Managing Trust in Active XML

Etienne Canaud, Salima Benbernou, Mohand-Saïd Hacid
LIRIS - Lyon Research Center for Images and Intelligent Information Systems
Lyon 1 University
Villeurbanne, FRANCE
{*etienne.canaud, salima.benbernou, mohand-said.hacid*}*@liris.cnrs.fr*

Abstract

Active XML [19, 3] combines XML Data and service calls to allow a simple and powerful Web services implementation. Security in Active XML is currently handled by matching the structure of the received data with an XML Schema representing the allowed data (including service calls). This solution is not fully satisfactory in case of an open environment where the services do not often know or trust each other. Moreover, the strength of Active XML lies in its simple and dynamic structure, and the modified XML Schemas used for security matching can quickly limit the allowed services, or give too much freedom to services that should not be trusted. Given that the result of an Active XML service call is some Active XML data (that may include more service calls), Active XML data is recursive, thus involving more security concerns.
We propose a new framework based on the notion of Trust (Trusted Active XML) for handling security in Active XML. In this framework, "trusted" services'answers are not restricted to a specific data schema, while "untrusted" ones are prevented from performing some unwanted operations.

1. Introduction

The fast development of Web services during the last few years gave birth to numerous industrial standards amongst which some are now widely used, like WSDL [25] (services description), UDDI [21] (services repository and discovery), and BPEL4WS [10] (services workflow composition). Some other standards dedicated to Web services have also been proposed or are still under development, such as DAML-S [9, 20] (semantic services description) or XL [12] (services implementation and composition).
Active XML [19, 2] (AXML for short) proposes a simple framework for Web services execution: service calls are embedded into XML Data, thus making the execution dynamic and allowing to express intentional data [15] that can be turned into extensional data if needed. Figure 1 gives an example of AXML data, where the syntax is lighten for the sake of readability. Services calls have been simplified and are embedded using `<sc></sc>` tags.

One of the major issues in Active XML is to choose when to execute or not the service calls: when is it more appropriate to make the intentional data extensional ? This topic is discussed in [19, 1], and raises some interesting security concerns: an Active XML service will usually return some Active XML data, so the answer can contain some other service calls. Instantiation of intentional data is then a recursive process. Malicious services could be embedded into answers and might induce unwanted actions or return malevolent data if these services are called. An example of such a problem is depicted in figure 2, where a unreliable weather service will return a malicious answer that could lead to endanger persons or goods if used in a critical context.

Active XML security is based on typing [15]. One can check whether the structure of the data sent by a service matches some specific predicates, and then avoid to use the returned data or call the embedded services. Interesting investigations on the matching issue are still conducted, mainly when adding the service calls in the data structure. However, the main drawback with the proposed approaches resides in the way a service can decide to accept or ignore calls to other services.
For example, when ordering some books online, Bob might be reluctant to order on a Web site he doesn't trust. If Alice (a good friend of Bob) tells him that he can trust the "BuyBooks.com" Web site, he may be willing to use it and make use of his credit card number for the payment. Assume now that Bob is used to buy books on Amazon.com and trusts this Web site. If the book Bob wants to buy is not available on Amazon.com but the Web site proposes to Bob to buy the book on "BuyBooks.com", (while most of the operations will be conducted on the Amazon Web site), there is a chance that Bob will accept this deal. Bob can choose to trust or not a service that is proposed to him by another ser-

Initial Active XML Data about a city population.

```
<cityPop>
 <city>
  <sc>CountryInfo.com/GetCapital(
  <country>"P.R.China"<country>)
  </sc>
 </city>
 <population>
  <sc>CityStats.com/GetPopulation(
    <city>
     <sc>CountryInfo.com/GetCapital(
       <country>"P.R.China"<country>)
     </sc>
    </city>)
  </sc>
 </population>
</cityPop>
```

After the call of GetCapital( ), we will get:

```
<cityPop>
 <city>
  Beijing
 </city>
 <population>
  <sc>CityStats.com/GetPopulation(
    <city>Beijing</city>)
  </sc>
 </population>
</cityPop>
```

