

So far some "architecturally relevant variability mechanisms" ([3]) have been presented and their possible realization with UML State Machines has been shown. But still more information is needed to be able to create process family implementation artifacts based on a process family architecture. The variation points as well as the variants and their assignment to a certain variation point have to be indicated. Moreover, the variability mechanism which is used to bind a variant to a variation point has to be represented in some way. In [3] a notation for UML Activity Diagrams was introduced, which easily extends the Activity Diagram specification to support variability modeling. This extension can also be used in UML State Machines: All State Machine elements representing variation points are marked with the stereotype ≪VarPoint≫. On the other hand the variants are identified by the stereotype ≪Variant≫. The assignment of a variant to a variation point then is realized using UML Dependencies which carry the name of the variability mechanism as a stereotype, as listed in table 1. Furthermore, the use of another stereotype ≪Variable≫ is suggested to identify generally, if an element is in some way affected by variability. For example this is helpful to

indicate the process parts influenced by the utilization of the Parameterization variability mechanism, because as mentioned before its variation point as well as its variants are not visible in the diagram.

Variability Mechanism	**Stereotype**
Encapsulation of Varying Subprocesses	≪Implementation≫
Addition, Omission of Encapsulated Components	≪Option≫
Replacement of Encapsulated Components	≪Replacement≫
Parameterization	— (not visible in the SM)
Data Type Variability	— (not supported)
Inheritance	≪Inheritance≫
Strategy Pattern	≪StrategyPattern≫
Extension/Extension Points	≪Extension≫

Table 1. Stereotypes for identification of variability mechanisms in UML State Machines (based on [3])

Although most of the entries in the table should be comprehensible and pursuable immediately, some may seem kind of inconsistent to what was said in the corresponding subsections. That are the mechanisms Addition, Omission and Replacement of Varying Components. For those, the described variation points consist of more than one element and that does not fit to the concept of making one State Machine element explicitly a variation point by giving it the ≪VarPoint≫ stereotype. For Addition and Omission the solution is to add to the process all states which are variants concerning the variability mechanism Addition. Then, these states and all variants regarding the variability mechanism Omission will be marked with three stereotypes: ≪VarPoint≫, ≪Variant≫ and ≪Option≫. This means that such a state is a variation point and a variant at the same time and because of the ≪Option≫ stereotype can be either added to or omitted from the process. It is much the same for Replacement: all replaceable states have both the ≪VarPoint≫ and the ≪Variant≫ but not the ≪Option≫ stereotype. The replacing states on the other hand are only marked with ≪Variant≫.

3.9 Implementation Strategies

State Machines are most frequently implemented by using nested switch/case- or cascading if-constructs. Now it has to be investigated how the variability mechanisms explained so far could be applied. In [11] some implementation mechanisms for variabilities are described and compared:

- Conditional Compilation (Scatterd or Structured Preprocesser Directives, C++ Conditional Compilation)
- Dynamic Polymorphism (Polymorphism with Subclasses, Strategy with Template Parameters or Subclasses, Decorator)

– Configuration (Selection on Configuration Data, Interpreter)

It is at this point assumed that the first possibility, the conditional compilation, is adaptable for implementing State Machine variability. With this technique some parts of the completed program can be chosen for compilation. This is either done by scattered `#ifdef`s, in a logical unit (e.g. in a function) encapsulated `#if`s or constant variables. Since using those methods is only a matter of removing not needed variant-depending code during compilation, there should be no restrictions using them for this purpose. At least the mechanisms Addition, Omission and Replacement may be implemented that way. Moreover the third issue, configuration, seems feasible. In this approach data is read from an arbitrary configuration file and is then asked either in an `if` or a `switch` statement, to decide which code fragment should be executed. As said in the first sentence of this subsection, those constructs are the main concepts in most state machine implementations anyway, which would allow an easy integration of this implementing variability mechanism. Obviously, Parameterization would be served best by this solution because its application requires the presence of all options in the code which would be the case. Whether dynamic polymorphism can be used for implementing State Machine variability depends surely on the way a State Machine gets implemented. This and also if this paragraph's assumptions concerning Conditional Compilation and Configuration hold needs further theoretical as well as practical investigations. In such studies also a closer look at Aspect-Oriented Programming and its potentials should be taken.

4 Example: Variability in Automotive Systems

Now a little example from the automotive domain should demonstrate the application of three of the described variability mechanisms: Encapsulation, Addition/Omission and Parameterization.

Figure 6 shows the high level motor control process family architecture. At this point, only one variation point is apparent, the Submachine State *Starting Motor*, but because of the stereotype ≪Variable≫ of the Composite State *Motor is running*, one knows that in this Composite State some more variabilities can be found. However, before opening the engine hood, a closer look at the *Starting Motor* Submachine State and the modeled variability should be taken. It can be seen that there are two variants for this variation point and that the variability mechanism Encapsulation of Varying Subprocesses is used. The one variant models the starting of the motor with the existence of an immobilizer and the other the starting without one. By default the version without immobilizer is bound.

In figure 7, the hood has been opened and the internals of the *Motor is running* Composite State are revealed. Now, two uses of variability mechanisms can be observed. The first is in the bottom region of the state. The Composite State *Distance Checker is running* is marked with the three stereotypes ≪VarPoint≫, ≪Variant≫ and ≪Option≫ and marks therefore a use of the variability mechanism Addition/Omission. As a consequence, this state can either be added to or

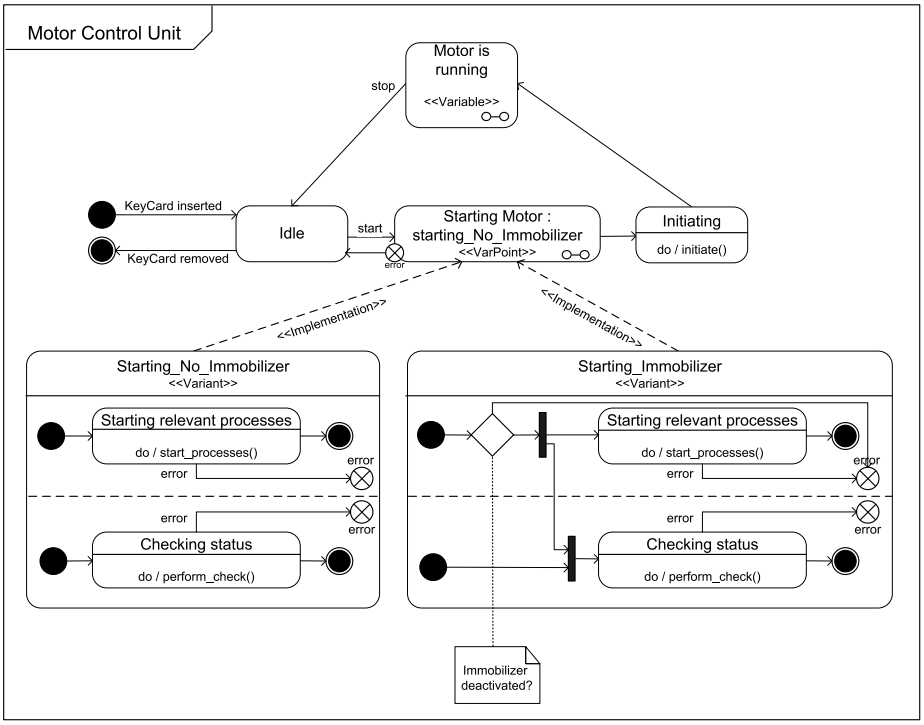


Fig. 6. High-level process with variability depending on the presence of an immobilizer

omitted from the state machine. The first case stands for a car equipped with a distance checker and therefore, by entering the *Motor is running* state and thus activating the two regions, the state would be entered and the distance checking and corresponding speed adjusting would be performed as long as the car is running. In the second case, which represents a car without that feature, the region would be entered and the transition from the initial state would directly lead to the final state of that region. The second variability in this diagram can be found in the upper region. The optionality of the feature cruise control is modeled with the aid of the variability mechanism Parameterization. As mentioned in section 3, the variation point lies outside the diagram, so the use of this mechanism could only be observed by the ≪Variable≫ stereotype of the transition using the parameter *cruiseAvailable* in a guard condition.

In this example, the three features immobilizer, cruise control and distance checker are independent and not related to each other which means that they can be each present or not present in any possible combination. But if there were any relation, e.g. that the distance checker requires the presence of a cruise control or that the immobilizer and the cruise control cannot be present at the same time, then those dependencies would have to be described outside of the State Machine in another document.

