Orchestration Driven by Formal Specification

Charif Mahmoudi
Logics, Algorithm, Complexity Laboratory
LACL, Paris 12 University
Creteil, France
charif.mahmoudi@sfr.fr

Fabrice Mourlin
Logics, Algorithm, Complexity Laboratory
LACL, Paris 12 University
Creteil, France
fabrice.mourlin@wanadoo.fr

***Abstract*—Mobile agent software provides a programming paradigm which allows reconfiguration during runtime. Because code migration is a basic concept, software architecture becomes more important. Classically, the lifecycle of distributed application starts with specification description. Several facets have to be specified: agent behavior, message exchange, service composition, but also architecture. This description has also two levels: software and hardware. We use formal specifications because our objective is to define properties about our application. Also, process algebra, like Pi-Calculus, is a formal language, which allows us to provide a formal description of architecture. We can then combine agent behavior and reason to define minimal constraint set of future runtime context. Our work provides a process from formal specification of distributed applications to a skeleton of BPEL script.**

Keywords-mobile agent; architecture specification; service composition.

I. INTRODUCTION

Mobile agent application is a kind of distributed application where software can react with its environment and react to external events. Also, this is particularly useful in case of unstable runtime context or when architecture changes during execution. For instance, grid computing needs large set of computing resources. But, if a resource is missing or fails, the whole computation has to continue until its end. In that context, the result is more important than the performance, also mobile agents are able to move computation to a node where computing resources is free [1]. Without mobile agent, it is not possible to adapt a distributed application to its runtime context because placement is defined at load time. This limit is suppressed with mobility.

Mobile agents are useful in other domain such as software administration or code instrumentation. Software administrator needs to deploy new distributed applications with adapted configuration for security, underlying services, etc. A first solution is to replicate a static image from one node of the network to the others, but the strategy becomes complex when the nodes are not similar. If location involves a specific behavior then mobile agent is a solution. It can adapt its mission to the precise location where it incomes. This can mean select specific permissions depending to a resource location, or choose between several persistence services, etc. [2].

Code instrumentation is another domain where context adaptation is essential. Software instruments can observe runtime properties such as time measure of methods, memory allocation of data structure or state of threads into a thread group. If the analysis is done after an execution, a classical approach can be applied, but if actions have to be done depending on features which are observed then only mobile agents can react and adapt their actions to a specific context [3] [4]. For instance, several threads are blocked because there is a gridlock. Also, a mobile agent can change state of one of the threads to force a specific execution.

We have presented the role of one agent into a distributed application but these examples are useful for understanding the concept of software adaptation based on code migration. Into a case study, there are a large number of mobile agents and all have a common objective, for instance data collection for a performance analysis. Coordination between agents is crucial to insure that all contributions will be used in a suitable manner. This means writing coordination specification. It plays the role of master description where each agent is a piece of software like a rugby player into his team. The whole objective is to win a match, but depending on his role into the group, his own behavior will be to adapt his actions to the context and his partners.

Our experience into software specification was about use of formal language like CCS [5] or Unity [6]. Agent migration needs a higher order language and Pi-Calculus possesses such kind of construction. Also, we used this formal language for writing our formal specifications. Pi-Calculus [7] has operational semantics, which allows us to evaluate terms and transform our specifications into other representations useful for reasoning. In this document, we present how we write coordination description of mobile agent group or agency. Then, we explain how this specification can be used to provide a more executable representation. Finally, we propose an approach to specify architecture and a way to exploit it by an agency. By the end, we sum up through an example that illustrates main concepts of agent migration with message definition.

II. COORDINATION SPECIFICATION

Coordination can be considered as a road book for an agency or group of mobile agents. It contains start state and a final state and between them a succession of steps. A step is realized by a mobile agent. Also, this action step is defined with a location where it has to be done, initial information

 ISBN: 978-1-61208-184-7

for the launch and eventually final information for observing results. This means that only one description is not enough but several descriptions are built concurrently.