After the call of GetPopulation( ), we get:

```
<cityPop>
 <city>
  Beijing
 </city>
 <population>
   14.000.000
 </population>
</cityPop>
```

Figure 1. Block of Active XML and its answer

vice that he may already trust (or not). When using a typing technique, it may be difficult to express these trust relationships. The reason is that Active XML must be able to operate in an open environment, where peers do not often know each other, directly or not (i.e. accessed directly or through the answer of another middle service). Using only typing could quickly lead to a situation where each peer would have either a list of services it will always accept to deal with (white list), or a list of services that it definitely refuses to deal with (black list).

AXML Data that will return the weather on Miami coast during the afternoon:

```
<weather>
 <sc>MiamiBoatRenting.com/GetWeather(
   <day>
     Today
   </day>
   <timeOfTheDay>
      Afternoon
   </timeOfTheDay>)
 </sc>
</weather>
```

Despite the fact the the service knows that heavy rain is predicted for the afternoon, the call of GetWeather( ) returns:

```
<weather>
   <conditions>
      Light wind, Cloudy,
      Maybe few raindrops
   </conditions>
</weather>
```

Figure 2. Example of a malicious Active XML service

The book buying example is relatively simple, but one can think of a more complex scenario where a travel agency would propose a flight with a night in a hotel instead of the night train ticket asked by the client: the structure of the proposed answer may differ considerably from the solution accepted by the typing method, and the answer could be rejected even if the proposed offer is better for the peer invoking the service. Moreover, the very interesting fact that services can include other services in their answers (with no depth limit) can be difficult to represent exhaustively using the modified XML Schemas proposed in [15]. The typing and its implementation described in that work emphasize on the function names, but do not display the importance of the Web sites they originate from (represented by their end point URL). This means that a function ("`GetNow( )`" for example) will be evaluated in the same way if it is issued by a trusted provider (Amazon.com) or an untrusted one (BuyBooks.com).

Our proposition is to handle service providers names and use them to establish a trust relationship between services that will give more freedom to trusted services, and then allow them to give answers that can differ from the authorized templates. Both the user and the service provider will take advantage of this approach, and the untrusted providers will still be limited in the answers they can provide if they don't benefit from the trust of another "trusted" provider.

The rest of the paper is organized as follows: Section 2 gives an overview of proposed approaches for managing security and trust in Web Services. Section 3 presents our approach for improving the flexibility of the security management in Active XML. Section 4 discusses our approach. We conclude in Section 5.

2. Related Works

2.1. Web Services Security

The Industry view on Web services security [18, 23] is mostly focused on concerns such as data integrity and confidentiality, authentication, and non repudiation of messages. These issues are considered by adapting general information security technologies (such as cryptography or digital signature) to XML data. The advantage is that these technologies have been extensively tested and improved for many years and that they are still a topic of concern in the research community. Some of the most significant specifications in XML services security are [16]:

- *XML Encryption* (XML Enc): it describes how to encode an XML document or some parts of it, so that its confidentiality can be preserved. The document's encoding procedure is usually included in the file, so that a peer possessing the required secrets can find the way to decrypt it.
- *XML Digital Signature* (XML DSig): it describes how to attach a digital signature to some XML Data. This will ensure data integrity and non repudiation.
- *Web Services Security* (WSS): this standard is based on SOAP, XML Enc and XML DSig and describes a procedure to exchange XML data between Web services in a secure way.
- *Security Assertion Markup Language* (SAML): it specifies how to exchange (using XML) authentication and authorization information about users or entities. Two services can use SAML to share authentication data in order not to ask a client to log again when it changes from one service to another (Single Sign On procedure).

Considering that Active XML is a language that is certified pure XML, all these recommendations can be used to ensure its security during the transfer and the storage of information.