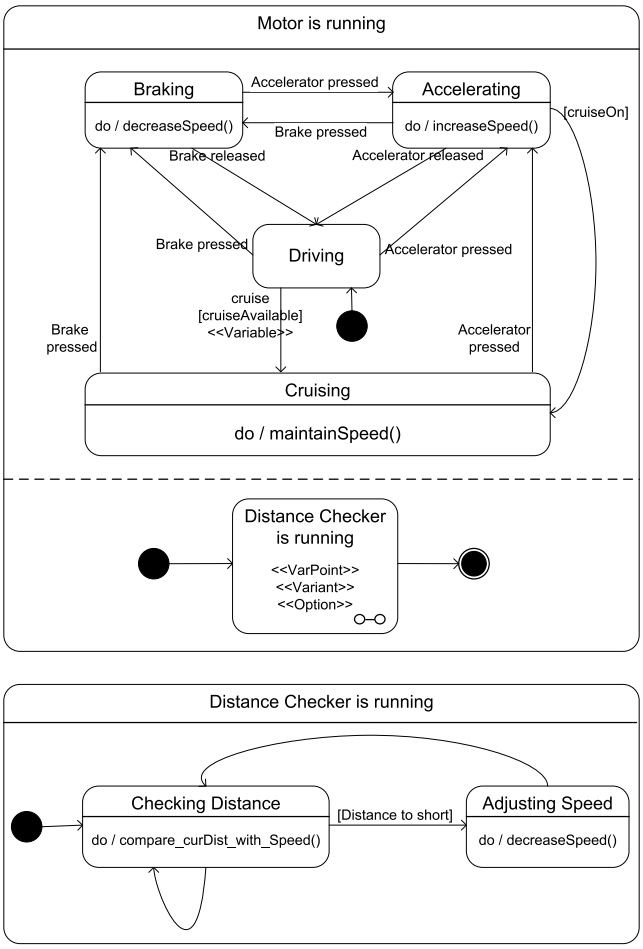


Fig. 7. Composite State 'Motor is running'

5 Conclusions

In section 1 the increasing demand for process oriented software and the resulting need of product family engineering approaches for processes has been depicted. Thereby the investigation of the capability of UML 2.0 State Machines for modeling variabilities has been motivated. After a general mapping of common generic process elements to State Machine elements, some of the most important variability mechanisms were explained and their possible utilization in this diagram type was analyzed and described. Furthermore, a notation was provided, which allows a seamless integration of this variability concept into UML State Machines by using only lightweight UML extensions. Additionally,

a short look on implementation strategies has been taken and in an example process of a motor control unit, the results of section 3 have been demonstrated.

Future research now has to take a deeper look at implementation mechanisms for variability and the dependencies between them and the variability mechanisms mentioned in this paper. It has to be investigated in what way the choice of a certain variability mechanism in the process family architecture enforces the use of a certain implementation mechanism. Moreover, it should be researched whether the utilization of the same variability mechanism in different modeling notations, e.g. in UML Activity Diagrams and State Machines, leads to the same set of possible implementation strategies or not.

References

1. Paul Clements, Linda Northrop. Software Product Lines: Practices and Patterns. Addison-Wesley, 2001
2. Ernst Richter, Arnd Schnieders, Jens Weiland. Process analysis and modeling in the automotive domain (in German), PESOA-Report No. TR 07/2004. DaimlerChrysler Research and Technology, Hasso-Plattner-Institute for IT Systems Engineering. Tech. Report, October 2004
3. Arnd Schnieders. Variability Mechanism Centeric Process Family Architectures. In Proceedings of the 13th Annual IEEE International Conference on the Engineering of Computer Based Systems ECBS 2006.IEEE Computer Society Press, 2006
4. UML 2.0 Superstructure Specification. Object Management Group. August 2005
5. James Coplien, Daniel Hoffman and David. Weiss. Commonality and Variability in Software Engineering. IEEE Software, November/December 1998
6. Mikael Svahnberg, Jilles van Gurp and Jan Bosch. A Taxonomy of Variability Realization Techniques. Software Practice & Experience, vol 35, pp. 1-50, 2005.
7. Hassan Gomaa. Designing Software Product Lines with UML: From Use Cases to Pattern-based Software Architectures. Addison-Wesley, 2005
8. James Rumbaugh, Ivar Jacobsen, Grady Booch. The Unified Modeling Language Reference Manual, 2nd ed. Addison-Wesley, 2005
9. Frank Puhlmann, Arnd Schnieders, Jens Weiland, Mathias Weske. Variability Mechanisms for Process Models, PESOA-Report No. TR 17/2005. Daimler-Chrysler Research and Technology, Hasso-Plattner-Institute for IT Systems Engineering. Tech. Report, June 2005
10. OCL 2.0 Specification. Object Management Group. June 2005
11. Claudia Fritsch, Andreas Lehn, Reza Rashidi, Dr. Thomas Strohm. Variability Implementation Mechanisms: A Catalogue (internal paper). Robert Bosch GmbH. Tech. Report 2003

System Integration via Service Enabling

Alexander Saar

Hasso-Plattner-Institute for IT-Systems Engineering
Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany
alexander.saar@hpi.uni-potsdam.de

Abstract. A few years ago the research field of Business Process Management emerged to cope with the challenges of enterprises regarding flexibility of their business processes. To achieve this goal several technologies for modeling, analysis, execution and management of business processes have been developed.
For the integration of IT-systems into such business processes web service technology is preferred, but in the majority of the cases the systems that form IT-landscapes are not available via service-oriented interfaces. This makes it hard to integrate these systems into business processes.
The paper in hand proposes a methodology for service enabling, which means the adaption or modification of non-service oriented systems in order to provide their functionality as services. The methodology provides a structured and repeatable procedure that can be used for the service-oriented integration of systems into business processes.
For evaluation purposes the methodology was applied in a case study. The case study considers a common service enabling scenario and shows how the methodology can be used for integration of IT-systems in a service-oriented manner.

1 Introduction

Today enterprises are under tremendous pressure to flexibly react on changes of their market, upcoming trends, new competitors or other changes in their business. For this purpose the field of *Business Process Management* (BPM, [1]) provides a set of technologies for implementing infrastructures that are able to manage business processes and that are flexible enough to react on permanent changes. These technologies include modeling, analysis, execution, composition, management and monitoring aspects of business processes.

To achieve the goal of integrating IT-systems into business processes BPM goes hand in hand with the web service community, which provides the capability for standardized execution of activities during the enactment of business processes. Therefore it is necessary that the IT-systems provide service-oriented interfaces. This is often not the case because IT-landscapes of enterprises today consist of poorly documented and structured legacy systems, web-based client-server or multi-tier systems, collaboration tools like instant messaging and document-based desktop applications.

In the majority of the cases these applications are mission critical for the enterprises and rewriting them in a service oriented manner is too expensive and risky. This leads to the question how to enable traditional non-service oriented applications to provide their functionality as services.

Another point is that *Service Oriented Architecture* (SOA, [2]) has the potential to change drastically the way in which software is developed, because global markets for services provide possibilities for reuse at a much greater scale. This leads to the need for providing competitive services to be offered on these markets, but lots of what makes a service competitive is already implemented in the traditional systems that form today's IT-landscape. To make use of the potential provided by SOA we need methodologies for migrating those traditional systems to service-oriented systems so that they can be adapted in a business process (see [3] for more information about motivation of migrating to service-oriented systems).

The step of finding and providing service-oriented interfaces for non-service-oriented systems is called *Service Enabling*. Until now there is no common definition of service enabling in the public. For this reason it was necessary to find a common definition that can be used for this work. The definition of the term *Service Enabling* that is used in this paper is given below.

> ***Service Enabling*** *is the adaption or modification of non-service oriented (typically monolithic) systems in order to expose their functionality via service-oriented interfaces.*

Important viewpoints for service enabling are:

Business Aspects like analysis of processes, selection of mission-crititical applications for migration to service-oriented systems, risk management and coordination between IT and management departments.

Technological aspects like software analysis (e.g. re-engineering techniques), migration and integration methodologies.

This work aims on the development of a process-driven service enabling methodology. The methodology uses the processes where the service enabled IT-systems should be integrated later on as starting point for service enabling. This process-centric view helps to find good services that contribute to the overall quality and performance of the process.

The methodology can be used as a development process for integrating IT-systems into business processes as well as providing services on the global market. It provides a structured and repeatable procedure that helps enterprises to migrate their IT-landscape to service-oriented processing.

