WSSecSpaces: a Secure Data-Driven Coordination Service for Web Services Applications*

Roberto Lucchi Gianluigi Zavattaro

Department of Computer Science
University of Bologna
Mura Anteo Zamboni, 7
40127 Bologna - Italy

{lucchi,zavattar}@cs.unibo.it

ABSTRACT

Web Services standards and protocols (WSDL, UDDI, SOAP, etc.) are the basis of a novel technology supporting Web based applications. Web Services are components offering ports at which service invocations can be sent using XML-based protocols. The tools currently proposed for specifying and programming the interdependencies among Web Services (BPEL, BizTalk, etc.) support the description of the flow of service invocation needed among collaborating Web Services in order to complete a specific task. In this paper we discuss the design and the implementation of a higher-level interaction model for Web Services that follows the tradition of data-driven coordination: Web Services do not coordinate via direct service invocation, but their interaction is mediated by a coordination space where shared data are stored and retrieved. Moreover, our proposal extends the traditional data-driven coordination model with a more sophisticated pattern matching mechanism that supports a controlled access to the shared data.

Keywords

Coordination Languages, Linda, Security, Web Services

1. INTRODUCTION

Web Services technology tends to provide standard mechanisms for describing the interface and the services available on the web, as well as protocols for locating such services and invoking them (see e.g. WSDL,UDDI,SOAP.More precisely, Web Services Description Languages (WSDL) is a format for describing network services as a set of endpoints operating on messages; Universal Description, Discovery and Integration (UDDI) serves as a discovery service for the WSDL descriptions; Simple Object Access Protocol (SOAP) is a simple and lightweight mechanism for creating the messages exchanged among the Web Services.

A relevant feature in the context of Web Services is the support for composing them in order to achieve complex tasks. For example, in business-to-business processes it is often the case to define new processes out of finer-grained subtasks that are likely available as Web Services.

The main effort in this respect is devoted to the definition of extensions of the Web Service standards, that support the definition of complex services out of simpler ones – the so called *Web Services choreography*. Several proposals that describe Web Services choreography have been already set up: BPML [1] by BPMI.org, XLANG [25] and BizTalk [9] (a visual specification environment for XLANG) by Microsoft, WSFL [17] by IBM, BPEL4WS [10] by a consortium grouping BEA, IBM, Microsoft, and others), etc.

These proposals permit to specify the interdependencies among collaborating Web Services via the indication of the flow of their relative invocations. In this paper we discuss the design and the implementation of a higher-level interaction model for Web Services that follows the tradition of data-driven coordination: Web Services do not collaborate via direct service invocation, but their interaction is mediated by a coordination space where shared data are stored and retrieved.

The main advantage following from the introduction of a coordination space is *loose coupling*. In standard web based applications we have, on the contrary, tight coupling in at least two ways. The first is that collaborating services must be up and running in order to complete their interaction. The second is that the interaction is programmed based on the interface of the services themselves; if the interfaces change, the application may break.

By loose coupling we mean that a Web Service can broadcast a request and then terminate. Some time later, a service may come online and may serve the request. This is said to be loose coupling in both "time and space": the interacting services do not need to be connected simultaneously and the application do not need to know the actual location of a service at the time the service is required. Moreover, since the communications do not occur directly but are mediated from the coordination space, the applications do not need to be programmed on the basis of the interfaces of the collaborating services. In this way, the modification/update of the interfaces does not influence the designed application.

While trying to introduce data-driven coordination in the Web Services architecture we have tried to be as much as possible compliant with the architecture itself: we have designed a Web Service, that we have called `WSSecSpaces`, which offers the coordination primitives as invocable services. In this way, for a Web Service is sufficient a direct connection to a `WSSecSpaces` in order to coordinate its activity with the other collaborating Web Services.

*Work partially supported by MEFISTO Progetto "Metodi Formali per la Sicurezza e il Tempo".

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SAC '04, March 14-17, 2004, Nicosia, Cyprus.
Copyright 2004 ACM 1-58113-812-1/03/04 ...$5.00.

The coordination facilities of WSSecSpaces are those offered by the generative communication paradigm [13]: a sender communicates with one or more receivers through a shared space where the emitted messages are collected; a receiver can read or consume the message from the space indicating with a pattern the kind of message he is interested in. This form of communication is referred to as generative communication because when a message is produced, it has an existence which is independent of its producer, and it is equally accessible to all components.