2.2. Typing and Pattern Matching

The problem we are concerned with in this paper occurs during evaluation of AXML service calls. We need to control the service calls in order to avoid those that could execute malicious actions (for example, to buy a house or to sell a car on eBay at a tiny price).
As explained previously, Active XML relies on a typing and function pattern matching algorithm that will compare the structure of the answer returned by the service with a "allowed structure" provided by the client. If the structures can match (using rewriting), then returned calls can be invoked. Details on the algorithm are given in [15]. This algorithm is k-depth limited, and its decidability remains an open problem when ignoring the k-depth limit.
CDuce [5] is an example of language with powerful pattern matching features. It can easily compare structures and corresponding data, and its strong capability to handle types (and subtyping) allows to define structures precisely. But CDuce is mostly oriented towards XML transformation, whereas AXML is definitely more simple and adapted for Web services.

2.3. Trust in Web Services

There are many approaches to consider the notion of "trust" in services. The most adopted vision of trust in services is based upon progressive requests and disclosures of credentials between the peers (according to a policy), that will gradually establish the trust relationship [4, 22]. The privacy of the peers can be preserved, and credentials do not have to be shown without a need for them, thus preventing the user from displaying some information that (s)he could want to keep away from a non authorized peer [13].
This rather technical representation of trust can be used in our work, but there are some other ways to consider it. For example, in [11], the analysis of trust is based upon the basic beliefs that will lead to the decision of granting trust or not. This approach is much more sociological and context dependent from the previous one, but it relies on the way a human being behaves when trusting or not another person. The conditions required to make the final decision of trust granting are divided into two major parts:

- *internal attribution*, representing the conditions that depend on the trusting agent's personality and skills,
- *external attribution*, that represents conditions that are completely independent from the agent (opportunity, interferences, ...).

Depending on these factors, a value representing the "trustfulness" is computed using a fuzzy algorithm. This value will allow the agent to take the decision and then to trust or not to trust the peer .
A way of establishing trust between peers with no prior relation between them is presented in [8, 7]. It relies on an ordering of trust relationship between the peers and a local trust server, and peers can exchange messages by following a path of trust servers where each server sending the message has to trust the server that will receive and relay it un-

til it reaches its destination. The notion of trust they use is not a quantitative one, even if in [7] trust is represented by a rational number between 0 and 1. The only goal of such a representation is to define an ordering between trust values, and such trust levels cannot be added or divided.

3. Handling Trust in Active XML

3.1. Trust Representation

We will first define the way we consider the notion of trust in Web services. Our definition of trust is thus based on the one given in [24]:
Trust is the characteristic that one entity is willing to rely upon a second entity to execute a set of actions and/or to make a set of assertions about a set of subjects and/or scopes.
However, we will enrich that definition by considering that a relation of trust happens in a specific context, and that trust is not only a binary characteristic but can have different levels of strength. The possibly subjective nature of this representation is extensively discussed in sections 4 and 5 .

The proposed trust representation is based on the "trust level" of a service. It represents the amount of trust that one can have in an entity providing this service. A first approach we choose for expressing the trust consists in assigning a single float numerical value τ such as $\tau \in [0, 1]$. The value 0 (default value) means that one does not trust the service provider at all, while the value 1 means that one has a complete trust in the service provider.
The way the trust level is computed is out of the scope of this paper. The reason is that it is highly context-dependent. For example, a large company can design a process that will assign a trust level according to a credential disclosure policy with the service provider (the more credentials disclosed, the higher trust level will be), and a simple citizen could assign trust levels to his favorite online commerce Web sites according to his personal feeling. Anyone can design a way to initialize this trust level that will fit his vision of trust and his needs [14].

3.2. Services Evaluation

An example of service evaluation tree is given in figure 3. The lines represent service calls, while the nodes ($\triangle_x$, with $x \geq 0$) stand for a service provider. A node thus defines the *entity* that the client chooses to trust. So, even if it is the service that calls the other services, the trust is granted to the *entity* providing the service.