Two specification approaches are observed. First, a top down specification approach needs to build all descriptions into a coherent manner. This starts with step definition; for instance, the objective to achieve and then the migration (from where to where); finally, the data format of the input message is defined. These descriptions can be completed by some more details about local resources and output message. This formal language has a syntax which allows designer to express mobility as term. Channels are used to exchange not only data but also agents which are specified as first order term. First, we present quickly higher order Pi-Calculus language.

A. Formal specification language

The descriptive ability that Pi-Calculus offers, emerges from the concept of naming, where communication links, known as channels, are referenced using a naming convention. Hence, mobility arises by having processes communicating the channel names. Some extensions are added by R. Milner himself to allow specification writers with higher order term [8]. Then, agent migration can be expressed through a communication of a first order term [9].

The Pi-Calculus notation (Fig. 1) models distributed agent into a system, which can perform input or output actions through channels, thus allowing the agents to communicate. The message which is sent from one agent to the other is a name, which gives a reference to a channel or a first order term which gives a reference to a local mobile agent.

P, Q	$::=$	0	nil
		$P \mid\mid Q$	parallel composition of P and Q
		$\overline{c}\langle v \rangle.P$	output v on channel c and resume as P
		$c(x).P$	input from channel c
		$(\boldsymbol{\nu} x)P$	new channel name creation
		$!P$	replication

Figure 1. Syntax of Pi-Calculus language.

When a term is received by an agent host, term unification algorithm is applied to propagate names into agent host definition. Operational semantics [9] is useful to build evaluation tree of the agent host term. As an example, we provide a specification of SLP protocol. The Service Location Protocol (SLP) is an Internet Engineering Task Force (IETF) standard track protocol [10] that provides a framework to allow networking applications to discover the existence, location, and configuration of networked services in enterprise networks.

SLP can eliminate the need for the user to know the technical features of network hosts. With the SLP, the user needs only to know the description of the service he is interested in. Based on this description, SLP is then able to return the URL of the desired service. SLP is a language independent protocol. Thus, the protocol specification can be implemented in any language. The SLP infrastructure consists of three types of agents:

1. *UserAgent (UA)* is a software entity that is looking for the location of one or more services; its role is client,
2. *ServiceAgent (SA)* is a software entity that provides the location of one or more services; its role is mobile agent,
3. *DirectoryAgent (DA)* is a software entity that acts as a centralized repository for service location information; its role is registry.

$$\mathbf{System} = \nu(SrvRqst, SrvRply, SrvReg, SrvUnReg, SrvAck)$$

$$UA(SrvRqst, SrvRply)$$

$$|SA(SrvReg, SrvUnReg, SrvAck)$$

$$|DA(SrvReg, SrvUnReg, SrvRqst, SrvAck)$$

Figure 2. Main term of SLP specification.

The subterms UA, SA and *DA* (Fig. 2) are detailed into annex. This grammar is useful for writing specification by hand but is quite complex to use into a workflow system also, we have translated this grammar into an XML schema. Also, this allows us to write specification in a more rigorous manner. Our XML schema stresses the structure of an agent based on the composition operators: sum, parallel, match, restriction, etc. A higher order Pi-Calculus specification becomes a well formatted XML description, which can be transformed into an object easily. It is the pilot of an activity of mobile agents.

B. Coordination of an agency

In the previous example, all components are independent and each has its own behavior. But, the problem is to describe relation between these behaviors. Coordination of software component is not a new challenge. Solutions have been already given by web project architects. Reo project forms a paradigm for composition of software components based on the notion of mobile channels [15]. This project defined its own coordination language which is a channel-based exogenous coordination model. The specification writer defines complex coordinators, called connectors, which are built out of simpler ones [16]. Of course, the Reo coordination language provides, pleasant features such that: loose coupling among components and services or support for distribution and mobility of heterogeneous components or compositional construction. But this language is not become a standard. Also, it is not easy to inter operate with other coordination model. But, Reo language stresses which are the key concepts into coordination. First, a composition of agents has two kinds of observation [14]. On one side, an external observer is not able to distinguish the structure of the composition. On the other side, an internal observer can follow the precise evaluation of the composition. Secondly, the better coupling is asynchronous and exchanges are considered as message passing [13].

