Trust Negotiation for Automated Service Integration

Filippo Agazzi, Michele Tomaiuolo
Dipartimento di Ingegneria dell'Informazione
Università di Parma
Parma, Italy
{agazzi, tomamic}@ce.unipr.it

Abstract—**This paper presents a generic Trust Negotiation framework for Web services, based on the WS-Trust standard. It allows users to create trust incrementally, by disclosing credentials step by step. This way, services and resources can be shared in an open environment, and access can be realized on the basis of peer-to-peer trust relationships. The paper also describes a practical implementation of the framework, which integrates a modular trust engine and a rule engine, which is used as a policy checker.**

Security; trust; Web services; rule-based systems

I. Introduction

The automatic or assisted management of trust relationships is a fundamental requirement to allow the provision and use of disparate services in an open environment. At the global scale, the assumption that all users are known in advance, or they can be easily managed through a traditional Access Control List, is not realistic. In fact, the potential user base of an application provided on the open Internet is still growing, with the mass adoption of social networking tools. Since nowadays contacts among people may develop fully online, possibly with no body of knowledge to associate with a name, more flexible schemes are needed. Currently, no general solutions are available for the problem of identity management, assuming a global database of names and personal profiles is both unfeasible and undesirable. Moreover, online interactions may involve human users together with software agents, possibly with a common understanding of the exchanged messages, on the basis of Semantic Web technologies [1]. Given such a new way people are using the Internet today, the approach of Automated Trust Negotiation (ATN) [2][3] is becoming relevant, because it allows unknown users and agents desiring to share any resource or service, to establish a level of trust in an incremental way through the exchange of credentials.

In this scenario, the open selection and composition of services is made possible, since ATN simplifies the creation and management of trust bounds. In fact, delegation and workflow composition [4] may only be applied on the basis of careful protection of resources and information. This requires a clear analysis of risks and opportunities associated with local trust bounds, on the basis of their socio-cognitive constituents [5], including competence, disposition, dependence and fulfillment. The problem of authorization can thus be solved in a fully distributed way, as access rights can be assigned and delegated on the basis of the local trust assumptions, in a typical Trust Management scheme [6].

This paper is organized as follows: Section II presents an overview and a literature review of ATN; Section III describes a generic trust negotiation framework for Web services, based on the WS-Trust standard; Section IV provides details about practical implementation and use of such a framework, including first performance results; finally, some concluding remarks are provided.

II. Background

A *credential* is generally defined as a digital certificate attesting, via a digital signature, the association of one or more attributes to an entity, identified through its public key. This entity, i.e. the certificate subject, can attest the ownership of the presented credential by demonstrating to possess the corresponding private key. Notably, the entity that originally issued and signed the certificate is not necessarily requested to participate directly in the verification process.

Attributes in a credential can be considered sensible or not. The case of non-sensible attributes does not require any particular care. On the contrary, for the case of sensible attributes, it is necessary to build a certain level of trust between negotiating parties via a structured list of release conditions. Such release conditions are generally known as *policies*. An *access policy* for a resource *R* is a boolean function, which allows or denies access depending on disclosed credentials. It can be written as: $f_R(C_1, C_2, \dots C_n)$, where each C_i is a credential which may be possessed by the requester. A credential itself often holds sensitive information, and it needs to be protected. Thus a *credential disclosure policy* should be defined for revealing a certain credential *C*. It will be a boolean function of the form: $f_C(C_1, C_2, \dots C_n)$. Finally, for selecting the credentials to disclose, a client could need to access a service policy *P*. But also this policy can be considered reserved. In this case, it should be associated with a *policy disclosure policy*: $f_P(C_1, C_2, \dots C_n)$. That is, credentials and policies are to be considered as sensitive resources, and thus they need to be protected by access policies, along other kinds of resources. Access control can be implemented on the basis of different kinds of security credentials, including X.509 certificates and SAML assertions. Moreover, different languages have been defined to represent policies [7][8] in an appropriate and expressive way.

These are the cases where trust negotiation provides its full benefits. Digital credentials are exchanged step by step, to increase the level of trust between involved parties, and the flow of credentials between two entities through a sequence of requests and releases is what is actually intended with trust negotiation.