The main drawback of this approach, when applied to open applications such as those based on the Web, is that also untrusted and malicious components may access the coordination space. For this reason, we extend the coordination facilities of WSSecSpaces with a sophisticated pattern matching mechanism that permits to restrict the visibility and the accessibility, as well as verifying the producer, of a specific message inside the space. This extended pattern matching mechanism is borrowed from SecSpaces [6], a data-driven coordination model designed for supporting security properties also in the context of open applications.

The paper is structured as follows. In Section 2 we recall the SecSpaces coordination model [6]; in Section 3 we discuss the choices behind the implementation of WSSecSpaces; Section 4 discusses the related literature while Section 5 reports some conclusive remarks.

2. SECSPACES COORDINATION MODEL

SecSpaces [6] is a coordination model that supports secure data-driven coordination in open environments. SecSpaces extends Linda [13] by enabling control of access to the entries stored in the coordination space and to authenticate/identify the producer of an entry or its reader/consumer.

The coordination primitives of SecSpaces are the classical ones of Linda: $out(e)$, $in(t)$ and $rd(t)$. The output operator $out(e)$ inserts an entry e in the space. Primitive $in(t)$ is the blocking input operator: when an occurrence of an entry e matching with the template t is found in the space, it is removed and its content is returned. The read primitive $rd(t)$ is similar to $in(t)$, but in this case the entry e is not removed from the space.

SecSpaces implements access control mechanisms by extending Linda tuples with special control fields, namely *partition* and *asymmetric partition* fields. The former ones are used to logically partitionate the space, whilst the latter ones, as will be clear in the following, allow for discrimination between the write and the read/remove access permissions on each entry.

Let $Mess$, ranged over by m, n, ..., be an infinite set of messages, $Partition$, ranged over by c, c_t, ..., be the set of partitions and $APartition$, ranged over by k, k', k_t, ..., be the set of asymmetric partitions. We also assume that $Partition$ (resp. $APartition$) contains a special default value, say # (resp. ?), used to allow any agent to access the space. Let "$\overline{\cdot} : APartition \rightarrow APartition$" be a function such that $\overline{?} = ?$ and if $\overline{k} = k'$ then $\overline{k'} = k$.

Access control mechanisms are based on control fields. In order to discriminate between rd and in access permission, entries have two occurrences of control fields, one associated with in operations and the other one with the rd operations. Differently from entries, a template can be used by in and rd operations and it has only one occurrence of control fields that is dynamically associated to the operation the agent is performing.

The set $Entry$ of entries, ranged over by $e, e', \ldots$, is defined as follows:

$$e =< \vec{d} >^{[c]_{rd}[c']_{in}}_{[k]_{rd}[k']_{in}}$$

where $c, c' \in Partition$, $k, k' \in APartition$ and the tuple of data $\vec{d}$ is a term of the grammar $\vec{d} ::= d \mid d; \vec{d}$, $d ::= m \mid c \mid k$. A *data field* d can be a message, a partition or an asymmetric partition.

We define $\tilde{\cdot}$ as the function that, given an entry e, returns its tuple of data, i.e., if $e =< \vec{d} >^{[c]_{rd}[c']_{in}}_{[r]_{rd}[r']_{in}}$, $\tilde{e} = \vec{d}$.

The set $Template$ of templates, ranged over by t, t', ..., is defined as follows:

$$t =< \vec{dt} >^{[c_t]}_{[k_t]}$$

where $c_t \in Partition$, $k_t \in APartition$ and $\vec{dt}$ is a term of the grammar $\vec{dt} ::= dt \mid dt; \vec{dt}$, $dt ::= d \mid null$. With respect to entries, data fields used by templates can also be set to an additional value (*null*) that denotes the wildcard: the wildcard is used to match with all field values.

DEFINITION 2.1. *Let $e =< d_1; d_2; \ldots; d_n >^{[c]_{rd}[c']_{in}}_{[k]_{rd}[k']_{in}}$ be an entry, $t =< dt_1; dt_2; \ldots; dt_m >^{[c_t]}_{[k_t]}$ be a template and $op \in \{rd, in\}$ be an operation. Let c_e and k_e be the control fields of e associated to op, we say that e $matches_{op}$ t if the following conditions hold:*

1. $m = n$
2. $dt_i = d_i$ *or* $dt_i = null$, $1 \leq i \leq n$
3. $c_e = c_t$
4. $\overline{k_e} = k_t$.