We considered these requirements to select a language for defining coordination of agents. An obvious solution could be to declare a master agent which contains the scheduling of coordination. But this approach has drawbacks. If the description is inside an agent, a new

 ISBN: 978-1-61208-184-7

coordination cost becomes another development and there is no standardization of the approach. When an agent pilots coordination programmatically, the state of the evaluation is difficult to observe. Also, the definition of coordination should be external and then its interpretation can be done by an agent or another engine.

Because of our past experience on Web service design, we studied several existing coordination languages such as WSCI (Web Services Choreography Interface), BPML (Business Process Modeling Language), WSCL (Web Services Conversation Language), BPEL4WS (Business Process Execution Language for Web services) [12].

WSCI is a description language based on XML, which aims at describing the messages exchanged between agents into coordination. BPML is a high level language which is used to describe business process as a sequence of simple, complex activities including the interaction between participants in order to achieve a common objective. WSCL language is used to describe the business logic or public sub processes based on the definition of a web service. BPEL [11] language (Business Process Execution Language) replaces previous specifications of Microsoft XLANG and WSFL (Web Services Flow Language) from IBM. BPEL is used to model two types of processes

- Abstract process: specifies the messages exchanged between the partners, without specifying the internal behavior of each.
- Executable process: specifies the execution order of activities constituting the process, the partners involved in the process, the messages exchanged between the partners, and processing of errors and exceptions specifying the behavior in case of errors or of exceptions.

An external observer can consider a process as a mobile agent if this agent has a formal declaration. In the context of BPEL language, this description is provided as WSDL format.

BPEL is a language for describing orchestration of Web services. But inside an orchestration services are composed and often a transaction is created for the execution. We consider that BPEL specification can describe the execution order between a numbers of agents constituting the process definition, the partners involved in the process, the messages exchanged between these partners. Next, we need to define a mapping between higher order Pi-Calculus and BPEL language. It means a transformation from a formal language into a more operational language.

C. From HOPi calculus to BPEL

Some works already exist about mapping between Pi-Calculus and BPEL. Faisal Abouzaid uses a first version of Pi-Calculus based on monadic expressions and first order term definitions [18], [19]. We extend this work and adapt it to our framework of mobile agent system. Two main features are taken into account: polyadic expression and higher order term which are used for communication description. Because BPEL language is verbose and contains a lot of technical details, we have developed a strategy to generate BPEL skeleton. The choice of BPEL language involves that each component can be considered through its WSDL description. This one contains several parts such as types, messages or port type, etc. Also, we consider Pi-Calculus specification as an input source for filling not only BPEL skeleton but also WSDL declaration.

Because our input specifications are written into XML format, each step of our strategy is an elementary transformation belonging to a more global chain called BPEL generation. We use the structure of specifications to enrich all our artifacts (WSDL and BPEL).

III. TRANSFORMATION INTO BPEL SCRIPT

As we presented in Section 2.A, a Pi-Calculus specification contains a main term, called System into Figure 2. This pi-calculus process is composed by parallel and synchronizing actions. So, the underlying rules of the mapping are correspondences between Pi-Calculus terms and BPEL blocks. Identifiers are essential to propagate data and refactoring is necessary as a pre statement for preparing future generation. The first part of our transformation chain is described as follows into figure 3:

Figure 3. Pre statement from a specification to an enriched description.

A. Structural transformation

We use a top down approach; this means that we exploit the structure of a higher order Pi-Calculus specification. First, we consider the main term as a main BPEL sequence. The definitions of each sub term are considered first as partners of the script.

Process calls can contain typed arguments. Abstract data types can be specified with Pi-Calculus language as a process. Such a definition is converted into types in the WSDL description. Of course, this declaration is included first into the WSDL flow where agent is declared. But, data types are shared between several process declarations, also, it is useful to create XML schema which contains complex type. Then, XML schemas are imported into WSDL definitions of associated agent. We build a dependency graph of the definitions (data and behavior). The edges represent definition importation and communication relation (I/O). This relation is used to enrich first XML representations with annotations. These ones are about oriented actions such as input message, output message, call of agent, etc.