The remainder of this paper is organized as follows. First related work is investigated in section 2. After that the proposed service enabling methodology is presented in section 3.1. For evaluation of the methodology it was applied in a case study. The case study considers a common service enabling scenario and is presented in section 4. Finally the results and an outlook about future work are summarized.

2 Related Work

Traditional approaches for service-oriented system integration aim at wrapping the system as a black box. This "black box" (or adapter) approach wraps the interfaces of an existing system by using service-oriented technology. This allows the integration of the system as a service (or a collection of them). The advantage of this approach is that only the system interfaces must be analysed and the internals of the system can be ignored. The main disadvantage of the "black box" approach is that it can complicate system maintenance and management. More information about this approach can be found in [4].

Another approach is the complete reverse engineering of existing code (also called "white box" approach) in order to extract business logic for re-implementation in a service-oriented manner. Thus it can solve the special problems of legacy systems like maintenance, management and changeability. The problem is that this approach can be very complex and expensive and it is not always possible to extract the full business logic perfectly.

Zhang and Yang propose an approach for incubating services in legacy systems [5], so that they can be used in a service-oriented environment. The approach is a kind of "grey box" approach, which means that only some valuable components are extracted from the legacy system and provided as self-contained services.

Depending on the situation all the presented approaches are useful, but a surrounding methodology is missing that guides engineerers through the whole migration process. On the one hand such a methodology should provide a step-by-step procedure for service enabling that leads to good services and thus contributes to the overall quality and performance of the process. Additionally it is necessary that the methodology is flexible enough to handle changing requirements.

Some concepts from Zang and Yang's methodology can be reused for building a service enabling methodology, because their approach takes already some service enabling related topics into account, for example domain analysis, service identification and service packaging. However changes in the structure as well as some additional steps are necessary in order to create a successful service enabling methodology.

3 Proposal for a Service Enabling Methodology

This section will introduce the methodology that is proposed for service enabling. Therefore the requirements that must be fulfilled by such a methodology are described first.

The list of requirements that must be fulfilled by an useful service enabling methodology are listed below.

1. It must cover all aspects from process up to integration and system level in order to identify good services.

2. It must provide a step-by-step procedure that leads engineers to the point of a service enabled system.
3. The methodology must contribute to the overall quality and performance of the process by supporting identification of suitable services.

3.1 Methodology Description

After presenting the requirements for the service enabling methodology, the details of the proposed methodology will be discussed now. Figure 1 illustrates the methodology. It shows an iterated procedure for service enabling that consists of four main parts. These parts are discussed in detail below.

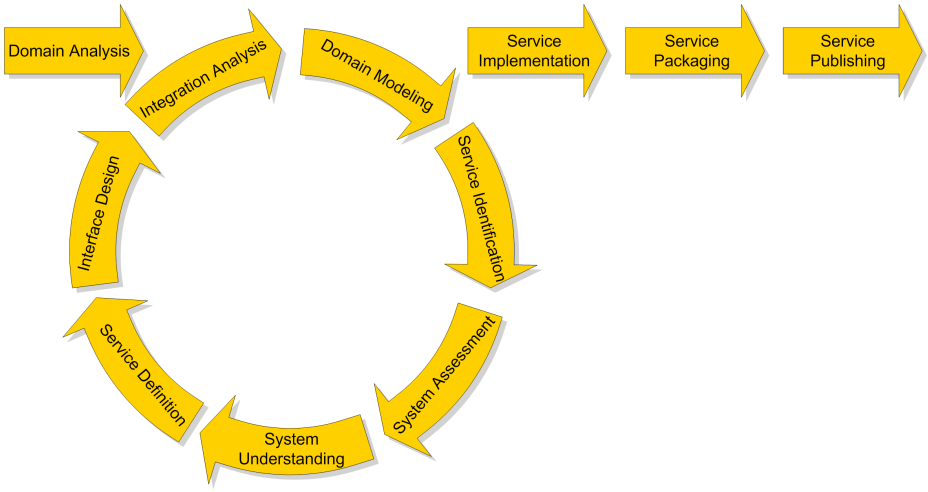


Fig. 1. Service Enabling Methodology

The first part is understanding the processes in our domain and identification of ideal services for that domain. Therefore the methodology starts with a domain analysis of the considered processes. In the domain analysis procedures, concepts and entities of the particular domain and the dependencies between them are detected.

After that the results of the analysis are modeled in the domain modeling step. The results of this modeling are process diagrams (e.g. in the *Business Process modeling Notation* (BPMN, [6]) and diagrams for static structures like ERDs (ERD, [7]) or UML class diagrams (UML, [8]). Which modeling notations are used depends on the domain and experience of the particular project and involved people.

The next step of the methodology is the identification of ideal services that support exactly the needs of the domain. The result of the service identification is a set of functional descriptions of services that are useful to be exposed as

independent services from the process point of view. After that the analysis of the process and domain level are finished for now and we can continue with the system level.

After ideal services for the domain have been found, it is necessary to take a look at the available systems and analyse what services they can provide. The functional description of the identified services can be used for selecting and assessing appropriate systems for implementation of the services. Assessment means that the system is only considered from the functional point of view. The internals of the system are not considered in this step, because there is potentially a huge number of systems that might be useful. Thus unpromising systems are sorted out from the list of candidate systems in the system assessment.

A system can be seen as useful for implementing a service if the following questions can be answered with "yes".

1. Does the system contain some reusable and reliable functionality (or business logic)?
2. Are experts available for the programming language of the system?
3. If there are licence costs, have we already a licence or is it possible to buy a licence for the system?

After selecting promising system candidates for implementing the services it is necessary to analyse the details of the selected systems in the system understanding step. This includes modeling the system, detection of system components and top level functions that can be used for service implementation. Additionally it is necessary to map domain model concepts to system level concepts. This includes analysis of input data, mapping of domain to system classes and so on.

As described in section 2 it might be reasonable to split the system into independent components, depending on the type and structure of the selected system. If there is a need to split the system into independent components, it must be asked if the reused components are fairly maintainable compared to the whole system. If the answer to this question is "no" it will be better to use the "black box" approach instead of extracting some components like in the "grey box" approach.

Extracting and refining components is only necessary if the used system provides diverse functionality that should be splitted for building self-contained services, e.g. if a system provides mail and calendar functionality these two can be splitted into two different services.

After we know the ideal domain services as well as existing physical services, it is now possible to use this information for finding services that are a good trade-off between the ideal and existing services.

The information from the system understanding is used together with the functional descriptions of the ideal services to define the services that should be implemented. These services are something between the ideal services that support exactly the domain requirements and the existing physical services. Afterwards the interfaces for the services are defined. For this purpose the *Web*

Service Description Language (WSDL, [9]) is used in the area of web services. In the interface definition it is necessary to define message style and encoding as well as exceptions that can be produced by the service.

Sometimes it turnes out that the selected system is not qualified for implementing a service, because the comparison between system and domain model shows non-bridgeable differences between the system and domain model. In this case it is necessary to proceed with another system that was selected in the system assessment.

At last service enabling has an additional aspect that is not covered by the original methodology. Because the methodology presented here uses business processes as starting point of service enabling, the integration of the resulting services into the processes needs to be considered. In other words it must be analysed how the services are used later on. This aspect is founded by the question where the input data for the service comes from and how is the output data used.

In the integration analysis incoming and outgoing messages of the services as well as the dependencies between them are analysed. This includes definition of data mappings, e.g. via XSLT transformations or BPEL XPath copy mechanisms.

During the integration analysis it is sometimes realized that additional services are necessary, e.g. for providing data from the external world. This is the point where the iteration comes into play, because if it is realized that additional services are needed, they must be added to the domain model and the iteration starts from scratch. Another argument for the iteration is that the processes can change during a service enabling project. Iterated development processes have been introduced not so long ago, because they support flexibility in terms of changing requirements or caused errors.

After one or more iteration rounds finally all service interfaces are well defined. Now it is possible to implement the services and package them together with according system components. After that the services can be deployed and published.

4 Case Study Results

In this section the case study that was used for evaluation of the methodology is presented. The case study is based on a common service enabling scenario that is described as follows.

In a company each employee collects the number of hours for the projects he worked on. This information is stored in the database of a time tracking system. Every three months a manager is responsible for the creation of a report that summarizes the company efforts and how they have been spent. For this the manager collects the effort information from the time tracking system in an Excel file. This Excel file is sent via e-mail to a consulting company. The consulting company uses an internal system to create some nice looking charts for the report from this information.

Additionally the manager is responsible for creation of a report document from a template and for writing some effort documentation. After receiving the diagrams from the consulting company he assembles the report and publishes it to an internal web portal. After that the process is finished.

