Formally Designing Web Services for Mobile Team Collaboration [*]

Schahram Dustdar, Pascal Fenkam
Technical University of Vienna, Distributed Systems Group
A-1040 Vienna, Argentinierstrasse 8/184-1
{s.dustdar, p.fenkam}@infosys.tuwien.ac.at

Abstract

We illustrate a symbiotic relationship between existing model oriented specification techniques and web services. Through the formal re-design of a platform for mobile team collaboration, we investigate the suitability of existing formal specification and verification techniques to web services. In general, this preliminary work suggests that a lot is still to be done on understanding the computational behavior of web services, hence on specifying and verifying them.

1 Introduction

Service-oriented Computing (SoC) has received considerable attention and many view it as the emerging distributed computing model for Internet applications. The Web service paradigm is one of the architectural manifestations of SoC. A web service is a self-contained, self-describing, instantiable, modular, Internet-, and component-based application. Web services interact using XML messaging protocols (e.g. SOAP [8]) and are described using XML technologies (e.g. WSDL [2], DAML-S [3]). While an important number of middleware have being developed for the deployment of such services, constructing dependable web services requires more research.

To get a flavor of the challenges in applying formal methods to the development of web services, we redesign the MOTION [5, 12, 24, 20](MObile Teamwork Infrastructure for Organizations Networking) platform which is a platform for mobile team collaboration that one of this paper's authors was co-architect and co-developer of. The MOTION platform is a typical example of enterprise wide, open, and Internet- and component-based application, hence, an interesting candidate for the application of the web service technology.

This paper shows that existing techniques for the formal specification and analysis of applications may be used for the construction of web service based applications. Despite the apparent straightforwardness of this exercise, however, many difficulties exist, one of which is interference control. A notable way to avoid this side effect is mutual exclusion which can be achieved in practice via transactions. The development of such systems is, however, still undergoing. The paper finally shows that a stepwise development approach is necessary for the construction of intelligible web services.

The remainder of the paper is organized as follows. The next section gives an overview of related attempts to specifying and verifying web services. In Section 3, we introduce the MOTION platform and its messaging service that we specify and discuss in this paper. Section 4 gives a brief overview of the abstract model used in this paper for specification of web services. The MOTION messaging service is specified in Section 5 and the resulting challenges are discussed in Section 6. Finally, Section 7 concludes the paper.

2 Related Work

The section gives an overview of existing works in formal specifying web services. Although some works have been presented on formal specifying web services, most of them are oriented towards verification and less towards showing how formal methods can be exploited to ease the design of web service based applications. The main trends in specifying web services are model-checking and petri net-based approaches.

[*]This work was partly supported by a research award for Development Methods for Dynamic Web services Workflows of the Chamber of Commerce Vienna (Wirtschaftskammer Wien) and by the Austrian Research Foundation Fond (FWF) through the RAY project (Number P16970-No4)

2.0.1 Model Checking Web Services.

This approach is defended by Nakajima in [14, 15] where a WSFL (web service flow language) flow description is translated into Promela, the input language of the SPIN model checker, and some properties such as reachability, deadlock-freedom, or application specific progress properties are checked.

2.0.2 Petri Net Based Analysis of Web Services.

Unlike the above approach which simply consists in applying model-checking to the verification of web services, Hamadi and Benatallah [9] propose a general theoretical algebraic framework for modeling and composing Web services. The service algebra operators are given by a language including constructs such as the parallel, the sequential, the choice, the iteration, and the discriminator operators. The semantics of this web service language is given using Petri nets. Although no means are given from transforming a resulting algebraic specification into a concrete Web-service (e.g. refinement), this work represents a valuable first step towards understanding the behavior of web services.

2.0.3 The DAML-S based Approach.

This approach is proposed by Narayanan and Mcllraith [21] and is based on the DAML-S ontology for web services [3]. DAML+OIL is a general XML, RDF - based framework for describing ontologies which are taxonomic information. DAML-S is one of the many ontologies defined in this framework.