B. Annoted XML flow

Each transformation is built with XSL-T language. This means that we use a standard language dedicated to graph transformation. Because each XML flow can be considered as a graph, we can use a set of rules for the basic construction of Higher Order Pi-Calculus language and a rule engine to select closest rules. A rule is a template based on specific patterns of XML from the input source.

Then, an XML output is computed based on the input. Our schema allows us to check the structure of agents before and to map Pi-Calculus structure on to BPEL blocks. For instance, a sequence of actions is mapped as a BPEL sequence. More complex is the transformation of data

 ISBN: 978-1-61208-184-7

exchange. An input of data means a message output; this involves not only a type definition for the message, but also a call to an operation of another agent.

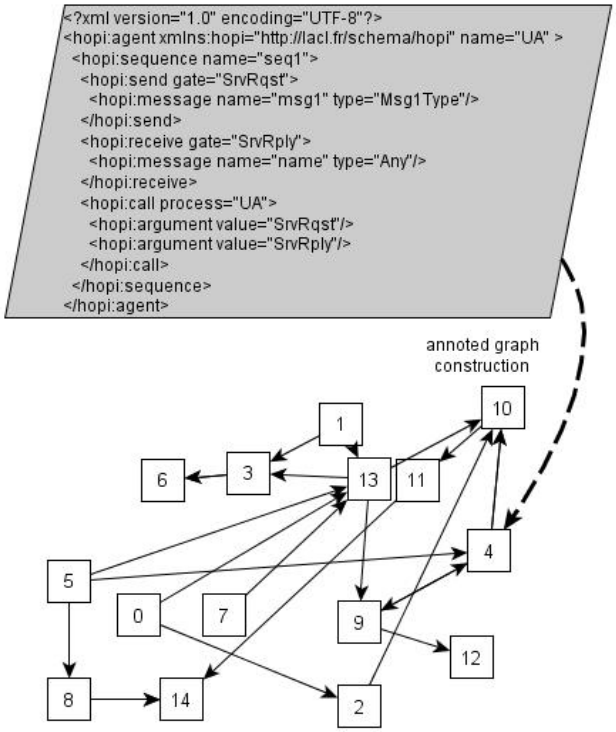

Figure 4. From a higher order Pi-Calculus into an annoted graph.

The previous figure (Fig. 4) shows an XML view of the Pi-Calculus specification of *UA* agent (SLP protocol). After enrichment, we obtained another graph where each node is an XML tag with new attributes and process annotations. More precisely, we save into annotations technical information useful for the construction of the final BPEL script and WSDL script.

C. BPEL and WSDL attributes

We have defined two main schemas; one is about useful details of WSDL language and another about useful details of BPEL language. So, we can label a node with attributes called: *partnerLink*, *variable*, *portType*, etc. For instance, if we consider a part of the specification of *UA* agent (complete definition is given in annex): an emission (1) on channel *SrvRqst*, then the corresponding labeled node is presented (2) below and the BPEL skeleton is displayed as (3):

$$\overline{SrvRqst}\langle Prmt, SrvRply\rangle \quad (1)$$

Previous parsing of input XML sources provides that *SrvRqst* channel is a link between *UA* agent and *DA* agent. Also send tag is transformed as follows,

```
<hopi:send gate="SrvRqst"                    (2)
  bpel:partnerLink="DA" bpel:portType="ns1:DA">
  <hopi:message name="msg1" type="Msg1Type"/>
</hopi:send>
```

During the synthesis of all tagged graph, a part of the BPEL action is given as follows.

```
<bpws:reply name="Reply"                     (3)
operation="SrvRqst" partnerLink="DA" portType="ns1:DA"
wpc:displayName="Reply" wpc:id="3">
  <wpc:input>
    <wpc:parameter name="msg1" variable="msg1"/>
  </wpc:input>
</bpws:reply>
```