Because the company restructures their IT-infrastructure to service-oriented processing, this interaction with the consulting company should be automated by a service for creation of the charts.

4.1 Domain Analysis and Modeling

In the first methodology steps the domain was analysed and modeled. The resulting process was modeled using BPMN as shown in figure 2.

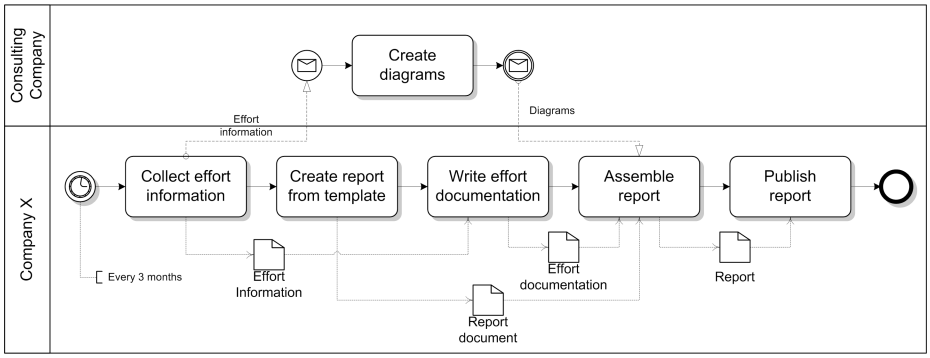


Fig. 2. Report Creation Process

Additionally the entities that are relevant for the process and the relationship between them have been modeled as an ERD[1]. The ERD is shown in figure 3. The diagram contains the company and its environment including employees, customers, projects. Important is the "Work Item" relation that describes the effort spent by an employee for a specific project. These work items are used in the next step for definition of the ideal service.

4.2 Service Identification

In our scenario we want to avoid that the manager must manually collect the effort information in an Excel file and send it via e-mail to the consulting company. Thus an ideal create diagram service will take work items from the time tracking system as input and should create diagrams as output. How this service is integrated with the existing systems is part of the integration analysis.

[1] The ERD style that was used for modeling comes from the Fundamental Modeling Concepts [7].

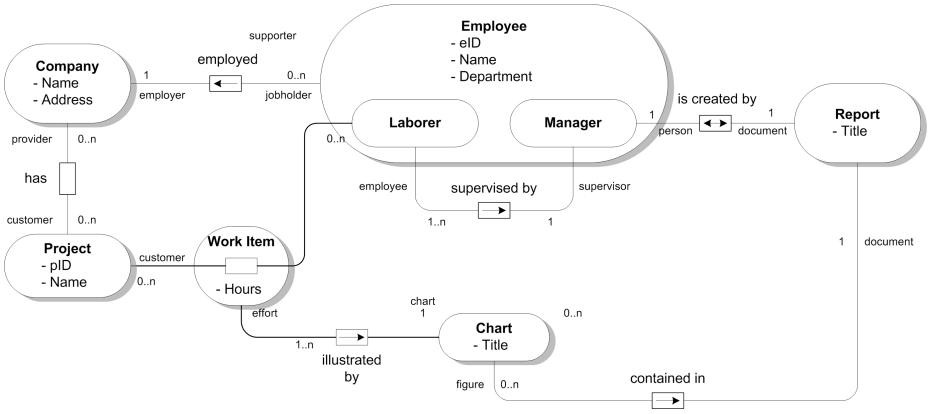


Fig. 3. Domain Entity-Relationship Model

4.3 System Assessment and Understanding

After the identification of ideal services it is possible to select appropriate system candidates for implementing the services. The consulting company uses an self-written application that is based on the framework JFreeChart [10] for creation of diagrams.

The questions from the system assessment can all be answered with "yes". In our case a system is already available that has some reusable and reliable functionality, namely creation of charts and diagrams. Furthermore the consulting company owns this application and a license is of no importance.

After the first assessment the selected system was analysed. Even if there is some documentation available about JFreeChart, it was also modeled for completeness of the case study. The resulting UML class diagram is shown in figure 4.

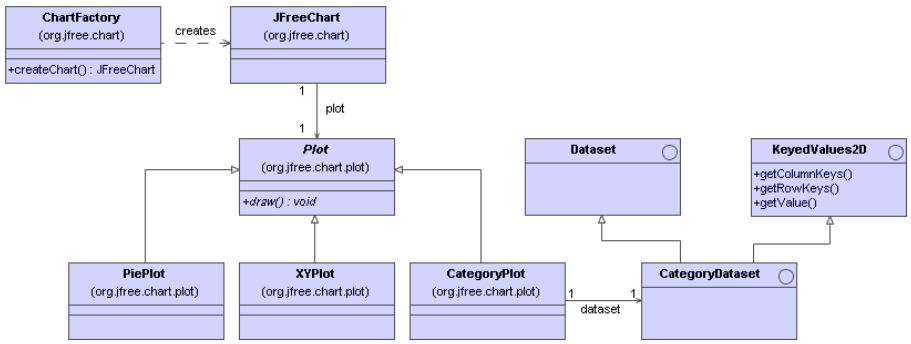


Fig. 4. System Model

Now we can try to map domain concepts to system concepts. In our scenario we can map the *chart* entity of the domain model to the JFreeChart class. Futhermore *projects* are mapped to categories of a CategoryDataset and the employees with according work hours are mapped to key/value pairs of a category. In other words each project (category) contains a set of employee/work hour (key/value) pairs.

If such a mapping is not possible it is necessary to select another system from the list that results from the system assessment. For example it is imaginable that JFreechart only supports the creation of pie charts. If so it will be impossible to create a categorized chart for projects, employees and assigned work hours.

Since we want to integrate the system as a service, we can not use it via traditional user interfaces like GUIs or command line. For this reason we need to identify the top-level functions that can be used for implementing the service. In the case of JFreeChart all needed top-level functions are combined in the ChartFactory class shown in figure 4.

4.4 Service Definition and Interface Design

Now it is possible to define our final service. As mentioned before an ideal service for our domain will take work items as input and should create charts as output. Since work items do not contain all necessary information for creation of charts with the existing system, e.g. title, resolution and so on must be provided. This information was provided by the consulting company before and it is necessary to add this information to the input message of our service.

Therefore the input message of the service should contain chart descriptions instead of work items. This chart description can contain the information of work items in form of categorized key/value datasets as well as additional information that is necessary for creating the charts. This is also a benefit for the consulting company, because a more generic service can be offered to other companies as well.

If we have this definition of our final service, we can now define the interface of the service. An excerpt from the defined WSDL is shown in listing 1.1.

Listing 1.1. Service Interface Description

```
<wsdl:definitions name="ChartService" ...>
  <wsdl:types>
    <xsd:schema
      targetNamespace="http://www.example.org/charts/ChartService" ...>

    <!-- type definitions -->
    <xsd:complexType name="chartDescription">
      <xsd:sequence>
        <xsd:element name="title" type="xsd:string" />
        <xsd:element name="subtitle" type="xsd:string" />
        <xsd:element name="format" type="xsd:string" />
        <xsd:element name="antiAlias" type="xsd:boolean" />
        <xsd:element name="threeD" type="xsd:boolean" />
        <xsd:element name="data" type="tns:data"
          minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="data">
      <xsd:sequence>
        <xsd:element name="key" type="xsd:string" />
        <xsd:element name="value" type="xsd:double" />
        <xsd:element name="category" type="xsd:string"
```

```
            minOccurs="0" maxOccurs="1" />
        </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="createChart">
        <xsd:sequence>
          <xsd:element type="tns:chartDescription" />
        </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="createChartResponse">
        <xsd:sequence>
          <xsd:element name="chart"
            type="xsd:base64Binary" nillable="true"/>
        </xsd:sequence>
      </xsd:complexType>

      <!-- message element definitions -->
      <xsd:element name="createChart" type="tns:createChart"/>
      <xsd:element name="createChartResponse"
        type="tns:createChartResponse"/>

      ...
</wsdl:definitions>
```

4.5 Integration Analysis

The integration analysis is needed for defining how the service should be used later on. If we assume that the effort tracking system provides a service for retrieving work items, then we can combine the services by using a BPEL [11] process engine or a self-written application. In each case it is necessary to define a mapping between the work items and the chart description. In case of BPEL this can be done via XSLT [12] or BPEL XPath copy mechanisms [11].

Additionally it is necessary to provide the needed additional information like chart title or resolution. If we use BPEL for invocation of the service, then we can add a new service to the domain model that requests the manager for providing the additional information. In this case we must enter a new iteration of the methodology in order to analyse how this service can be provided.