A policy language for Trust Negotiation must allow to specify all these conditions. A policy has to be considered satisfied only when the requester discloses all the required credentials, and this verification requires to use a formal policy language, with precise semantics. Another important consideration is that, to fit the wide Internet, such language has to be comprehensible and agreed by all involved entities. In the last years, scholars and firms have proposed various languages, like the IBM Trust Policy Language or the Role-Based Trust Management Language (RT). All those languages, however, were related to some particular engines to compute and decide about certain policies. Moreover, a number of languages are being proposed by ongoing research works, but with a limited scope of application, to be shared by some nodes which interact using the same framework or the same software infrastructure, for example in the context of Web Services.

With respect to the management and computation of policies in a trust negotiation, a particularly important element is the policy compliance checker. Starting from a policy and a set of credentials, the policy compliance checker must be able to find the credentials which satisfy the policy, if they are effectively available as a subset of all disclosed credentials. For this purpose, it is also necessary to translate each credential from its original format into an assertion of the policy language. Considering the example of a client requesting a service, one of the problems to solve is how the client comes to know which credentials it is required to present, and how the policies protecting the service and the credentials are disclosed.

A. Negotiation Strategies

From the architectural point of view, each entity participating in an Automated Trust Negotiation has a Security Agent (SA). The SA has the fundamental responsibility of managing the negotiation, computing available policies and credentials, both the local ones and those disclosed by a remote entity, and taking the decision to authorize the disclosure of some credentials and policies at a given phase of a negotiation. These decisions, as well as the exchanged messages and the disclosure of policies and credentials, can be conducted in a number of ways, which is essentially unlimited. A negotiation strategy defines the protocol for the modality and decisions.

The main goal of a strategy is to reach a successful completion of the negotiation protocol, in the respect of certain requirements. A strategy decides when and which credentials must be disclosed and inserted into a message to send to the other party; how much computational load to dedicate to the negotiation (e.g., the maximum number of rounds) and other decisions about the behaviour to pursue during the negotiation. Moreover, a negotiation is not always possible, since for example one of the two parties does not possess sufficient credentials: the strategy has to determine the moment to abandon the negotiation, since it is not possible to conclude it with success.

The execution of a negotiation requires some agreement on a common protocol, with the intended agreement that each subject is free to apply a possibly different strategy. The characteristics of a negotiation are defined by the adopted strategies. Some of the tasks of such strategies are related to which credentials are released, when they are released, which parties are required to unlock the release of another credential and when the negotiation closes, successfully or not. The success of a negotiation is not always possible. One of the subjects could not have all the needed credentials, or one of the subjects could implement a policy imposing a cyclic dependency. Therefore, it is worth defining properties that should be expressed, in the best possible way, by a strategy:

- A strategy should bring a negotiation to success, when such a possibility exists. Strategy having such a property are said to be complete.
- Ideally, a strategy should avoid the release of information which is not strictly required to bring the negotiation to an end.
- A strategy should truncate a negotiation when it cannot bring to a successful conclusion.
- A strategy should recognize a cyclic dependency among credentials and policies.
- The strategy should be reasonably efficient.

There is a vast choice of possible negotiation strategies, each one with its peculiar features. An important distinction can be drawn upon the level of prudence in the disclosure of credentials and policies. In [2] and [7], the following strategies are considered:

Eager Strategy. This strategy is complete and efficient. Participants release all their credentials as soon as the relevant policy is satisfied, without waiting the credential to be requested. This strategy is very simple and brings the negotiation to success whenever it is possible. Nevertheless, it reveals more credentials than those strictly needed to create the minimum level of trust.

Parsimonious Strategy. In this strategy, the number of exchanged credentials is minimized. It is reasonably efficient and it concludes with success whenever it is possible. At the beginning, parties exchange credential requests, but not the credentials themselves. All possible release sequences are then explored. The path that brings the negotiation to success with the minimum number of exposed credentials is selected and followed. Unfortunately, due to the possible limitations in the level of cooperation between two subjects, the global minimum solution is not guaranteed.

Prudent Strategy. This strategy allows establishing trust without revealing irrelevant credentials, while remaining reasonably efficient. In [9] the communication complexity is shown to be $O(n^2)$, and the computational complexity to be $O(n^m)$, where n is the number of credentials and m is the size of the policy regulating the release of credentials.

In the heterogeneous world of Internet, each entity must be free to choose the strategy that is the best compatible with its own requisites and objectives. It is quite possible that two unknown entities will choose different strategies. Thus, there is a problem of how to make such strategies interoperable, and if it is possible. In [10], a family of strategies, called DST (Disclosure Tree Strategy), is proposed as a solution to this problem. A family of strategies is defined as a set of reciprocally compatible and interoperable strategies. An important advantage regards the fact that a Security Agent can choose, among a set of strategies belonging to the same family, the closest one to its own requisites. Moreover, this way it can adopt different strategies, during the different stages of a negotiation.