Receive and reply activities go hands in hand in a request-response flow. After this output message, *UA* agent receives detail about *Print* service. We follow the same approach as before, first the Pi-Calculus term, then the tagged node and finally BPEL action.

$$SrvRply(Name) \quad (4)$$

As before, the dependency graph provides that the input channel is a link between *UA* agent et *IdleDA* agent.

```
<hopi:receive gate="SrvRply"                 (5)
  bpel:partnerLink="IdleDA"
  bpel:portType="ns1:IdleDA">
  <hopi:message name="name" type="Any"/>
</hopi:receive>
```

Finally, a BPEL action is:

```
<bpws:receive createInstance="yes"           (6)
name="Receive" operation="SrvRply" partnerLink="IdleDA"
portType="ns1:IdleDA" wpc:displayName="Receive"
wpc:id="2">
  <wpc:output>
    <wpc:parameter name="Name" variable="Name"/>
  </wpc:output>
</bpws:receive>
```

Receive activity is known as blocking activity as in Pi-Calculus. This means it will wait till any message received. And it will create a new process instance. Inside the receive activity an output element is specified which refer to the request variable. The request variable data can be used in other activity in the business process.

Then the definition of the agent ends with a call to UA definition. We follow the same approach as before, first the Pi-Calculus term, then the tagged node and finally BPEL action.

$$\nu\, Print\, UA\langle SrvRqst, SrvRply\rangle \quad (7)$$

The dependency graph offers a lot of metrics such as scope and depth. Also we can label the XML tags as follows:

```
<hopi:call process="UA"                      (8)
  bpel:partnerLink="UA"
  bpel:portType="ns1:UA"
  bpel:operation="process">
  <hopi:argument value="SrvRqst"/>
  <hopi:argument value="SrvRply"/>
</hopi:call>
```

The depth of this call corresponds to a distance computation in dependency graph. The closest definition is considered as a solution.

```
<invoke partnerLink="UA" portType="ns1:UA"   (9)
operation="process" inputVariable="UARequest"
outputVariable="UAResponse" />
```

UARequest and *UAResponse* are declared as local variable of the process definition. Their type is automatically computed from the input XML source. In our case, *UARequestType* is a couple of information and *UAResponseType* is a Boolean value as acknowledgment. We could detail all the primitive of the syntax presented before but the size of the document does not allow us to give more details about them. We have presented these three actions because they support higher order feature of the formal language. In definition (2), the Msg1Type can be the definition of another agent. This means that the message is linked to the port type of the mobile agent. So, the agent host

 ISBN: 978-1-61208-184-7

can then invoke all operations presented into the description of this port type.

D. Synthesis and control

After creating the upper part of the WSDL definition of agent, controls have to be applied to validate relations between the agents of the agency. This means that BPEL skeletons are used to check partner link definition and also their role into the main script. Type checking is also applied on variable used as parameters or as local data. The objective is to provide XML definitions as good as possible to specifiers.

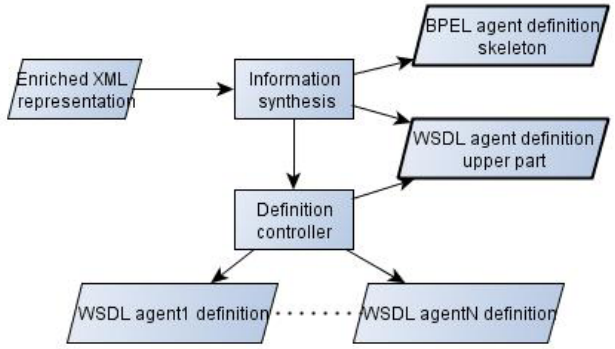

Figure 5. Information synthesis.

Previous diagram (Fig. 5) depicts the last part of our approach. As mentioned in the last section, a labeled graph is synthesized to obtain a couple of description (bold lines). Next controls are done by the use of type definition as XML schema and agent definition as WSDL definition.