4.6 Interface Implementation, Packaging and Publishing

At last the interface for the service must be implemented with an appropriate techology, e.g. Java or C#. For implementation of the service the EJB standard was selected, because JFreeChart is written in Java and EJB allows standardized cross-platform deployment. From EJB 2.1 on web service interfaces are supported as well. After packaging the EJB along with the application libraries, the resulting JAR can be deployed in an application server like JBoss and the service can be registered in an UDDI registry.

The service is now ready for use. A chart that was created with the final service is shown in figure 5.

5 Summary and Outlook

This work has described the need for service enabling of traditional non-service-oriented systems. For this reason a methodology for service enabling was proposed. The methodology provides a step-by-step procedure that helps to find

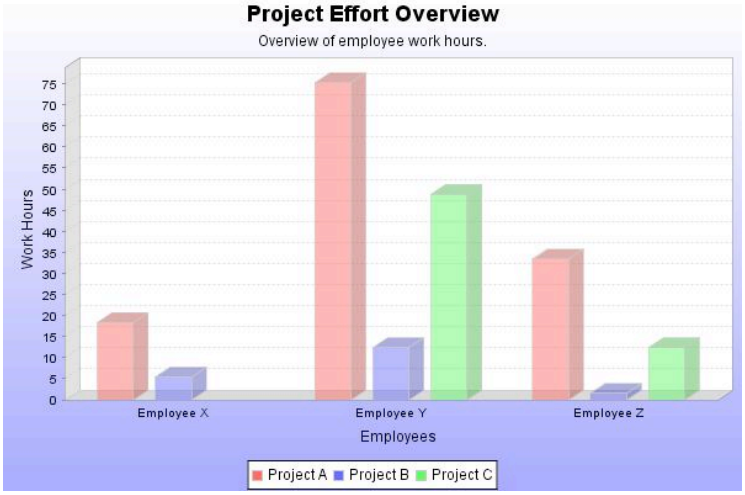


Fig. 5. Resulting chart

good services by negotiation of trade-offs between the ideal services for a domain and the existing physical services.

It was shown that the methodology works in common service enabling scenarios through applying it in a case study. However this shows only that the methodology works in general. For further evaluation it will be necessary to define measurable criteria and compare this approach with others in order to identify benefits and drawbacks of the methodology.

Additionally there are some tasks that are not covered by the methodology until now. These tasks include migration of data when extracting system components as well as monitoring of service execution in order to improve the performance of the process. It is part of future work how these issues can be integrated into the methodology.

References

1. van der Aalst et al.: Business Process Management: A Survey. In: Proceedings of the International Conference on Business Process Management (BPM 2003), Springer Verlag Berlin Heidelberg (2003) pp. 1–12
2. Dostal et al.: Service-orientierte Architekturen mit Web Services. Spektrum, Akademischer Verlag (2005)
3. Mohammad El Ramly, Reiko Heckel: Reverse Engineering Service Models from Legacy Software (2005)
4. Dietmar Kuebler, Wolfgang Eibach: Adapting Legacy Applications as Web Services. http://www-128.ibm.com/developerworks/webservices/library/ws-legacy/ (2002)
5. Zhuopeng Zhang, Hongji Yang: Incubating Services in Legacy Systems for Architectural Migration. In: Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04), apsec (2004) pp. 196–203

6. Business Process Management Initiative: BPMN Specification Releases 1.0. http://www.bpmn.org/Documents/BPMN%20V1-0%20May%203%202004.pdf (2004)
7. Knoepfel et al.: Fundamental Modeling Concepts: Effective Communication of IT Systems. John Wiley and Sons Ltd (2006)
8. Born et al.: Softwareentwicklung mit UML 2. Addison-Wesley, Mnchen (2005)
9. Web Service Description Language. (http://www.w3.org/TR/wsdl.html)
10. JFreeChart. (http://www.jfree.org/jfreechart/)
11. Business Process Execution Language. (http://www-128.ibm.com/developerworks/library/specification/ws-bpel/)
12. XSLT. (http://www.w3.org/TR/1999/REC-xslt-19991116)

Specifying Service Landscapes

Martin Breest

Hasso-Plattner-Institute for IT Systems Engineering at the University of Potsdam,
Prof.-Dr.-Helmert-Str. 2-3, D-14482 Potsdam, Germany
martin.breest@student.hpi.uni-potsdam.de

Abstract. In today's enterprises, failure during business process enactment can lead to lost revenues, cancelled contracts and even sues for caused damages. To find failures quickly, we need to be able to conduct root-cause analyses from business processes to the used enterprise services of a service-oriented architecture (SOA) down to the software and physical elements of the IT infrastructure. In order to not only find failures but to develop and apply strategies to avoid them, we require a model that relates the aspects of a SOA with those of the IT infrastructure. Because no such model exists yet, we propose to add a process model based on the concepts of the Business Process Execution Language (BPEL) [2] to the Common Information Model (CIM) [1].

1 Introduction

In today's severe competition, global-running enterprises have to keep their business processes and services running at any cost. Failure during business process enactment can lead to lost revenues, cancelled contracts or even sues for caused damages. Because business processes in today's enterprises rely on service-oriented architectures (SOA), an effective monitoring of all aspects of a SOA and the underlying IT infrastructure is required to locate failures easily, solve them rapidly and apply solutions quickly. However, to find the root causes of such failures and their impact on other elements of the service landscape, it is not only necessary to monitor each aspect separately but to establish traceability between them. This would also allow us to not only monitor a SOA to react on failures but to simulate failures in different scenarios beforehand in order to develop and apply strategies to avoid them in the future.

In order to trace failures, we require a model in which traceability between the aspects of a SOA and the IT infrastructure is established, and an instance model that contains information about the concrete elements of a particular service landscape and their state. However, in most enterprises, it is often problematic to create such an instance model for a number of reasons. Firstly, the knowledge about the required aspects is often spread across different organizations and stored in various silos. Secondly, a model that allows to represent this information is currently not available. Thus, to conduct the desired simulations, a model with established traceability is required.

Because we want to talk about business processes that rely on a SOA, the core concept of such a model is a *service*. Services provide business functionality on a

coarse-grained level through a well-defined interface. The business functionality is provided or implemented by *software elements*. Software elements are part of software systems that provide access to the implemented business functionality. Software systems are usually deployed on one or more *physical elements*. Physical elements might be connected with each other over network connections. Finally, the coarse-grained business functionality provided by services is orchestrated by *business processes* to fulfil a certain goal.

When we talk about SOAs, we mean web service architectures (WSA) [3] as a certain technical realization. Therefore, with services we mean Web Services that have a service interface described using WSDL [5] and can be accessed using HTTP as transport protocol and SOAP as message format. When we talk about software elements and systems, we want to focus on those that are common in the J2EE-world [7]. Concerning software systems, these are for example J2EE application servers, databases, Business Process Management Systems (BPMS) and applications like SAP/R3 modules. Concerning software elements, these are EJBs, Servlets and J2EE resources. When we talk about business processes, we consequently mean processes specified in the Business Process Execution Language (BPEL) [2]. And with physical elements, we mean machines, switches, routers or bridges.

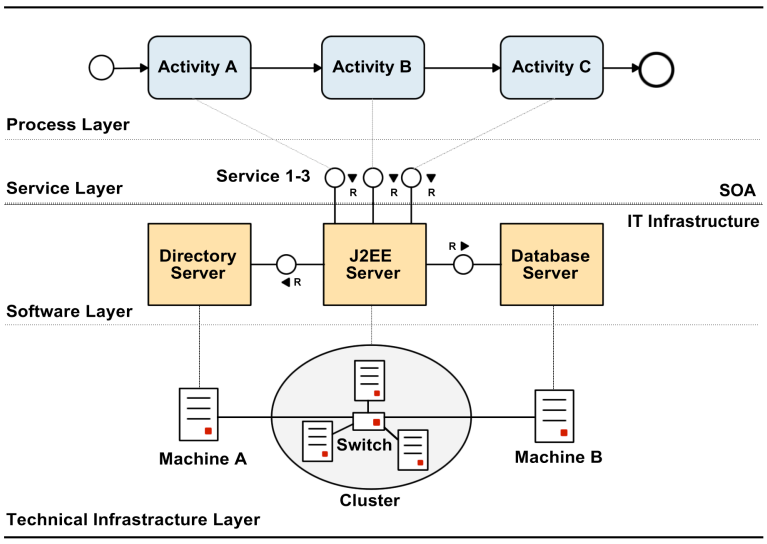


Fig. 1. Overview of the core concepts of the required model, their relationships and the layer they belong to.