3 Overview of MOTION

MOTION is a platform we designed and prototyped in the MOTION European project [5, 12, 24, 20] where the needs of two well known organizations were addressed. The first is a manufacturer of mobile phones and the second is a producer of white goods (e.g. refrigerators, washing machines). The platform has a service architecture that supports mobile teamworking by taking into account different connectivity modes of users, provides access support for various devices, supports distributed search of users and artifacts, offers user management facilities and supports various messaging techniques.

3.1 The MOTION Architecture

The MOTION platform was constructed by assembling different components: DUMAS (Dynamic User

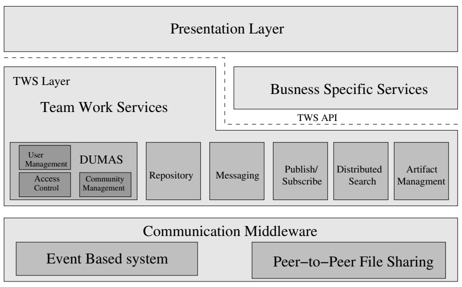

Figure 1. The MOTION Architecture

Management and Access Control System) [6], an XQL engine [7], a repository (comparable to a file system), an artifact manager (comparable to a shell that provides primitives for accessing the file system), etc. These components are integrated into the platform by means of the event-based architectural style implemented by PeerWare [18] (see Figure 1).

The MOTION platform is based on a peer-to-peer (p2p) architecture, by which we mean that each device may host and manage a service (user management, artifact management, SMS service, etc.) independently of the behavior of other devices. The justification of this architecture can be found in [20]. We call each device in this p2p architecture a peer (or a MOTION peer). This paper addresses the issue of messaging is MOTION.

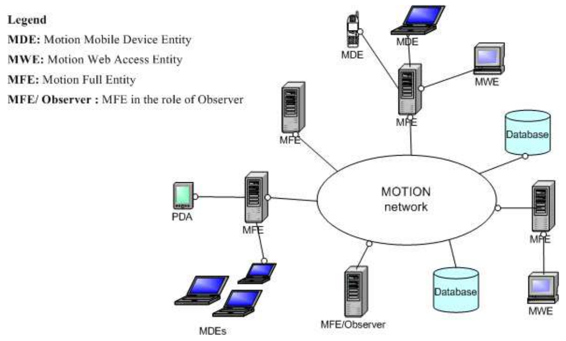
Legend
MDE: Motion Mobile Device Entity
MWE: Motion Web Access Entity
MFE: Motion Full Entity
MFE/ Observer : MFE in the role of Observer
MDE
MDE
MWE
MFE
MFE
Database
MOTION network
PDA
MFE
MFE
MDEs
MFE/Observer
Database
MWE

Figure 2. The Conceptual View of MOTION

3.2 Conceptual View of MOTION

MOTION peers have different capabilities and cannot, therefore, be used equally for storing data and hosting services (see Figure 2). To support this heterogeneity we give the customers the possibility to specify which profiles and services they would like to store on their peers. In terms of services, we may say that each device hosts a user management service. On the other hand, a peer may host a service such as an SMS service that allows other peers to send short messages to mobile phones.

Although the notions of service and web were present in the MOTION vocabulary, the notion of web service was not explicitly and formally there. We now re-organize the platform and distinguish the following basic web services: User Management WS, Community Management WS, Messaging WS, Publish/Subscribe WS, Artifact WS, Search WS, Access Control WS, Authentication WS. In particular, the layered view of the MOTION architecture is now extended to include a SOAP communication middleware on which other components rely.

3.3 Messaging in MOTION

MOTION Messaging is an integrated messaging service that enables users to communicate and exchange information. Notifications based on subscriptions are also delivered by this messaging service. MOTION messages are sent to users using technologies such as desktop notifications, email (i.e., SMTP), and GSM short messages (SMS). MOTION Messaging enables employees to stay in direct and constant contact no matter what devices they are currently using.

The MOTION Messaging component in our prototype consists of five main components including the SMS web service, the SMTP (email) web service, the desktop messaging web service, the main messaging service, and the repository that some of them may access (see Figure 3). The MOTION messaging web service is the interface between the business specific services and remainder of the messaging apparatus.