In [15], only the name of the function used to call the service is handled. However, in this work we emphasize on the provider, because trust is usually oriented towards the entity providing the service rather than towards the service itself. In the following, the term *"entity"* will refer to a provider. We made a first simplifying assumption which consists in giving trust to service providers only, but our future work will handle service specific trust settings.

Let's consider the case where a client invokes a first service $\triangle_0$ (named *entry service* in the following). The *entry service* $\triangle_0$ will send back an answer that can contain calls to other services $\triangle_i$, with $i \geq 0$. All these services $\triangle_i$ can themselves include calls to some $\triangle_i$ in their answers, and so on. Even if a function can recursively call itself, we will consider the evaluation of this *entry service* as a tree, and not as a graph, as depicted in figure 3. The main reason for this choice is that we consider the service calls as being independent from each other, and then a service can be called many times in the same tree and be granted a different amount of trust every time.

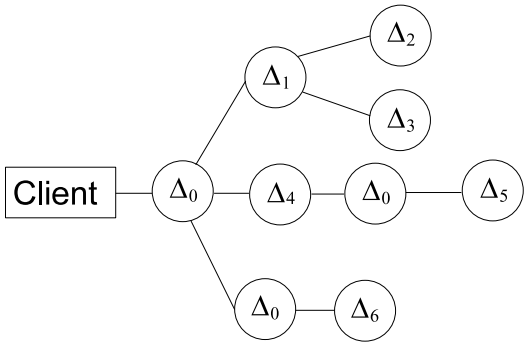

Figure 3. Service Calls Tree for Service Evaluation

Typing pattern matching (also called *rewriting*) will consist in invoking services needed to let the answer match the given schema of accepted structures. As pointed out by the authors of [15], an acknowledgement of the user is needed during rewriting before calling a service if it might have side effects or some cost. But there is a security concern, where malicious services could be called without asking the user, in case the service provider has been compromised or the side effects of the service call are not fully known.

Our approach is to invoke all the *trusted* services except those that have already been chosen not to be invoked by

the user (more details on the "to call or not to call" issue can be found in [19]). Our algorithm depends on some settings that the user will have to choose, depending on the context:

- *Trust Threshold* (Θ): This is the lower limit of trust level τ that the service must have in order to be considered "secured". All the service calls issued by a service $\triangle_i$ with $\tau_i \geq \Theta$ will be considered to be secure and can therefore be invoked without further security check. We state that $\Theta \in [0, 1]$.
- *Trust Transmission Function* ($\Psi(\tau_i)$): This function describes how a service $\triangle_j$ will benefit from the trust given by the user to the service $\triangle_i$. $\triangle_i$ is the parent service of $\triangle_j$. This means that a call to $\triangle_j$ will be inserted in the answer to $\triangle_i$. This function $\Psi()$ takes τ_i as input and produces τ_j as output. We state that $\tau_i \geq \tau_j$. The exact specification of this function is to be done by the user. Additional information about the theory of trust update functions can be found in [14].

These *Trust Threshold* and *Trust Transmission Function* have to be defined according to the *context* C. A context is defined as a particular set of facts or circumstances that surround a service call. For example, calling a service returning the weather will not require the same trust settings in the context of a city tourism trip or in the context of a mountain climbing expedition. The user must then set the *Trust Threshold* and *Trust Transmission Function* for all the critical contexts (where money is involved for example), but can rely on default values for service calls where there is no particular context. A system must have a default context trust settings, but can have as many specific context trust settings as needed (possibly none).
As a first approach, we chose to keep the same context during all the service calls corresponding to a same entry service call.

The algorithm for the evaluation of services is given figure 4. For the sake of optimization, we will consider a depth-first approach for traversing the tree.