If a rd or in operation with template t is performed on a matching entry e, only the data fields (and no control field) are returned, i.e., the return value is $\tilde{e}$.

Condition 1. and 2. rephrase the classical Linda matching rule, that is test if e and t have the same arity and if each data field of e is equal to the corresponding field of t or if this latter one is set to wildcard. Condition 3. tests that the partition field of the entry –associated to the operation *op*– is equal to that of the template. Condition 4. checks that the asymmetric partition field of the template corresponds to the co-key of the asymmetric partition field of the entry associated to the operation *op*.

As shown in the matching rule, partition fields are a special kind of data fields that do not accept wildcard in the matching evaluation. Partition fields logically partition the space and the access to a partition is restricted to only those processes that know the partition identifier. Indeed, in order to perform an operation on a partition, processes must know the partition field identifying that specific partition. However, in order to allow processes to interact with each other via SecSpaces primitives, a special default value # of the partition field –known by any agent– has been defined. Similarly, a default value known by any process has also been defined for asymmetric partition fields (denoted by "?").

Differently from partitions, the asymmetric partition fields make it possible to discriminate between the permission of write and read/remove of an entry. This is explained in the following Example.

EXAMPLE 2.2. *As explained in detail in [6], by exploiting asymmetric partition fields it is possible to implement* one-to-many, many-to-many and many-to-one *communication. In this example we explain how to implement one-to-many communication guaranteeing the authentication of the producer: the idea is that we use an asymmetric partition k -known only by the producer- in the asymmetric partition fields of each entry used in the communication. In this way, consumers/readers can certify the identity of*

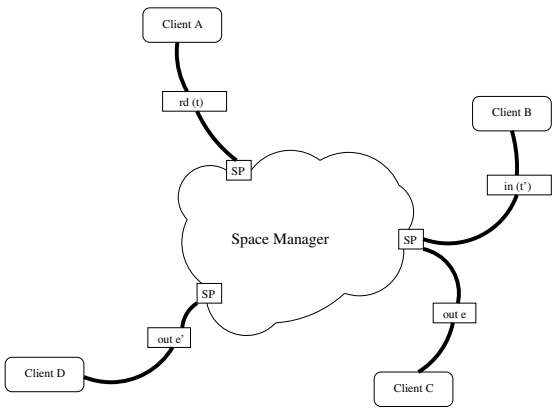

Figure 1: `SecSpaces` architecture

the producer by setting the asymmetric partition field of the template to $\overline{k}$, that we assume to be known by any process allowed to read/remove such data. To write a datum d, the producer performs an $out(< d >^{[c]_{rd}[c]_{in}}_{[k]_{rd}[k]_{in}})$ operation, whilst to read a datum a process performs a $rd(< null >^{[c]}_{[\overline{k}]})$ operation. It is easy to observe that only the producer can perform the output because it is the only one that knows k. Finally, this example can be furtherly extended for supporting different groups of readers and consumers simply by using two distinct asymmetric partition values for rd and in operators.

Finally, the return value of read/input operations does not include the control fields; it contains only the data stored inside the tuple of data fields. Therefore, new access permissions can be acquired only by performing read/input operations of entries containing partition or asymmetric partition values inside the tuple of data.

3. THE WSSECSPACES WEB SERVICE

`WSSecSpaces` is a Web service that implements the coordination primitives of `SecSpaces`. To show that `WSSecSpaces` provides a secure implementation of `SecSpaces` we analyse security issues at the implementation level and then we discuss the adopted solutions. Figure 1 describes the architecture of systems composed by processes interacting via `SecSpaces` primitives and by the shared space. The elements represented are Client, Space-Plug (SP for short) and the Space Manager. Space-Plugs provide clients an interface to interact with the shared space, that use the communication channels client-SP to invoke coordination primitives and to receive their return values. One of the most important entry points for an attack are the communication channels. The risk is raised because enemies can eavesdrop or manipulate the exchanged data along the path client-SP, changing the expected behavior. Furthermore, an attacker SPE (*Space-Plug Enemy*) can replace a SP in the interaction with the clients. If this happens, clients read and write information from and to a fraudulent server SPE (that acquires all transmitted data) believing to interact with a trusted SP.