4 An Abstract Web-Service Model

We present a VDM-based abstract web service model. VDM [19] is an environment for developing correct model-oriented applications. The method is composed of a formalism for specification and techniques for stepwise refinement. This systematic development approach makes it suitable for the development of complex software systems. The language can be used for abstract specification as well as for low-level specification. VDM supports representational and operational abstractions. Representational abstraction describes the modeling primitives necessary to specify a software. By operational abstraction, the behavior of the software is captured through functions and operations. A complete description of the language can be found in [25].

Our abstract web service model includes the sequence construct, the if-the-else construct, the repeat construct, and the parallel construct. The notion of service is modeled with the VDM++ concept of class.

5 Modelling Messaging Web Services

We model the MOTION messaging web service. This service includes the a GSM service, a SMTP service, and a desktop messaging service which are addressed depending on the availability criteria of the user. In fact, each user has the possibility to specify her availability criteria which can be SMS, email, or desktop message. Messages are sent as desktop messages when the user is reachable through the MOTION platform. The following sequence diagram characterizes the case where the user's availability criteria is SMS.

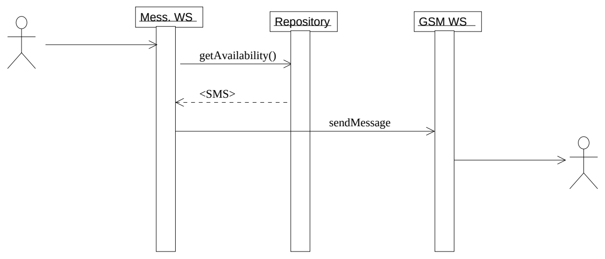

Figure 4. Messaging Sequence Diagram

Before presenting the specification of the different actions we present the data types related to this specification and defined in the class *WebService*. A message (*Message*) is simply a sequence of characters and a mailbox maps an address to the set of messages received at this address. Further, some of the data types are not further specified and, therefore, defined as token. An address is either an email address, an SMS address (i.e. a mobile phone number in normal cases), or a desktop address.

$UserID = \mathsf{token}$; $DesktopAddress = \mathsf{token}$; $EmailAddress = \mathsf{token}$;

$SMSAddress = \mathsf{token}$; $Address = DesktopAddress \mid SMSAddress \mid EmailAddress$;

$Message = \mathsf{char}^*$; $Mailbox = Address \xrightarrow{m} (Message\text{-}\mathsf{set})$

On the other hand, instance variables are defined and include *availability*, *sm*, *em*, and *dm*. The first maps each user identifier to the address at which she is available. For instance, if the user with identifier

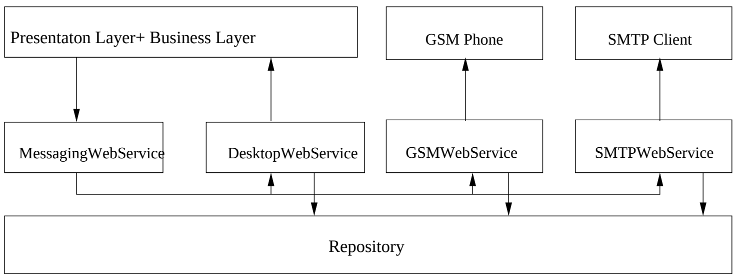

Figure 3. The MOTION Messaging Architecture

Jane is available per email, *availability*(*Jane*) would return her email address. The three other variables are respectively for mapping SMS addresses, email addresses, and desktop addresses to messages received at these addresses. Clearly, it is required through the invariants that all addresses in the domain of *em*, *sm*, and *dm* are email, SMS, and desktop addresses respectively. The invariant on the SMS mailbox additionally requires that any message in its range be of length smaller that $SMSMAX$, a natural constant value.

```
$availability : \; UserID \xrightarrow{m} Address;$

$dm : \; Mailbox;$
    inv $\triangleq \forall ad \in$ dom $dm \cdot$ is_$(ad, DesktopAddress)$

$em : \; Mailbox;$
    inv $\triangleq \forall ad \in$ dom $em \cdot$ is_$(ad, EmailAddress)$

$sm : \; Mailbox;$
    inv $\triangleq \forall ad \in$ dom $sm \cdot$ is_$(ad, SMSAddress) \quad \wedge$
$\forall msg \in \bigcup$ rng $sm \cdot$ len $msg < SMSMAX$
```