The aforementioned core concepts and their basic relationships are illustrated in figure 1. As you can see, the concepts belong to four layers. They are derived from the viewpoints proposed in [4]. The identified layers are:

- a *process layer*, that contains the activities of a process, their execution order and their relationships to the services that provide the used business functionality,
- a *service layer*, that contains the services of the service landscape that are implemented and provided by software elements and systems, and integrated into business processes,
- a *software layer*, that contains the software elements and systems that implement or provide access to business functionality, their relationships among each other and their deployment on certain physical elements,
- a *technical infrastructure layer*, that contains the physical elements and the network connections between them.

In this paper, we examine how the Common Information Model (CIM) that covers the three lower layers can be enhanced by concepts of the BPEL model that covers aspects of the two upper layers to provide traceability across all four layers. Having a model that relates all aspects allows us to answer the following key questions:

- What is the root cause for a failure during process enactment?
- Which processes are affected when a physical or software element breaks down?

Because, we are concerned about simulation, we want to look for a model that allows to answer the second question.

This paper is organized in the following manner. In section 2, we give an introductory example of a service landscape and describe two scenarios for a better understanding. We then look at the BPEL model for describing business process and CIM for describing IT infrastructures. We also show that these models are not connected yet and that we have a problem with traceability because of that. In section 3, we propose to add a process model based on the concepts of BPEL to CIM to create one model with established traceability across all four layers. We also show the applicability of the approach on the two described example scenarios. In section 4, we give a conclusion of the achieved results and a summary of the next steps towards simulation.

In this paper, we use the Business Process Modelling Notation (BPMN) [8] to illustrate the elements of the process layer, the block diagram notation of the Fundamental Modelling Concepts (FMC) [9] to illustrate the elements of the software layer, circles to illustrate the services of the service layer and, finally, an own notation to illustrate the technical infrastructure layer consisting of boxes representing physical elements and lines representing the network connections between them. We also use the Unified Modelling Language (UML) [10], later, when we add the process model to CIM.

2 Existing Models and the Problem of Traceability

In order to simulate failure scenarios and to be able to reason about the impact of an error in one element to the other elements of a service landscape, we need:

- a *meta model* - that defines all concepts to express classes, their properties and their relationships
- a *model* - that contains all classes to describe the concrete elements of the four layers and their relationships
- *instance models* - that contains a representation of all actually existing elements that can be found in the four layers of a service landscape and their state

Having a model and an instance model for a certain simulation, we would be able to reason about failures and their impact. Because there exists no such model yet, we want to look at BPEL for describing business process and CIM for describing IT infrastructures in this section. However, to get a better understanding for what we require the model and the instance models, we, first, want to have a look at an example service landscape.

2.1 An Introductory Example

Figure 2 illustrates various elements that might really exist in a service landscape of an enterprise, like the machine `M1` or the database server `OracleDB`. All these elements can be described in an instance model, which then can be used together with the model to simulate a failure scenario.

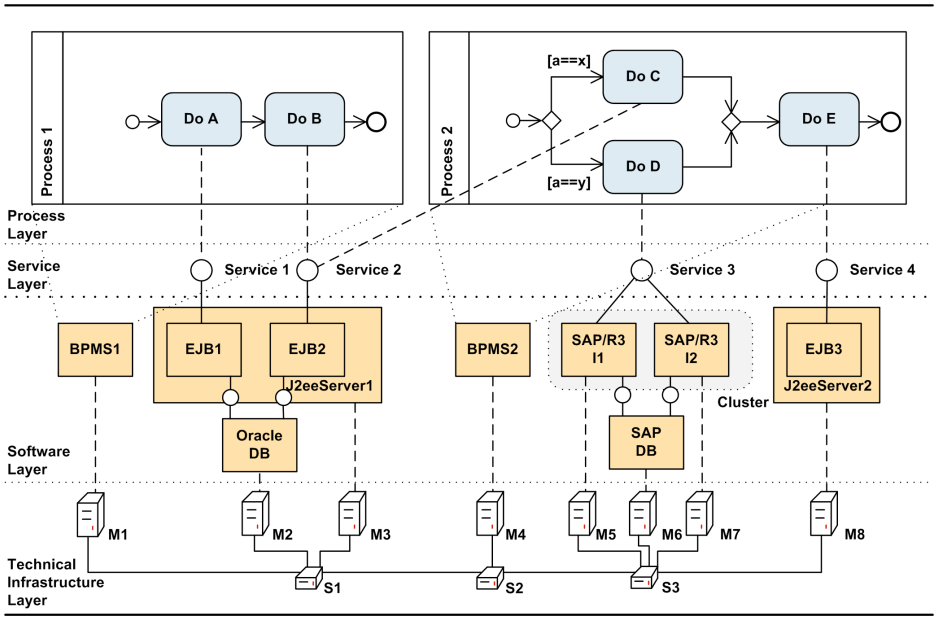


Fig. 2. An example service landscape of an enterprise.

As you can see, there are two process definitions `Process 1` and `Process 2` in the service landscape that can be enacted by two different BPMS `BPMS1` and `BPMS2`. `Process 1` consists of two activities `Do A` and `Do B`, and `Process 2` of three activities `Do C`, `Do D` and `Do E`. All these activities define an invocation of one of the services `Service 1-4`. One can also see that activity `Do B` and `Do C` both depend on the same service.

The services are provided by different software elements and systems. `Service 1` and `Service 2` are provided by two EJBs `EJB1` and `EJB2` that run on a J2EE-Server `J2eeServer1`. Both EJBs depends on the availability of the `Oracle DB` database. `Service 3` is provided by two clustered `SAP/R3` instances. Both depend on the `SAP DB` database. Finally, `Service 4` is provided by `EJB3` that runs on another J2EE-Server `J2eeServer2`.

The software systems are deployed on 8 machines `M1-8` and connected by 3 switches `S1-3`.

Having described the elements of the instance model, we can now develop the two following scenarios for failure simulation to point out how a model with established traceability can help us to identify the impact of the failure.

Scenario 1: In the first scenario, we want to know which impact an error of element `SAP/R3 I1` would have on our business processes. We can see that this element provides `Service 3` which is only used by activity `Do D` of `Process 2`. One can also see, that `Service 3` is provided by a clustered `SAP/R3` system and that an error of the first instance of this cluster would have no impact on the enactment of `Process 2`.

Scenario 2: In the second scenario, we want to know which impact an error of the whole `SAP/R3` cluser has on our business processes. We can see that because `Service 3` is provided by the cluster only, an error would have an impact on the enactment of `Process 2`.

Having an understanding of the problem to solve, we now want to examine the existing models and the aforementioned problem of traceability.

2.2 Describing Business Processes with BPEL

The BPEL model allows to describe business processes in two ways: as executable processes and as abstract processes. An executable business process, on the one hand, describes the actual behaviour of a participant in a business interaction. An abstract business process, on the other hand, contains the descriptions for business protocols, which means that it describes the message exchange of the business party involved in the business protocol without revealing the internal behaviour.

An enacted BPEL process can be used as Web Service and can also orchestrate a number of other Web Services. The abstract process part is therefore obviously used to create the service operations for the WSDL service description that enables a BPEL process to be used as Web Service in an interaction,

whereas the executable process part defines the orchestration of number of Web Services using their service operations. Because we are concerned about orchestration, we want to focus on the basic elements that are required to define the executable process part.

A BPEL process can be defined using the `Process` class. Each process can have exactly on initial activity. An activity in BPEL can be defined using a subclass of the class `Activity`. BPEL provides various classes to defines basic and structured activities of a process. Whereas basic activities define a certain execution step, structured activities define the execution order of a number of activities. Because we only want to represent basic processes in our simulations, we want to focus on the three activity classes `Invoke`, `Flow` and `Sequence` of BPEL.

With an `Invoke` activity, we can define the invokation of a particular Web Service operation. We therefore reference a particular port and operation defined in the abstract part of a WSDL service description and a certain partner link that is also defined in the BPEL process. The partner link references a certain partner link type that is usually defined in a separate BPEL partner definition. This partner link type, finally, references the port type and the address defined in the concrete part of the WSDL service description via a WS-Addressing [6] endpoint reference.