The security issues concerning communication on channels, such as entity authentication, privacy and integrity of data, are well know problems and several cryptographic protocol [22] have been proposed to solve them. In order to avoid such attacks, our implementation exploits the HTTPS protocol between clients (e.g., other Web services) and `WSSecSpaces`. Finally, as will be explained in detail in Section 3.3, `WSSecSpaces` provides just one access port that can be used to perform output, read and removal operations on the space. In other words, our proposal has only one space plug that is described by the WSDL (Web Services Description Language)document associated with `WSSecSpaces`. WSDL documents are usually published on a UDDI server; the problem of obtaining a trusted reference at the document is left to the interaction phase between clients and the UDDI server.

3.1 Implementation of control fields

In this section we tackle with particular care the problem of implementing control fields modeled in the previous section. The implementation of partition fields does not involve relevant problems; indeed, the only assumption is that a process should not have any way to guess an unknown partition field used by other processes. Similarly to symmetric encryption keys (see e.g. [22]), we need to implement the set $Partition$ so that to guess one of its values has low probability that can be realized, e.g., by encoding partitions with data composed by 512 bits.

The implementation of asymmetric partition fields is more sophisticated because the function $\overline{\cdot}$ must guarantee that: i) the coordination service (that is the implementation of the shared space) should be able to check whether two fields k and k' match (i.e. verify if $k' = \overline{k}$); ii) it should be unfeasible for an agent to guess $\overline{k}$ starting from the knowledge of k.

The implementation we use in this paper for asymmetric partition fields exploits asymmetric cryptography. Formally, let *PlainText*, ranged over by $p, p', \ldots$, be the set of plaintexts, *Key*, ranged over by $PrivK$, $PubK$, $\ldots$, be the set of encryption keys containing private and public keys. In the following, when we refer to pairs of private and public keys $(PrivK, PubK)$, we assume that a plaintext encrypted with $PubK$ (resp. $PrivK$) can be decrypted only by using $PrivK$ (resp. $PubK$). Let $Ciphertext$, ranged over by s, $s_t, \ldots$, be the set of ciphertexts obtained by encrypting plaintexts with encryption keys (we denote with $\{p\}_k$ the encryption of p with key k).

Each asymmetric partition field, except the default value ? encoded with $?'$, is encoded by a triple $(p, PubK, s)$. Assuming a perfect version of cryptographic operations, as usually done when cryptographic protocols are analysed (see e.g. [11]), the following implementation of $\overline{\cdot}$ satisfies the requirements of asymmetric partition fields (for more details, see [6]):

- given $?'$, we have that $\overline{?'} = ?'$;
- given the triple $(p, PubK, s)$, we have that $\overline{(p, PubK, s)} = (p', PubK', s')$ if $s = \{p'\}_{PrivK'}$ and $s' = \{p\}_{PrivK}$.

3.2 Entry, template and XML-based matching rule

The SOAP protocol–used to perform Web services invocations– as well as most of the data involved in the Web Services technology are based on XML: this simplifies and enables interoperability between applications running on different architectures [4]. In order to be consistent with these goals, `WSSecSpaces` provides a definition of entry and template based on XML. In the following we describe how we have defined the basic elements of `SecSpaces`, i.e. entries, templates and the matching rule in `WSSecSpaces`.

Entries and templates structure [18] have been defined by exploiting XML-Schema.An entry is described by an element entry which contains three elements: rd, in and datafield. The rd and in elements represent the control fields associated with rd and in operations, respectively. Their content is composed by two elements: apartition and partition. The former represents the asymmetric partition field and –following the implementation of Section 3.1– contains three elements: plaintext, key and ciphertext that contain a plaintext (of type "string"), a cryptographic key (of type "string")

and a ciphertext (of type "binary"), respectively. The partition field is the content of partition (of type "string"). The third element of entry is dataField and contains the sequence of data fields. More precisely, it contains an unbounded sequence of field elements that can contain any XML type.