A high level specification of the messaging service that we intend to construct is given below. Note how the behavior of this service is independent of any other service. Its pre-condition requires that the availability medium of the specified user is indeed defined.

```
class $MessagingWebService$ is subclass of $WebService$

    $SendMessage\,(message : Message, receiver : UserID)$
    ext wr $em, sm, dm$ :
    pre  $availability\,(receiver) \in$ dom $em \cup$
         dom $sm \cup$ dom $dm$
    post let $ad = availability\,(receiver)$ in is_$(ad, EmailAddress)$
             $\Rightarrow \overleftarrow{em}\,(ad) \cup \{message\} = em\,(ad) \wedge$
         is_$(ad, SMSAddress) \;\Rightarrow\; (\overleftarrow{em}\,(toEmail(ad)) \cup$
          $\{message\} = em\,(toEmail(ad))) \wedge \overleftarrow{sm}\,(ad) \cup$
         $\{substring\,(message, SMSMAX)\} = sm\,(ad) \wedge$
         is_$(ad, DesktopAddress) \;\Rightarrow\; \overleftarrow{dm}\,(ad) \cup$
         $\{message\} = dm\,(ad)$

end $MessagingWebService$
```

The post-condition requires that if the specified user is available through email, then she should receive this message as email. A similar requirement applies to the desktop medium. For the SMS medium, however, only the first $SMSMAX$ characters of the message are sent to the user via a GSM network. In addition, the entire message is sent to the user as email. A mechanism, called *toEMail*, is assumed that transforms an SMS address to an email address.

The next specification represents a refinement of the above service. We invoke specialized services for sending emails, SMS, and Desktop messages.

```
class $MessagingWebServiceR$ is subclass of $WebService$

    $SendMessage : Message \times UserID \xrightarrow{o} ()$

    $SendMessage\,(msg, receiver) \triangleq$
      let $ad = availability\,(receiver),$
          $em = toEmail\,(ad),$
          $ad = availability\,(receiver)$ in
      if (is_$(ad, EmailAddress)$)
        then $SMTPWebService`SendEmail(msg, ad)$
      else if (is_$(ad, SMSAddress)$)
          then { $GSMWebService`$
                $SendSMS(substring\,(msg, SMSMAX), ad) \|$
                $SMTPWebService`SendEmail(msg, em)\}$
          else if (is_$(ad, DesktopAddress)$)
              then $DesktopWebService`$
                  $SendDesktopMessage(msg, ad)$
    pre  $availability\,(receiver) \in$ dom $em \cup$ dom $sm \cup$
dom $dm$

end $MessagingWebServiceR$
```

The proof that $MessagingWebServiceR.SendMessage$ satisfies the specification $MessagingWebService.SendMessage$ is done in the traditional predicate calculus. The specification of a $GSMWebService$ is given below. The services $SMTPWebService$ and $DesktopWebService$ are specified in a similar manner. Unlike the specification of $MessagingWebService.SendMessage$ that we refined, we need not to do so for $GSMWebService$; it is provided by some third party. We, therefore, do not need to know the details of this service.

```
class $GSMWebService$ is subclass of $WebService$

        $SendSMS\,(msg : Message, receiver : SMSAddress)$
        ext wr $sm$ :
        pre  $availability\,(receiver) \in$ dom $sm \wedge$
             len $msg < SMSMAX$
        post let $ad = availability\,(receiver)$ in
                 $sm\,(ad) = \overleftarrow{sm}\,(ad) \cup \{msg\}$

end $GSMWebService$
```

Each user possesses an SMS mailbox. Sending her an SMS, therefore, consists in storing it in this mailbox.

6 Discussion

The above specification seems straightforward; it however, hides some fundamental issues that we discuss in this section. Prior to this discussion, we present the advantages of this approach when applied to the design of web services.