3.3. Example

Figure 5 gives an example. It represents a scenario where Bob wants to get some information on a book. To get the information, he makes use of a service located on `bookInfo.com`. His system is set up with the following parameters, which correspond to the default context:

- *Trust Threshold* (Θ): 0.5
- *Trust Transmission Function* $\tau_j = \Psi(\tau_i) = 2/3 \times \tau_i$
- *Trust level* τ for web site `bookInfo.com` : 0.9
- *Trust level* τ for web site `amazon.com`: 1

Nota: the *rewriting method* we refer to in this algorithm is the one described in [15]. Whether to use the rewriting method, negotiate trust or stop the evaluation process depends on the user preferences. $\triangle_k$ refers to services called by $\triangle_j$. The values of $\Psi()$ and θ are the ones defined for the context $c_l \in C$ of the entry service call.

```
For a service △j in a context cl
   IF ∃τj
   THEN IF τj ≥ θ
        THEN needed service calls
             △k are evaluated
        ELSE use rewriting method
             or stop here
             or negotiate trust
        ENDIF
   ELSE IF △j is not the entry service
        THEN IF Ψ(τi) ≥ θ
             (△i being the service calling △j)
             THEN needed service calls
                  △k are evaluated
             ELSE use rewriting method
                  or stop here
                  or negotiate trust
        ELSE use rewriting method
             or stop here
             or negotiate trust
```

Figure 4. Services Evaluation Algorithm

- *Trust level* depends on providers (web sites), not on services.
- Ignore services whose *trust level* is smaller than the *trust threshold* Θ.

Since the *trust level* τ of `bookInfo.com` (0.9) is greater than the *trust threshold* Θ (0.5), the service calls included in its answer will be considered. These calls are inserted among some XML data related to the book information.
The first of the two calls included in the answer will return the price of the book if available on `Amazon.com`. As `Amazon.com` has a trust level of 1, this service will be invoked and the answer will be retrieved.
The second call from the first answer consists in getting some offers for buying the book through an auction system (provided by the Web site `auctions.com`). Since this Web site is not yet registered in our "trusted entities" list, we will first compute its trust level for this session. The entity (`bookInfo.com`) that returned a call to this provider has a trust level of 0.9, so the trust level for `auctions.com` will be 0.6 ($2/3 \times 0.9$). As it is still higher than the *trust*

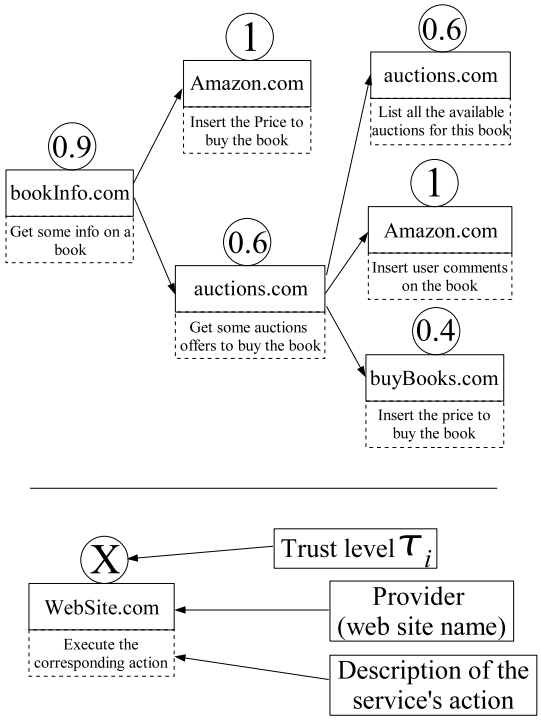

Figure 5. Example of Trust handling in AXML Service calls

threshold Θ (0.5), the service will be invoked and the service calls it will include in its answer will be considered. There are 3 service calls in this answer:

- The first one will simply list the available auctions for the book. As it is still located on `auctions.com`, the computed trust level for this session will be kept and the service will be invoked.
- The second one lists the user's reviews from `Amazon.com`. The trust level is 1, and then the service is called and the list of reviews is returned.
- The last service will try to insert the price of the book from `buyBooks.com`. Since it is not a "trusted entity", we will compute its trust level . The entity (`auctions.com`) that returned the call made to this provider has a trust level of 0.6, so the trust level for `buyBooks.com` will be 0.4 ($2/3 \times 0.6$). As it is not enough to reach the *trust threshold* of 0.5, this service call will be ignored.