Differently from entries, templates contain only one occurrence of control fields and their data fields can also be set to wildcard. Templates are represented by the element template that contains three elements: apartition, partition (which have the same meaning and structure of the homonymous elements contained in entry) and tdataField. The latter one contains an unbounded sequence composed by field, wfield and wnull elements. The elements field have the same meaning of the homonymous ones contained in entry and the matching between two field elements holds if they have the same content. wfield and wnull are used to implement wildcards. Similarly to recent object oriented Java coordination middlewares such as TSpaces [28] and JavaSpaces [12] that allow the use of complex classes as typed wildcards, `WSSecSpaces` improves the matching rule by exploiting XML technologies. More precisely, in `WSSecSpaces` the elements wfield are used to specify an XML-Schema: an element field matches with an element wfield if its content can be validated with the schema contained in the latter element (i.e., it is conformant with the structure defined in the wildcard schema). Finally, the element wnull (whose content is empty) provides a more general wildcard implementing the *null* defined in the model: any element field matches with an element wnull. We have been forced to define an extra element wnull because, using XML-schema, it is not possible to define a top type, that is an XML-schema such that any document results to be valid with respect to such schema.

The matching rule of `WSSecSpaces` differs from the formal definition (see Definition 2.1) in the evaluation of the matching between data fields: they match if each element field of the entry matches with the corresponding element (field, wfield or wnull) of the template. More formally, let $WField$, ranged over by wf, wf', ..., be the set of the possible contents of wfield elements. We model the content of field elements with the data fields (d) defined in Section 2, whilst now the content of the fields used by templates is defined as follows: $dt ::= d \mid wf \mid wnull$. Let $\triangleright_{XML}$ be a relation between data fields and wildcard fields: we say that $d \triangleright_{XML} wf$ if d can be validated using the schema wf.

DEFINITION 3.1. *Let* $e =< d_1; d_2; \ldots; d_n >^{[c]_{rd}[c']_{in}}_{[k]_{rd}[k']_{in}}$ *be an entry,* $t =< dt_1; dt_2; \ldots; dt_m >^{[c_t]}_{[k_t]}$ *be a template and* $op \in \{rd, in\}$ *be an operation. We say that* e $matches_{op}$ t *if conditions 1, 3 and 4 of Definition 2.1 and the following condition hold:*

2'. $dt_i = wnull$ *or* $(d_i \triangleright_{XML} dt_i$ *if* $dt_i \in WField)$ *or* $d_i = dt_i$, $1 \leq i \leq n$.

It is easy to observe that this rule is a conservative extension of the original matching rule. Furthermore, the access control mechanisms of `WSSecSpaces` have the same properties of `SecSpaces` because they are based on control fields; therefore the matching rule on data fields does not affect these properties (for more details, see [6, 5]).

It is worth noting that the matching rule can be implemented using standard tools for XML processing. Moreover, in the case of typed wildcards the termination of the validation process is ensured. This is an important aspect because this ensures that no unexpected exceptions or non–terminating evaluations of the matching rule can be forced by malicious clients of `WSSecSpaces`.

Finally, the schema describes the structure of the return values of the operations (ret); it can be composed only of either a sequence of data fields (datafield) or of the element noValue that denotes the absence of a return value; the former is used for rd/in operations whilst the latter for out ones.

3.3 WSDL and implementation

The Web Services Description Language (WSDL) is a XML-based language used to describe and publish information concerning the service, such as the format of the exchanged information and the modalities a client may use to interact with the service.

In this section we comment only the significant design aspects of the WSDL document associated to `WSSecSpaces`, for the remaining details interested readers can find the complete WSDL source of `WSSecSpaces` in [18]. Since all coordination primitives provide a return value the interaction is always of kind *send-response*; therefore we have opted for a solution with only one access port –whose location address exploits the HTTPS protocol– where the operation the client is willing to perform is described in the content of the SOAP documents used for service invocations.

From the point of view of the internal implementation, the core of `WSSecSpaces` is a Java [15] Servlet that has been tested by using Tomcatas servlets container. Finally, the implementation exploits JAXP [24] for Java XML-processing operations and the standard security Java package for public key encryption algorithm; more precisely, the actual implementation exploits the DSA algorithm.