6.1 Stepwise Specification of Web Services

The MOTION Messaging case study reveals the advantages of the concepts of explicit and implicit specifications that have been proposed in the VDM methodology. An implicit specification is defined by formulating only the pre- and post-conditions of a method. An explicit specification is constructed by specifying with more details how the behavior of the method could be realized (e.g. *MessagingWebServiceR.SendMessage*).

In the context of web services, we notice that we have required implicit specifications for all actions while explicit specifications were needed only for services that we intend to construct ourselves. The example shows that we are not interested in the details of the implementation of *GSMWebService*. In this case, the explicit specification is sufficient; it informs us about the behavior of the service while hiding unnecessary details. *We, therefore, argue that some kind of implicit specifications are important for the specification of web services.*

In the case of a web service that we intend to construct, the implicit and explicit specifications are required. The implicit specification is intended for the user of the services on one hand, and for us on the other hand. It captures the behavior of a web service without showing implementation details. An explicit specification shows how we intend to construct a given service. The relationship between the implicit and the explicit specifications is a refinement relation. These two ways of specifying web services suggest that a web service be constructed by:

1. defining the implicit specification of its actions,
2. showing some desired properties about the implicit specifications of the actions,
3. defining the explicit specifications of these actions,
4. showing that an explicit specification is indeed a refinement of the related implicit specification,
5. translating the explicit specification into a dedicated web service description language (e.g. BPEL4WS, DAML-S [3], WSDL [2]).

This approach ressembles that of Heckel et al. [10, 1] who propose using UML to first model the desired application, then using OCL or CSP for checking the properties of the future application before translating the model into a web service description language [10]. We believe that approaches that recommend modelling applications using languages such as BPEL4WS and DAML-S are inadequate to the development of dependable web services since it is impossible to formulate fundamental properties such as invariants on data types.

6.2 Interference in Web Services

One of the issues that we discuss in this paper is that of interference: the fact for two or more processes to concurrently access some shared resource which may be an instance variable within an object, an entry in a database, or a printer.

In the previous section we argued that it should be possible to specify web service actions by means of pre- and post-conditions. Since pre- and post-conditions have been used essentially for the specification of sequential programs, our example (and DAML-S) may suggest that web services are sequential systems. This is false; if not explicitly constrained, web service instances are executed in parallel. In this case, it is likely that interference will occur, often resulting in unexpected results.

Let us consider our case study with *Jane* being a user identifier with the mobile phone number X and such that $availability(Jane) = X$. That is, *Jane* is available via her mobile phone. Finally, we assume that $sm(X) = \{\}$; that is, *Jane* has no SMS in her SMS mailbox. From this state, we let two other users *Max* and *John* send the messages "You are beautiful!" and "Let us meet!" to *Jane* respectively and simultaneously. This results in a final state satisfying

$sm(X) = \{$"Your are beautiful!", "Let us meet!"$\}$. From *Max* viewpoint, this does not satisfy the post-condition of the implicit specification *MessagingWebService.SendMessage* which requires that the equality $sm(X) = \{$"Your are beautiful!"$\}$ holds. The invocation of the same service by *John* has interfered with the invocation by *Max*, resulting in some unexpected results.

A way to avoid this would have been to underspecify *SendMessage* in the following way:

```
class $MessagingWebService$ is subclass of $WebService$

        $SendMessage\,(msg : Message, receiver : UserID)$
        ext wr $em, sm, dm$ :
        pre  $availability\,(receiver) \in$ dom $em \cup$ dom $sm \cup$
             dom $dm$
        post let $ad = availability\,(receiver)$,
                 $em = toEmail\,(ad)$ in
             is_$(ad, EmailAddress) \Rightarrow msg \in em\,(ad) \wedge$
             is_$(ad, SMSAddress) \Rightarrow (msg \in em\,(em)) \wedge$
             $substring\,(msg, SMSMAX) \in sm\,(ad) \wedge$
             is_$(ad, DesktopAddress) \Rightarrow msg \in dm\,(ad)$

end $MessagingWebService$
```

This would, however mean that an implementation that not only sends the required message, but also some 100000 other messages, satisfies the specification! *Our conjecture is, therefore, that pre- and post-conditions are not sufficient for describing the behavior of web service activities.* It is widely recognized that pre- and post-conditions are not sufficient for the specification of concurrent systems; in this sense, one may argue that our conjecture is a well known truth. Although this objection may be valid, it is important to recall it since there is an increasing trend of specifying web services in this way. We show in the next section that even existing approaches for specifying concurrent systems are still inadequate for web services.