With `Flow` and `Sequence` activities, we can define the execution order of a number of activities. With a `Sequence` activity, we can define the sequential execution of a number of activities. Using a `Flow` activity, we can define the concurrent execution and synchronization of a number of activities. Concurrent execution means that the execution path can be split into multiple execution paths that are executed in parallel. Synchronization means that a number of execution paths that run in parallel are synchronized to one single execution path. We can express the synchronization dependencies between activities using the `Link` class. A `Link` represents a synchronization point that can be used to link two activities. Therefore, for each `Activitiy` we can define a number of incoming and outgoing links. To synchronize a number of incoming links we can also define a `joinCondition`, that defines the condition under which the activity can be enacted. We can also define `transitionConditions` for outgoing links to define which path(es) should be taken after an activity has been enacted.

We can see that BPEL provides basic constructs to describe business processes, although we did not present all of them in detail. We have also shown how Web Services can be integrated into BPEL processes. However, because a WSDL service description does not reference the software element that actually implements the services, the link between service and service providing software element is missing.

2.3 Describing IT Infrastructures with CIM

The Common Information Model (CIM) developed by the Distributed Management Task Force (DMTF) is an approach to the management of different aspects of IT infrastructures, such as systems, networks and users, that is based on the

object-oriented paradigm. CIM provides a management model to establish a common conceptual framework for the description of the managed environment.

The management model is based on a mature meta model that defines concepts like `Class`, `Property`, `Qualifier` to be able to classify managed elements and concepts like `Association` and `Reference` to describe relationships and dependencies between them. It consists of:

- a *core model* - that contains a basic set of classes that are applicable to all management domains
- *common models* - that contain classes common to particular management domains, such as network, operating or database systems
- *extension schemas* - that represent technology-specific extensions of the common models, such as classes for the description of the windows operating system.

The core model establishes a basic classification of the elements and associations of the managed environment. The base class of the class hierarchy is `ManagedElement` which is further subclassed to `ManagedSystemElement`, `Configuration`, `Setting`, `StatisticalInformation` and others. The `Managed SystemElement` is further subclassed to `LogicalElement` and `PhysicalElement`. Whereas a `PhysicalElement` represents any component of a system that has a physical identity, a `LogicalElement` represents systems themselves, system components, system capabilities, software and services. The difference between `ManagedElement` and `ManagedSystemElement` is therefore that the later one also has a state like 'OK' or 'Starting' that indicates the operational status of the represented element. A `Configuration` aggregates `Settings` that define certain parameters to be applied to one or more `ManagedSystemElements` and therefore defines a certain behaviour or functional state of such an element. The class `StatisticalInformation` is the base class for any kind of statistics for a managed element.

The core model provides two different association types, `Component` and `Dependency` association, that can be established between `ManagedElements` resp. `ManagedSystemElements` and all subclasses. Whereas a `Component` association establishes a 'part of' relationship between `ManagedSystemElements`, a `Dependency` association describes a functional dependency or existence dependency between two `ManagedElements`. An association in CIM is represented by a class that has two or more references to the associated classes.

There are three other classes of the core model, that are important for this work: `System`, `Service` and `ServiceAccessPoint (SAP)`. A `System` represents individual entities that can be uniquely identified and are more than the sum of their parts. A `System` is therefore a host for `Services`, `SAPs` and `SoftwareElements` which are all subclasses of `ManagedSystemElement`. A `Service` describes a certain functionality provided by a `System` (e.g. the capability to validate a credit card). A `SAP` defines the way the provided functionality can be accessed (e.g. via a certain URI). The functionality is implemented by one ore more `SoftwareElements`. A `Service` can be composed out of other `Services` and can depend on the functionality of other `Services` of the `System`. A `Service`

might also depend on `SAPs` that provide access to `Services` of other `Systems`. All the mentioned associations are represented by subclasses of the `Component` and `Dependency` association.

The power of CIM lies in the ability to define an arbitrary number of relationships and dependencies between `Managed Elements` without changing the referenced classes. With the described classes of the core model and the subclasses of the various common models it is possible to describe a number of different systems ranging from database systems to network systems. However, we can state that CIM does not provide a common model for process management.

2.4 Problems Concerning Traceability

When we look at the two presented models, we can see that, on the one hand, BPEL and WSDL allow to describe the business processes and enterprise services of a SOA, and that traceability between both aspects is established. However, a WSDL description of a service does, besides an URI by which the service can be accessed, not define which software element actually provides the service. Therefore, BPEL and WSDL cover the aspects of the process and service layer but not of the software layer.

CIM, on the other hand, allows to define all logical elements and physical elements of the IT infrastructure. We also know that in CIM a SAP can describe an URI that can be used to access the provided service. Therefore CIM covers aspects of the service, software and technical infrastructure layer and establishes traceability between them; but it does not cover aspects of the process layer. Thus, to establish traceability across all four layers, we obviously have to somehow connect BPEL and CIM.

3 Connecting the Models to Establish Traceability

When we connect the models, we want to use the CIM core model as base model and use the concepts of BPEL to add a common process model to CIM. We use the CIM core model as base because, as aforementioned, it is based on a mature meta model and allows to define Dependency and Component associations between arbitrary classes. This perfectly fits our requirements for a model for failure simulation. Because only through the definition of dependencies we are able to analyze the impact of a failure of one element to the other elements described in an instance model.

Why can BPEL and CIM be connected? We can connect both models because the service definition used in BPEL and WSA is basically the same as in CIM. WSA defines a service as "an abstract resource that represents a capability of performing tasks that represent a coherent functionality". This definition fits with the service definition used in CIM, in which a service provides some (technical) functionality that is implemented by a software element and

can be accessed by a SAP. We have already mentioned that a SAP can be used to describe an URI, which basically defines the access to a certain resource.

How can BPEL and CIM be connected? Because in the CIM core model a service can be accessed via a SAP, an invoke activity in our new process model relies directly on one SAP. It relies on only one SAP, because we do not want to consider situations where multiple SAPs provide access to basically the same service that is only implemented differently (e.g. service bus). The SAP that is used is actually the one defined in a WSDL service description by an address, referenced through a WS-Addressing endpoint reference in a BPEL partner definition and used by an invoke activity of a BPEL process definition through referencing a certain partner link.

3.1 Formalizing the Process Model

Figure 3 illustrates the formalization of the process model as a UML class diagram. The process model is build around the two classes `BPMSystem` and `Process`.

A `BPMSystem` represents a `System` (BPMS) that is able to enact processes based on the known process definitions. However, process definitions can be known and used for enactment by multiple BPMSs. This relationship is described by the `BPMSystemProcess` dependency.

A `Process` represents a BPEL process definition. The `Process` class and the classes used to describe the elements of a process are all subclasses of `ManagedElement` because they contain basic management information. To describe a BPEL process for simulation purposes, we decided to take a minimized set of classes from the BPEL model that can be extended by additional classes when necessary. These classes are the abstract class `Activity`, the class `Invoke` for defining service invocations, the class `Flow` to define concurrent as well as sequential activity executions and the class `Link` to be able to define synchronization points.

A `Process` has exactly one initial `Activity` in the process model, which is expressed by the `ProcessActivity` dependency. An `Activity` can either be a `Flow` activity or an `Invoke` activity. A `Flow` activity contains a number of `Activities` and `Links`. Both dependencies are described by the classes `FlowActivities` and `FlowLinks`. The activities are linked together using the `OutgoingLink` and `IncomingLink` dependency. Both dependencies represent the link concept used in BPEL. An `Invoke` activity is connected to exactly one `SAP`. The `InvokeSAP` class describes this dependency. The dependency establishes the desired traceability between the process and the service layer. A `SAP` actually provides access to the functionality of exactly one `Service`.

The introduced dependencies bascially state that if an error in a `Service` occurs, the `SAP` that provides access to the `Service`, all `Invoke` activities that use the `SAP` to access the provided functionality, the successor `Activities` and the `Process` itself are affected.