III. Application of ATN to WSs

This section describes a generic trust negotiation protocol for web services. The protocol is designed in conformance to relevant standards for Web services security. Thus, it first presents an overview of these standards.

A. Standard protocols for Web services security

SOAP Web services can exploit the SOAP header as an extensible container for message metadata, which provides developers with a set of options also covering the most typical security issues. The so-called WS-* specifications are designed in order to be composed with each other. *WS-Security* supports the definition of security tokens inside SOAP messages and uses XML Security specifications to sign or encrypt those tokens or other parts of a SOAP message. It provides a level of abstraction which allows different systems, using different security technologies, to communicate securely using SOAP in a way which is independent from the underlying transport protocol. This level of abstraction allows developers to use existing security infrastructure and established industry standards for authentication, encryption and signature, but also to incorporate new security technologies. Other specifications provide additional SOAP-level security mechanisms. *WS-SecureConversatio*n defines security contexts, which can be used to secure sessions between two parties. *WS-Trust* specifies how security contexts are issued and obtained. It includes methods to issue, validate, renew and forward security tokens, to exchange policies and trust relationships between different parties. *WS-Policy* allows organizations to specify various requirements and qualities about the Web services they expose. This specification provides a general purpose model and the corresponding syntax to describe the requirements and constraints of a Web service as policies, using policy assertions. *WS-SecurityPolicy* is based on the structure of WS-Policy and allows an entity to define, through a set of policy assertions, its own security constraints and requirements. Moreover, a set policy subjects can be associated with each specified assertion. WS-SecurityPolicy allows a Web Service to define a set of assertions, and thus its own security requirements, using a standard and interoperable format [8].

Apart from WS-* specifications, additional formats and protocols are being defined by OASIS, to provide a higher level of interoperability among services. The eXtensible Access Control Markup Language (*XACML*) is a language for specifying Role- or Attribute-Based Access Control policies. The Security Assertion Markup Language (*SAML*), in particular, is an open XML-based format to convey security information associated with a principal. The generic structure of a SAML assertion is very similar to what is usually called a "digital certificate", i.e., an issuer attests some properties about a subject, defines the validity limit of the claim, and digitally signs the document to prove its authenticity and to avoid tampering. SAML itself deals with three different kinds of assertions: (*i*) authentication, (*ii*) attribute, and (*iii*) authorization decision [8].

The WS-Trust standard [11] defines mechanisms for mediating trust relations among entities in the context of Web Services. It considers a security model in which a Web service can request that a received message proves a set of claims (e.g. name, key, privileges, etc) or, more commonly, that it carries a security token representing a relation between the sender and some other entity, trusted by the service provider. In this context, a service provider can request a client, before accessing its services, to present a token released by a trusted entity. A new client would probably not possess a proper token to access the service, in advance. For this reason, WS-Trust defines a protocol for allowing a client to contact an authority, trusted by the service provider, to request the token. Such an authority is defined as a Security Token Service (STS). An STS, on his turn, can define the requirements which clients have to satisfy to obtain the release of a token. As a STS is responsible for releasing those tokens, it is also known as a "token issuer".

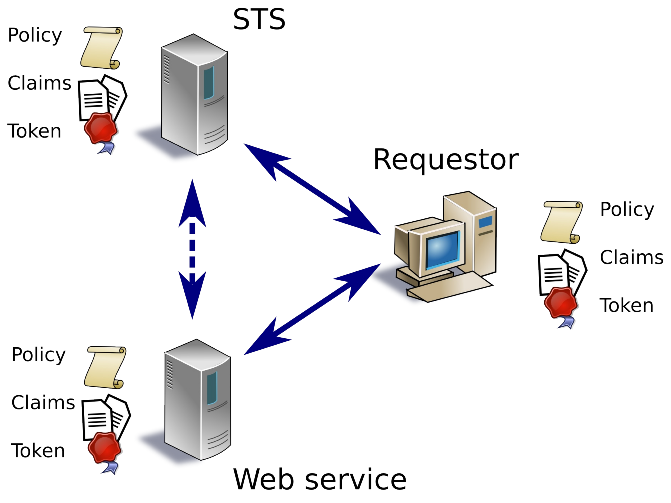

Figure 1. WS-Trust architecture

In Fig.1, arrows represent possible paths of communication among the Requestor (client), the Web service Provider, and the STS. The Requestor contacts the STS for receiving a token. The STS has the duty to verify that the Requestor possess the necessary attributes for obtaining a token. In the case if the policy of the STS is satisfied, the STS releases a token. At this point, the Requestor can send a message to the Web service Provider, attaching the obtained token.