6.3 The Inadequacy of Rely-/Guarantee Specifications

Rely- and guarantee conditions have been proposed [11, 26, 22, 4] for the specification of shared state parallel programs. A program specification is of the form $(P,\ R,\ G, E)$ where P, R, G, and E are the pre-, rely-, guar-, and post-conditions respectively. A program satisfies its specification if, when started in a state satisfying its pre-condition and in an environment whose transitions satisfy R, it terminates in a state satisfying E and each of its transition satisfies the guar-condition G. This is the most compositional approach to the stepwise development of shared state parallel programs [16].

We conjecture that the rely-/guarantee approach to specifying concurrent systems is not adequate to web services. In fact, the rely-condition R captures the behavior of the environment in which a component z is intended to be executed. When deploying a component, a developer is responsible of ensuring that this condition holds. In the case of web services, however, a service is not shipped to the using developer for deployment. It is, therefore, impossible for this developer to ensure the rely-condition of the components used within the web service. In fact, we believe that the owner of the web service is responsible in ensuring that the environment of the deployed components satisfies their rely-conditions. A way to do this is to use mutual exclusion techniques such as transactions.

Mutual exclusion mechanisms allow a program to exclude other programs from accessing the same resources as they do. In practice, mutual exclusion is achieved using e.g. transactions and semaphores. The necessity of this paradigm concerning dependable web services is increasingly acknowledged and research is going on for defining suitable semantics and constructing prototype implementations. An overview of requirement to and difficulties of these tasks are given by [13, 23, 17].

7 Conclusion

We presented an attempt to introduce web services into an existing platform, MOTION for mobile team collaboration. The messaging part of this middleware was considered and a formal specification constructed. Though this exercise revealed that the traditional stepwise development approach of software systems can be applied to the development of web service based applications, the issues of formal specifying web services remains widely open. While rely- and guarantee conditions have successfully been used for the specification of shared state concurrent applications, it can not be applied "as it" to web services. There is no means for the user of a web service to constrain the environment in which it is running. Description languages based on XML such as DAML-S and BPEL4WS are (1) not easily readable and (2) are at a lower level of specification; they specify how a web service is intended to be implemented, and (3) do not suitably support data modelling.

References

[1] L. Baresi, R. Heckel, and D. Varr S. Thöne. Modeling and validation of service-oriented architec-

tures: Application vs. style. In *Proc. of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2003)*, Sept. 2003.

[2] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web service description language (wsdl) 1.1,w3c note, March 2001. Available from http://www.w3c.org/TR/wsdl.

[3] J. Hobbs O. Lassila D. Martin S. Mcllraith S. Narayanan M. Paolucci T. Payne K. Sycara H. Zeng) DAML Services Coalition (A. Ankolekar, M. Burstein. Daml-s: Semantic markup for web services. In *Proceedings of the International Semantic Web Working Symposium (SWWS 2001)*, November 2001.

[4] J. Dingel. *Systematic parallel programming.* PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, December 1999.

[5] Pascal Fenkam, Schahram Dustdar, Engin Kirda, Harald Gall, and Gerald Reif. Towards an access control system for mobile peer-to-peer collaborative environments. In *IEEE 11th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET-ICE 2002)*, pages 95–100, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, Jun. 10-12 2002. IEEE Computer Society Press.

[6] Pascal Fenkam, Harald Gall, and Mehdi Jazayeri. Constructing CORBA Supported Oracles: A Case Study in Automated Software Testing. In *Proceedings of the 17th IEEE Automated Software Engineering Conference, Edinburgh, Scotland*, pages 129–138, September 2002.