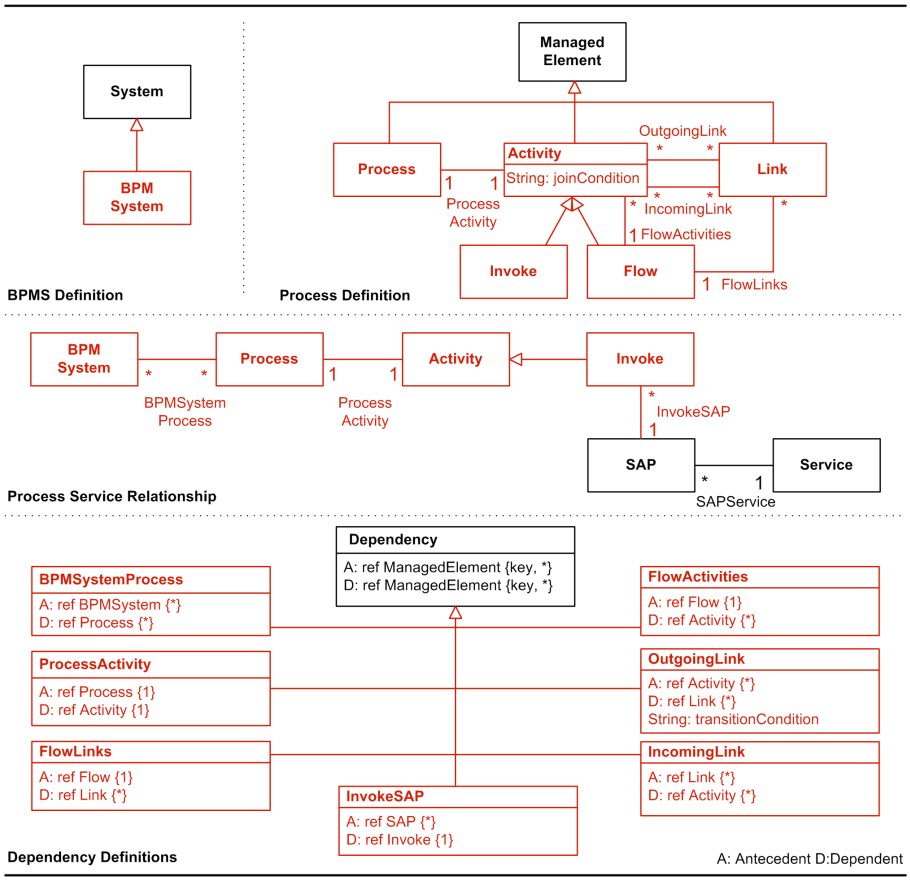


Fig. 3. Classes and dependencies of the process model added to CIM. The classes drawn with red color are part of the new process model.

Thus, having the extended model and an instance model that describes a certain service landscape, we are now able to analyze the impact of an error across all four layers.

3.2 Applying the Formalization to the Example Scenarios

The application of the new process model to the two defined scenarios is illustrated in figure 4. The illustrated object diagram does not contain each detail of the created instance model but is sufficient to demonstrate the core concept.

In scenario 1, we have defined that the first instance of the SAP/R3 cluster has an error. We therefore create two instances `SAP/R3 I1` and `SAP/R3 I2` for the elements of the cluster. The service is provided via one single SAP, which is a web service modelled as instance `SAP/R3 WS`. This SAP is used by the invoke activity `Do D` of `Process 2`, which are both modelled as instances. Having this part of

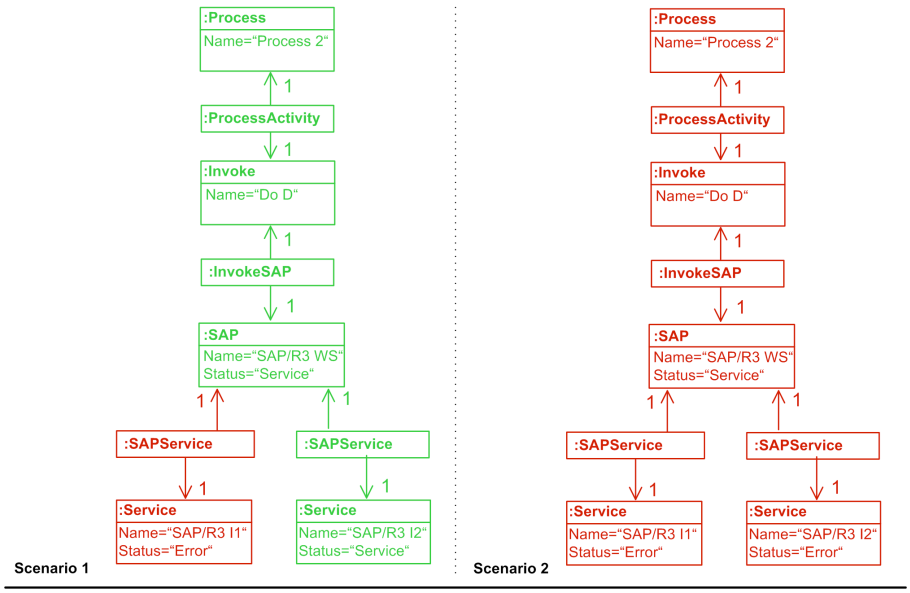


Fig. 4. Instance models of the two scenarios described in the introductory example. Elements that are drawn with red color indicate an error or the impact of an error.

the instance model and the correspoding model, an algorithm can determine that an error of the first SAP/R3 instance will have no impact to the business process because the other instance of the cluster is still providing the service.

However, in scenario 2, we have defined that both instances `SAP/R3 I1` and `SAP/R3 I2` of the cluster have an error. Therefore, an algorithm would be able to determine that the error of both instances affects the enactment of process instances of `Process 2` because the required service can no longer provided.

3.3 Evaluating the Results

With the application of the extended model to the two described scenarios, we were able to say whether a certain business process is affected by an error in the IT infrastructure or not.

What does affected actually mean? Affected means, that it depends on the execution path taken during process enactment, whether a certain process instance that is enacted based on a process definition is affected by an error or not. Thus, to determine the magnitude of an error to the business processes of an enterprise, it is necessary to know how important the affected business processes are, for example if they are primary business processes or supporting processes. It is also necessary to now by which percentage certain execution paths in a process are taken. Only having this statistical information, one can say which

impact an error in the IT infrastructure really has to the business processes of an enterprise. This also defines which strategies have to be developed and applied to reduce the impact of such an error in the future.

4 Conclusion

In this paper, we described how CIM can be enhanced by a process model to establish traceability between a SOA and the IT Infrastructure. We achieved this by adding a number of concepts from BPEL as classes and associations to CIM. We have also shown that the classes can be used to create instance models and that the established traceability allows us to analyze the impact of a failure to different elements of a service landscape.

However, having a model with established traceability between all aspects is not sufficient to conduct the desired simulations. Therefore, a number of further research steps have to be done.

Firstly, with the presented approach, we only scratched the surface of CIM. To create meaningful instance models that allow to represent all elements of a real service landscape, we certainly have to add new common models and extension schemas to CIM.

Secondly, we left open where the data for the creation of concrete instance models comes from. For simple simulations, we could create setup files that contain the required data about the elements of a particular service landscape by hand. However, it would be more appropriate to leverage the data stores spread across the organizations of an enterprise to gather the desired information automatically. The advantage using CIM is thereby that various management enabled systems, such as J2EE servers, database servers and storage systems, and physical elements, such as routers, switches and bridges, are able to provide management information in the CIM format.

Finally, we did not mention that we require special algorithms that are able to analyze the impact of an error using the extended CIM model and a particular instance model. These algorithms are actually the complex part when using CIM, because they have to be able to traverse and evaluate the dependencies and relationships between the various elements of an instance model and across the four layers of the model.

Thus, to really be able to conduct the desired simulation, a lot of research is still to be done. It is questionable whether a self-defined model that supports the concepts and relationships for the desired domain would make it easier at least to conduct the desired simulation.

However, CIM is adopted by the industry. Microsoft uses it to manage some of there software systems and IBM to manage there storage systems for example. This definitively shows the applicability of CIM.

References

1. DMTF: CIM: Common Information Model Schema Version 2.11. http://www.dmtf.org/standards/cim/cim_schema_v211, December 2005.
2. BEA Systems, IBM, Microsoft, SAP AG and Siebel Systems: BPEL4WS: Business Process Execution Language for Web Services 1.1. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/, May 2003.
3. W3C: WSA: Web Services Architecture. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/, February 2004.
4. ISO/IEC: ITU-T X.901 ISO/IEC 10746-1 Open Distributed Processing Reference Model Part 1. Draft International Standard (DIS) output from the editing meeting in Helsinki (Finnland), May 1995.
5. W3C: WSDL: Web Service Definition Language 1.1. http://www.w3.org/TR/wsdl, March 2001.
6. W3C: WS-Addressing: Web Services Addressing. http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/, August 2004.
7. Sun Microsystems: J2EE: Java 2 Platform, Enterprise Edition. http://java.sun.com/javaee/index.jsp.
8. Business Process Management Initiative: BPMN: Business Process Modelling Notation 1.0. http://www.bpmn.org/Documents/BPMN V1-0 May 3 2004.pdf, May 2004.
9. FMC Consortium: FMC: Fundamental Modelling Concepts. http://www.f-m-c.org, 2005.
10. G. Booch, J. Rumbaugh, I. Jacobson. The Unified Modelling Language User Guide. Addison-Wesley. May 2005.