IV. ARCHITECTURE DESCRIPTION

Because mobile agent system exploit network and is deployed over a set of computers, it is necessary to have a specification language that can model computers at a higher level of abstraction and enable analysis of description. The language should be powerful enough to capture high-level description of software architecture. On the other hand, the language should be simple enough to allow correlation of the information between the specification and the architecture manual.

Architecture Description Languages (ADL) enables design automation of embedded processors [21]. The ADL specification is used to generate various executable models including simulator, compiler and hardware implementation. This language is a reference in the architecture specification domain but it is not natural to compose such a specification with other process algebra specification.

A. Agent host and neighboring

We consider each node of our network as a future host for receiving mobile agents. Also, it is essential to describe which the local services available are for an incoming agent. More precisely, this can be viewed as a first security layer where local services are callable under condition on the role of the caller.

We need a description language for our software architecture which can be composed with HIGH ORDER (H.O.) Pi-Calculus. Matthew Hennessy proposes a process algebra called SafeDPi, which is based on Pi-Calculus. This is an extension used to type processes depending on their location [22]. Also, this is precisely what is important in our context of partner link and end point definition. If a unique resource location (URL) is used to call a Web service. We need to express this uniqueness into our specifications and define migration of agent based on this feature. Moreover, SafeDPi language is defined to embed higher order Pi-Calculus definition of agent host [23].

So, software architecture takes the form:

$$[\![host]\!]_1\,[\![UA]\!] \mid (new\ SrvReg{:}E)([\![host]\!]_2\,[\![SA]\!] \mid [\![host]\!]_1\,[\![DA]\!]) \quad (10)$$

Where there are two agents *UA* and *DA*, which are on the same location $host_1$, and the agent *SA* is running on another location called $host_2$. The agents *DA* and *SA* share information called *SrvReg* which is the gate to publish a service into the registry. The agents *UA*, *SA* and *DA* are defined with H.O. Pi-Calculus language.

We use this language to provide a formal description of all nodes which can host mobile agents. New location can also be taken into account as a new configuration of the network. So, our case study use 3 agents on 2 distinct nodes called $host_1$ and $host_2$.

$$location[host_1 : Host_1, host_2 : Host_2] \quad (11)$$

The type of the locations defines which kind of agent can be deployed on it. (12)

$$Host_1 = location[SrvRqst{:}w(DA), SrvRply{:}r(IdleDA)]$$

This definition stresses that a node of type $Host_1$ can support an agent which exploits a couple of resources called *SrvRqst* for sending a message to *DA* and *SrvRply* for receiving a message from *IdleDA*. The location definition should be completed to support also a *DA* agent. The $Host_2$ definition is defined with the same approach. We use such specification as a set of constraints for the deployment step. When an agent is deployed or installed initially on a node which is specified as previously; we can checked whether the communication services are compatible.

As in Section 2, we have defined an XML schema for SafeDPi-Calculus language. A deployment specification is an XML flow and we compare provided services of a node like $host_1$ with the required services used by *UA*. This is done importation of an XML flow into another one and the control of invoke, receive and reply actions.

B. Local resources access and migration

Previously, we specified message types. Also, channel can also be types. For instance, a link between $host_1$ and $host_2$ can only support message of type *Document*. Also, we have added type on communication link. Now, we restricted the definition (12) into (13) to limited value on gate.

$$Host_1 = location\begin{bmatrix} SrvRqst{:}w(Msg1Type{:}DA), \\ SrvRply{:}r(Msg2Type{:}IdleDA) \end{bmatrix} \quad (13)$$

Now, we can check message type from agent specification and possible message type into deployment specification. But we need to have more control about the definition of node. Also want to express that from node $host_2$, it is possible to move to $host_1$ but not from another

node. Also, we place oriented links between nodes. So, location declaration can be enriched as follows:

$$[host]_{,1}\,[UA] \,|\, [goto]_{,pt}\ [host]_{,2}\,SA \rightarrow [host]_{,1}\,[UA] \,||\, [host]_{,2}\,[SA]$$

This expresses that on $host_1$ the agent *UA* can be placed in parallel with a mobile agent with the ability to move towards $host_2$. Then, after the migration, each code can continue its evaluation on two separate nodes. A constraint is added on the migration statement: this will occur though the port type called *pt* on $host_2$. This allows us to add new deployment restrictions. Because these restrictions can be checked into agent specification, we can raise anomalies if a rule is not respected.