4. Discussion

The proposed trust-based solution can handle services with a high number of recursive calls, independently of the depth of the service evaluation tree (as long as the services' trust levels are defined). If no trust level can be computed, we can use the existing matching algorithm. The trust solution can therefore be considered as a way to optimize the existing algorithm, break the k-depth barrier, and ignore the freedom limit in the structure of returned answers from trusted services.
Moreover, as we check the trust level of the nodes in the evaluation tree only once, the complexity of the algorithm given in figure 4 is polynomial. In this approach, we neglected the search time of entities in the database of trusted entities.
This trust-based solution is highly flexible and can be adapted to fit various needs. An industry could for example handle all trust level settings by using credential exchange policies, including trust negotiation processes that happen during services evaluation.

On the other hand, this rather subjective definition of trust could make it difficult to define precisely the trust level on an entity or the method to compute or negotiate it. This is highly context-dependent, and therefore known to be not easily automatized. Since avoiding heavy human management is one of the most important goals of Web services, it could not be the best solution for quickly changing environments where a fine tuning of parameters is always required.
Another drawback of this solution is that it still relies on the rewriting method when we have to perform the service call and that no trust can be computed or negotiated (or that the trust level is not high enough to trust the entity).
We can also consider an attack consisting in returning an answer containing an infinite recursive call, thus bringing our trust evaluation in a loop. This flaw is avoided in the matching method by setting up a depth limit, but setting one in our trust algorithm would not be a wise approach to solving the problem. Thus, we plan in a future work to improve our trust evaluation algorithm to detect such a malicious scheme.

In case of a malicious service call returned by a trusted Web service, one could blacklist the incriminated entity, but it can be too late. One could then only grant trust to entities providing a credential that prove they will carry the responsibility of all the actions achieved by the services they include in their answers. Such a behavior is not idealistic and can be currently observed through better business practices organizations such as BBB (Better Business Bureau) [6].
Concerning spoofing techniques, (when a malicious entity would steal the identity of a trusted one), we rely on authentication standards such as XML Digital Signature. These

XML standards can be easily integrated in our architecture since Active XML is written in XML and it is therefore compliant with all XML security standards. An integration of WS Security standards [17] into Active XML is currently under development by the AXML Team.

Active XML is handling separately the name of the called function (what we refer to as *service* and the URL where this function is found [15]). Simple parsing on the URL can provide us with the *provider*, in case we assimilate the provider to the domain of the URL. For example, in case of the service call "http://www.myWebSite.com/services/Weather", the URL is "http://www.myWebSite.com/services", the domain (provider) is "myWebSite.com" and the service name is "Weather". One has to note that the URL of the service is actually any kind of URI.
Trusted Services (and their corresponding trust level) will be stored on the user system. A removal of not used references is needed in order not to slow down the system. One also need a way to decide when to include a potentially trusted service in the list of valid trusted services. A blacklist of services not to deal with is also required for optimization purpose and to improve ease-of-use of the system. For the sake of space limitation, all these features were omitted. They will be investigated in another paper.

5. Conclusions and Future Work

The trust-based approach of Active XML security will provide a greater flexibility in services' answers, while ensuring security purposes. Its parameters can be easily set to fit the users' needs, but the design of trust settings (trust negotiations and trust transmission functions) could be time consuming and context dependent. It would be however faster to design and more reusable than schemas used for answers rewriting: it is quite difficult to represent all the "authorized answers" structure since it is also a highly context dependent information.
In the approach developed in this paper, trust is represented in a quantitative way ; this approach has some drawbacks such as the subjectivity of the trust level, and we will now focus on designing a framework that handles trust qualitatively, based on credentials exchange.
We also plan to improve the services trust evaluation algorithm and give details on its complexity when taking the trusted sites checking into account.

6. Acknowledgements

The authors would like to thank the reviewers for their useful comments, as well as the members of the Shanghai JiaoTong University Cryptography and Security Lab, and particularly Pr. Zeng GuiHua.