4. RELATED WORK

In the last years we assisted to a renewed interest on Linda-like coordination languages from both implementation and theoretical point of views. In particular, three main aspects have attracted the attention of the coordination community: i) security in coordination; ii) integration of Linda with Web technologies (e.g., XML); iii) coordination of Web Services. `WSSecSpaces` tries to meet all these aspects.

`SecSpaces` extends Linda for supporting secure coordination and can be compared with other proposals; the most interesting are Klaim[20] and SecOS [27]. The former exploits a standard access control mechanism where permissions describe which operations the agents can perform on the available spaces. In its original version Klaim does not allow dynamic permission acquisition, this is supported by MuKlaim [14] that is a sort of core for Klaim because implements its basic features. Further, we consider that the classical solutions of Klaim is not particularly suitable for environments where the system configuration changes frequently. The latter is a capability-based system and `SecSpaces` can be viewed as an extension of this model. `SecSpaces` refines the access permissions, indeed in SecOS the rd and in operations have the same access permission; moreover, the dynamic privileges acquisition in `SecSpaces` happens only if an agent reads an entry that contains a control field value in the data fields whilst in SecOS, for instance, an agent can acquire the right permission to perform the output of any entry it can read. Other proposals are Klava [3] and a secure version of Lime [16]. The former introduces encrypted messages into the fields of the tuples and the matching rule allows the evaluation of messages encrypted into fields; the encryption of messages ensures that they can be read only by the allowed clients. The latter presents an implementation of Lime [19] that provides a password based access control mechanism at the level of entries and tuple spaces. Finally, [21] is a general model for coordination middlewares which exploits process handlers to control the behaviour of processes; a language derived from CCS is used to describe which operations are allowed.

Considering the integration of XML in Linda-like platforms, one former proposal is XMLSpaces [26], a coordination middleware –

based on TSpaces– that exploits XML: entries and templates are XML documents and XML-query, as well as DTD (Document Type Definition), are used in the matching rule. For a complete survey of XML-based coordination infrastructures see [7].

The third category we consider is Linda and Web Services and, to the best of our knowledge, the most interesting proposals are [2] and Ruple [23]. The former is an implementation of a Web service –based on JavaSpaces– that allows for a location-based coordination of Web services (i.e. different locations support different coordination spaces). The latter is an implementation of a coordination infrastructure for Web services and it is the only related work that supports both security and XML-technologies; it provides an access control mechanism based on digital certificates that does not discriminate between the destructive and non-destructive read operations. Further, the use of digital certificates bounds the possible readers/consumers of a datum at creation time; in this way it is not supported a dynamic privileges acquisition on the data that are already available in the repository.

5. CONCLUSION AND FUTURE WORK

In this paper we have discussed the design and the implementation of `WSSecSpaces`, a data-driven coordination service to be exploited in Web Services applications. As already discussed in the Introduction, loose coupling among Web Services is one of the main advantages introduced by `WSSecSpaces`. In other terms, `WSSecSpaces` represents in the setting of Web Services what JavaSpaces [12] is for Jinitechnology. Moreover, `WSSec-Spaces` supports a more sophisticated pattern matching mechanism that permits to control the access to the data available in the coordination space, as well as the authentication of the producer of a specific datum.

Actually, the security of data transmitted on channel is preserved by HTTPS transfer protocols used in the SOAP service invocation protocol, the study of alternatives based on other technologies is left as future work. For instance, there are some other interesting technologies such as XML-encryption [8] that can be coherently combined with the Web Services systems and `SecSpaces`. Further, we intend to investigate on the definition of a Web Services coordination infrastructure that takes into account quality of service requirements, e.g. based on the configuration/topology of the system.

6. REFERENCES

[1] Business Process Modeling Language (BPML). [www.bpmi.org].

[2] P. Álvarez, J.A. Bañares, P.R. Muro-Medrano, F.J. Nogueras, and F.J.Zarazaga. A Java Coordination Tool for Web-service Architectures: The Location-Based Service Context. In *Scientific Engineering for Distributed Java Applications*, volume 2604 of *LNCS*, pages 1–14. Springer-Verlag, 2003.

[3] L. Bettini and R. De Nicola. A Java Middleware for Guaranteeing Privacy of Distributed Tuple Spaces. In *Proc. of FIDJI'02, Int. Workshop on scientific engineering of distributed Java applications*, 2002. To appear in LNCS.