C. SLP case study

As presented before, this protocol is suitable for our presentation. It simulates the need of a print service by a client called *UA*. The service *print* is published by an agent called *SA* into a registry called *DA*. In nominal scenario, when *UA* agent asks the registry to know where the *print* service is, then it receives the service on the node where it is. Thus, we have to express code mobility and agent moves from $host_2$ to $host_1$. The data to be printed do not move on the network. This part of the description is just specified but the use of one specification level called software specification. Then, physical constraints are described through another level of specification called deployment specification. Because the formats of these specifications are compatible, we are able to combine them and check is software constraints are satisfied through physical constraints.

V. CONCLUSION

Through this paper, we have described a process to generate executable representation from formal specification. Of course, this work is currently prototyped through several examples and we need to complete our BPEL generator to help designer into his business process definition.

We think that design of distributed application can evolved by the use of mobility feature. Engineer has to separate the concerns: a level for software component and another for the deployment. With mobile agent, placement is not frozen from load time. But depending on runtime, mobile agents can move component and adapt initial placement as a new configuration for continuing the execution. Next direction is to provide our work to project partner for deeper validation. The extensions of Pi-Calculus language are as rich as extensions of BPEL language, also we are confident in our approach to assist business analyst in a more formal approach and check business property of his whole system.

REFERENCES

[1] C. Moemeng, V. Gorodetsky, Z. Zuo, Y. Yang, and C. Zhang, Agent-Based Distributed Data Mining: A Survey, L. Cao (ed.), Data Mining and Multiagent Integration, Springer Science + Business Media, May 2009, pp. 234-246.

[2] Takahashi, H. and Kavalan, V., A mobile agent for asynchronous administration of multiple DBMS servers, Systems Management, 1998. Proceedings of the IEEE Third International Workshop on 22-24 Apr 1998 pp. 32 - 33.

[3] Ponci, F. and Deshmukh, A.A., A Mobile Agent for measurements in distributed power electronic systems, Instrumentation and Measurement Technology Conference Proceedings, 2008. IMTC 2008. IEEE, 12-15 May 2008, pp. 870 - 875.

[4] Giacomo Cabri, Letizia Leonardi, and Franco Zambonelli, Mobile Agent Coordination for Distributed Network Management, Journal of Network and Systems Management Volume 9 Issue 4, Dec 2009.

[5] Verdejo, A. and N. Martí-Oliet, Implementing CCS in Maude 2, in: F. Gadducci and U. Montanari, editors, Proc. 4th. Intl. Workshop on Rewriting Logic and its Applications (2002)

[6] M. Große-Rhode. A Compositional Comparison of Specifications of the Alternating Bit Protocol in CCS and UNITY Based on Algebra Transformation Systems. In K. Araki, A. Galloway, and K. Taguchi, eds., Proceedings of the 1st International Conference on Integrated Formal Methods (IFM'99), pages 253–272, UK, 1999. Springer Verlag.

[7] B. C. Pierce, D. R#my, and D. N. Turner. A typed higher-order programming language based on the picalculus. In Workshop on Type Theory and its Apphcation to Computer Systems, Kyoto University, July 1993.

[8] R. Milner. Communicating and Mobile Systems: The Pi-Calculus. Cambridge University Press, Cambridge, UK, May 1999.

[9] Davide Sangiorgi. Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms. PhD thesis, LFCS, University of Edinburgh, Avr 1993.

[10] C.Perkins and E. Guttman. Service Location Protocol (SLP), Version 2. Sun Microsystems, http://www.ietf.org/rfc/rfc2608.txt.

[11] F. Curbera et al. Business process execution execution language for web services, version 1.0. Standards proposal, BEA Systems, International Business Machines Corporation, and Microsoft Corporation, http://www-106.ibm.com/developerworks/library/ws-bpel/, 2003.