References

[1] S. Abiteboul, O. Benjelloun, B. Cautis, I. Manolescu, T. Milo, and N. Preda. Lazy Query Evaluation for Active XML. In *ACM SIGMOD / PODS 2004 Conference (SIGMOD 2004)*, Paris, FRANCE, June 2004.

[2] S. Abiteboul, O. Benjelloun, and T. Milo. Active xml and active query answers. In *6th International Conference On Flexible Query Answering Systems (FQAS 2004)*, Lyon, FRANCE, June 2004.

[3] Active XML Home Page (AXML). http://www-rocq.inria.fr/gemo/gemo/projects/axml/.

[4] T. Barlow, A. Hess, and K. E. Seamons. Trust negotiation in electronic markets. In *Proceedings of the Eighth Research Symposium on Emerging Electronic Markets (RSEEM 01)*, 2001.

[5] V. Benzaken, G. Castagna, and A. Frisch. Cduce: An xml-centric general-purpose language. In *Proceedings of the ACM International Conference on Functional Programming*, Uppsala, SWEDEN, 2003.

[6] Better Business Bureau (BBB). http://www.bbb.org/.

[7] M. Clifford. Networking in the solar trust model: Determining optimal trust paths in a decentralized trust network. In *18th Annual Computer Security Applications Conference*, page 271, December 2002.

[8] M. Clifford, C. Lavine, and M. Bishop. The solar trust model: Authentication without limitation. In *14th Annual Computer Security Applications Conference*, page 300, December 1998.

[9] DAML-S Home Page. http://www.daml.org/services/.

[10] S. S. et al. Web services : Been there, done that. *IEEE Intelligent systems*, 18(1):72–85, 2003.

[11] R. Falcone, G. Pezzulo, and C. Castelfranchi. A fuzzy approach to a belief-based trust computation. *Lecture Notes on Artificial Intelligence, special issue on "Trust, Reputation and Security: Theories and Practice"*, pages 73–86, 2003.

[12] D. Florescu, A. Grünhagen, and D. Kossmann. Xl: A platform for web services. In *Conference on Innovative Data Systems Research (CIDR)*, Asilomar, CA, USA, January 2003.

[13] J. Holt, R. Bradshaw, K. E. Seamons, , and H. Orman. Hidden credentials. In *2nd ACM Workshop on Privacy in the Electronic Society (WPES'03)*, Washington DC, USA, October 2003.

[14] C. Jonker and J. Treur. Formal analysis of models for the dynamics of trust based on experiences. In *Autonomous Agents '99 Workshop on "Deception, Fraud and Trust in Agent Societies*, pages 81–94, Seattle, USA, May 1999.

[15] T. Milo, S. Abiteboul, B. Amann, O. Benjelloun, and F. Dang Ngoc. Exchanging intensional xml data. In *In Proc. of ACM SIGMOD 2003*, San Diego, CA, USA, June 2003.

[16] M. Naedele. Standards for xml and web services security. *IEEE Computer*, pages 96–98, April 2003.

[17] OASIS Web Services Security TC. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss.

[18] Organization for the Advancement of Structured Information Standards (OASIS). http://www.oasis-open.org/.

[19] The Active XML Team. Active XML primer. Technical report, INRIA, France, 2003.

[20] The DAML Services Coalition. DAML-S: Web Service Description for the Semantic Web. In *The First International Semantic Web Conference (ISWC)*, pages 348–363, June 2002.

[21] UDDI. The UDDI Technical White Paper. http://www.uddi.org/.

[22] M. Winslett, T. Yu, K. E. Seamons, A. Hess, J. Jacobson, R. Jarvis, B. Smith, and L. Yu. Negotiating trust on the web. *IEEE Internet Computing*, November/December.

[23] World Wide Web Consortium (W3C). http://www.w3.org/.

[24] WS-Trust working group. Web Services Trust Language Specification (WS-Trust) 1.1.

[25] WSDL Home Page. http://www.w3.org/tr/wsdl/.