[4] Jon Bosak. XML, Java and the future of the Web. *World Wide Web Journal*, 2(4):219–227, 1997.

[5] Mario Bravetti, Roberto Gorrieri, and Roberto Lucchi. A formal approach for checking security properties in SecSpaces. In *1st International Workshop on on Security Issues in Coordination Models, Languages and Systems*, volume 85.3 of *ENTCS*, 2003.

[6] Nadia Busi, Roberto Gorrieri, Roberto Lucchi, and Gianluigi Zavattaro. Secspaces: a data-driven coordination model for environments open to untrusted agents. In *1st International Workshop on Foundations of Coordination Languages and Software Architectures*, volume 68.3 of *ENTCS*, 2002.

[7] P. Ciancarini, R. Tolksdorf, and F. Zambonelli. Coordination Middleware for XML-centric Applications. In *Proc. ACM/SIGAPP Symp. on Applied Computing (SAC)*. ACM Press, 2002.

[8] World Wide Web Consortium. XML Encryption Syntax and Processing. W3C Recommendation 10 December 2002. http://www.w3.org/TR/xmlenc-core/.

[9] Microsoft Corporation. Microsoft BizTalk Server. [http://www.microsoft.com/biztalk/default.asp].

[10] F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. Business Process Execution Language for Web Services (BPEL4WS 1.0). [http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/], 2002.

[11] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Trans. on Information Theory*, 29(2):198–208, 1983.

[12] Freeman, Hupfer, and Arnold. *JavaSpaces Principles, Pattern, and Practice*. Addison-Wesley, 1999. first edition.

[13] D. Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.

[14] Daniele Gorla and Rosario Pugliese. Resource Access and Mobility Control with Dynamic Privileges Acquisition. In *Proc. of the Int. Colloquium on Automata, Languages and Programming (ICALP'03)*, LNCS, 2003. To appear.

[15] James Gosling, Bill Joy, and Guy Steele. *The Java Language Specification*. Addison-Wesley, 1996.

[16] Radu Handorean and Gruia-Catalin Roman. Secure Sharing of Tuple Spaces in Ad Hoc Settings. In *1st International Workshop on on Security Issues in Coordination Models, Languages and Systems*, volume 85.3 of *ENTCS*, 2003.

[17] F. Leymann. Web Services Flow Language (WSFL 1.0). [http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf], Member IBM Academy of Technology, IBM Software Group, 2001.

[18] R. Lucchi and G. Zavattaro. WSSecSpaces: a Secure Data-Driven Coordination Service for Web Services Applications, Technical Report UBLCS-2003-11, University of Bologna (Italy), 2003. [http://www.cs.unibo.it/techreports/].

[19] A. Murphy, G. Picco, and G.-C. Roman. A middleware for physical and logical mobility. In *21st International Conference on Distributed Computing Systems*, pages 524–533, 2001.

[20] R. De Nicola, G. Ferrari, and R. Pugliese. KLAIM: A Kernel Language for Agents Interaction and Mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330, May 1998. Special Issue: Mobility and Network Aware Computing.

[21] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Formal Specification and Enactment of Security Policies through Agent Coorination Contexts. In *1st International Workshop on on Security Issues in Coordination Models, Languages and Systems*, volume 85.3 of *ENTCS*, 2003.

[22] B. Schneier. *Applied Cryptography*. Wiley, 1996.

[23] Rogue Wave Software. Ruple. http://www.roguewave.com/developer/tac/ruple/.

[24] Sun Microsystems, Inc. *Java API for XML Processing*. http://java.sun.com/xml/jaxp/index.html.

[25] S. Thatte. XLANG: Web Services for Business Process Design. [http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm], Microsoft Corporation, 2001.

[26] Robert Tolksdorf and Dirk Glaubitz. XMLSpaces for Coordination in Web-based Systems. In *Proc. of the Tenth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises WET ICE*. IEEE Computer Society, Press, 2001.

[27] Jan Vitek, Ciarán Bryce, and Manuel Oriol. Coordinating Processes with Secure Spaces. *Science of Computer Programming*, 46:163–193, 2003.

[28] P. Wyckoff, S.W. McLaughry, and D.A. Ford. TSpaces. *IBM System Journal*, August 1998.