[12] Siebel. Business process execution language for web services bpel4ws, version 1.1. http://www.siebel.com/bpel, 2003.

[13] Dave Clarke. A Basic Logic for Reasoning about Connector Reconfiguration. Fundamenta Informaticae 81(4):361-390, Jun 2008.

[14] Sascha Kluppelholz and Christel Baier. Symbolic model checking for channel-based component connectors. Science of Computer Programming 74(9):688-701, Sep 2009.

[15] Sun Meng, Farhad Arbab, and Christel Baier. Synthesis of Reo circuits from scenario-based interaction specifications. Science of Computer Programming 76(8):651-680, Avr 2011.

[16] Carolyn Talcott, Marjan Sirjani, and Shangping Ren. Comparing three coordination models: Reo, ARC, and PBRD. Science of Computer Programming 76(1):3-22, May 2011.

[17] Farhad Arbab. Elements of Interaction. In Marc Aiguier, Francis Bretaudeau, and Daniel Krob, editors, Complex Systems Design & Management, pages 1-28. Springer, 2010.

[18] Faisal Abouzaid. Toward a pi-calculus based verification tool for web services orchestrations. In Proceedings of the 8th International Conference on Enterprise Information Systems (ICEIS06), Paphos 2006.

[19] F. Abouzaid, "A Mapping from Pi-Calculus into BPEL", in Proc. ISPE CE, 2006, pp. 235-242.

[20] Uwe Nestmann and Frank Puhlmann: Business Process Specification and Analysis. In Process Algebra for Parallel and Distributed Computing. Boca Raton, Chapmann & Hall/CRC Press (2009) pp. 129-160

[21] ANSI/IEEE Std 1471™-2000, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems.

[22] Matthew Hennessy , Julian Rathke , and Nobuko Yoshida . SafeDPi: A language for controlling mobile code (2003). In Proc. FOSSACS, LNCS 2987

 ISBN: 978-1-61208-184-7

[23] C Lhoussaine. Type inference for a distributed pi-calculus (Jun 2003). Science of Computer Programming

ANNEX

$$\mathbf{DA(SrvReg, SrvUnReg, SrvRqst, SrvAck)} = \nu(input, reset, channel, inputIdle, resetIdle)$$
$$(SrvReg(S_{reg}) . ((DA_{Mem}(input, reset, channel, inputIdle)$$
$$| IdleDA_{Mem}(input, resetIdle, channel, inputIdle) | \overline{input}(S_{reg}) . \overline{SrvAck})$$
$$+ SrvUnReg(S_{reg}) . \overline{reset} . \overline{resetIdle} . \overline{SrvAck})$$
$$| SrvRqst(S_{rqst}, ch) . \overline{channel}(S_{rqst}, ch))$$
$$. DA(SrvReg, SrvUnReg, SrvRqst, SrvAck)$$

$$\mathbf{UA(SrvRqst,\ SrvRply)} = \nu(Print)$$
$$\overline{SrvRqst}(Print, SrvRply) . SrvRply(Name). UA(SrvRqst, SrvRply)$$

$$\mathbf{IdleDA_{Mem}(input,\ resetIdle,\ channel,\ inputIdle)} =$$
$$inputIdle(S_{reg}) . channel(S_{rqst}, ch) . [S_{reg} = S_{rqst}] \overline{ch}(S_{reg}) .$$
$$\overline{input}(S_{reg}) . IdleDA_{Mem}(input, resetIdle, channel, inputIdle) + resetIdle.0$$

$$SA_1(SrvReg_{SA_1}, SrvDeReg_{SA_1}, SrvAck_{DA_1}) =$$
$$\overline{SrvReg}_{SA_1}(Service(print, f)). SrvAck_{DA_1}$$
$$. \overline{SrvDeReg}_{SA_1}(Service(print, f)). SrvAck_{DA_1}$$
$$. SA_1(SrvReg_{SA_1}, SrvDeReg_{SA_1}, SrvAck_{DA_1})$$

 ISBN: 978-1-61208-184-7