[7] Ingo Macherius Gerald Huck. GMD-IPSI XQL Engine Tutorial. Technical report, GMDs XML COmpetence center, 1999. Available from http://xml.darmstadt.gmd.de/xql/.

[8] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, and Henrik Frystyk Nielsen. SOAP Version 1.2 Part 1: Messaging Framework, W3C Recommendation. Technical report, W3C, June 2003.

[9] Rachid Hamadi and Boualem Benatallah. A petri net-based model for web service composition. In *Proceedings of the 14th Australasian Database Conference (ADC 2003)*, volume 17 of *CRPIT*. Australian Computer Society, November 2003.

[10] R. Heckel and H. Voigt. Towards consistency of web service architectures. In *Proc. of the 7th World Multiconference on Systemics, Cybernetics, and Informatics (SCI 2003)*, July 2003.

[11] C.B. Jones. Tentative steps towards a development method for interfering programs. *Transactions on Programming Languages and Systems*, 5(4), October 1983.

[12] Engin Kirda, Harald Gall, Pascal Fenkam, and Gerald Reif. MOTION: A Peer-to-Peer Platform for Mobile Teamwork Support. In *Cooperative Support for Distributed Software Engineering Processes Workshop, 26th COMPSAC Conference, Oxford, England*, pages 1115–1117. IEEE Computer Society Press, August 2002.

[13] T. Mikalsen, S. Tai, and I. Ravellou. Transactional attitudes: Reliable composition of autonomous web services. In *Workshop on Dependable Middleware Based Systems*, March 2002.

[14] Shin Nakajima. Model-checking verification for reliable web service. In *OOPSLA Workshop on Object-Oriented Web Services*, November 2002.

[15] Shin Nakajima. Verification of web service flows with model checking techniques. In *In Proceedings of International Symposium on Cyber Worlds: Theories and Practice*. IEEE Computer Society Press, November 2002.

[16] Leonor Prensa Nieto. The rely-guarantee method in isabelle/hol. In *Proceedings of ESOP 2003*, volume 2618, pages 348–362. Springer Verlag, 2003.

[17] Paulo F. Pires and Mario Benevides and Marta Mattoso. WEBTRANSACT: A Framework for Specifying and Coordinating Reliable Web Service Compositions. Technical Report ES-578/02, 2002.

[18] Gian Pietro Picco and Gianpaolo Cugola. PeerWare: Core Middleware Support for Peer-To-Peer and Mobile Systems. Technical report, Dipartimento di Electronica e Informazione, Politecnico di Milano, 2001.

[19] Nico Plat and Peter Gorm Larsen. An Overview of the ISO/VDM-SL Standard. In *ACM SIGPLAN Notices*, pages 76–82. ACM SIGPLAN, September 1992.

[20] Gerald Reif, Engin Kirda, Harald Gall, Gian Pietro Picco, Gianpaola Cugola, and Pascal Fenkam. A web-based peer-to-peer architecture for collaborative nomadic working. In

10th IEEE Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), Boston, MA, USA, pages 334–339. IEEE Computer Society Press, June 2001.

[21] Sheila A. McIlraith Srini Narayanan. Simulation, verification and automated composition of web services. In *Proceedings of the 11th International World Wide Web Conference (WWW 2002)*. ACM, November 2002.

[22] Ketil Stølen. *Development of Parallel Programs on Shared Data-Structures*. PhD thesis, Department of Computer Science, University of Manchester, 1990.

[23] F. Tartanoglu, V. Issarny, N. Levy, and A. Romanovsky. Dependability in the web service architecture. In *Proceedings of WADS 2002*, 2002.

[24] The MOTION Consortium. http://www.motion.softeco.it.

[25] The VDM Tool Group and The Institute of Applied Computer Science. *The IFAD VDM-SL Language*. IFAD, December 1996. Available from http://www.ifad.dk.

[26] Q. Xu and J. He. A theory of state-based parallel programming by refinement: part 1. In J. Morris and R. Shaw, editors, *Proceedings of the 4th BCS-FACS Refinement Workshop*, pages 326–359. Springer Verlag, 1991.