The security token released by the STS must have some features, in particular: (*i*) being verifiable as effectively released by the STS, and (*ii*) effectively authorizing the requester to the use of some services. These features depend on the type of token being released: various technologies may be used to implement the token, such as X.509 and SAML. SAML is well fit for this scenario as it provides a secure way to make assertions about some subjects and their attributes. Otherwise these features may be guaranteed on the basis of a previous agreement, i.e., a secret, shared between the Web service and the STS bound to the service. In fact, an STS can be a platform-level Web service, bound to one or more Web services, for which it plays the role of a trusted authority. A Web service may trust the signature of the STS, or it may request an STS to validate the token, or validate it in autonomously.

A Requestor may be informed about the necessity to use a security token released by an STS, as the needed Web service can publish a policy where a certain *IssuedToken* is requested.

The interaction between a client and an STS occurs through a request-response protocol.

In particular, a *RequestSecurityToken* is used to request a token, and a *RequestSecurityTokenResponse* for responding to the request. Each request must be associated with an action which identifies the possible actions to request to an STS, as defines in the WS-Trust standard: to release, renew, cancel or validate a token. The requestor can also add claims, expressed in a certain "dialect" depending on the application. The requestor may also specify a service which the request applies to, if the STS is associated with multiple Web services; in this case, the exact endpoint reference of the Web service has to be specified.

The response may convey a token through a *RequestedSecurityToken* element. Additionally, it may convey other proofs through an *RequestedProofToken* element, containing data which the Requestor may use to demonstrate to be authorized for using the token. For example, it may contain a secret encrypted with the public key of the Requestor.

B. A Generic ATN Protocol for Web Services

An STS is normally integrated into a system using a single round of messages, i.e. a RequestSecurityToken (RST), sent from the requestor to the STS, followed by a RequestSecurityTokenResponse (RSTR), sent from the STS to the requestor. However, in some scenarios, more steps may be needed before a token is obtained. In fact, the WS-Trust standard foresees the extension of this basic mechanism, named "negotiation and challenge framework", which is depicted in Fig.2.

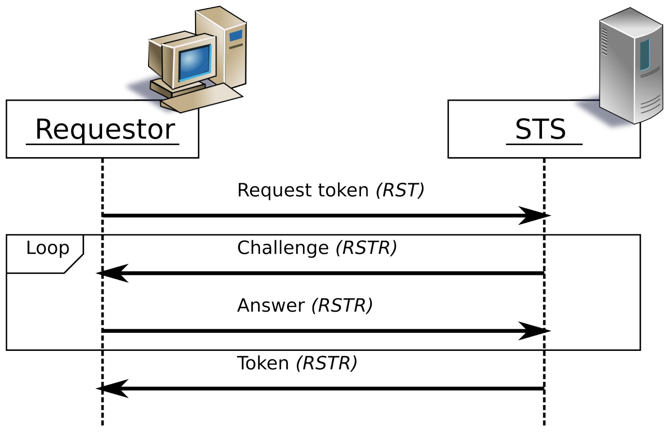

Figure 2. WS-Trust - Negotiation and challenge framework

The message exchange starts with a RST for requesting the token, then an arbitrary number of RSTR messages can be exchanged between the Requestor, or other entities, and the STS. Those RSTR messages may convey any additional information needed for completing the transaction, before finally transmitting the token. The WS-Trust standard defines some elements for proposing a "challenge" the other end, including: SignChallenge, BinaryExchange, KeyExchangeToken. However, it does not specify how to use such elements, or even other arbitrary elements, in a transaction. For example, Policy elements may be used by both parties to exchange their respective policies.

In this work, we propose a generic protocol for ATN. We decided to use some elements already proposed in [9], when possible. However, we organized the protocol and the content schemas to better distinguish the two fundamental phases of the negotiation: (*i*) the initialization, and (*ii*) the real exchange of credentials and policies.

In the initialization phase, the parties use an extensible TNInit element in a single turn of messaging. It contains information useful for defining the parameters of the following negotiation, and for verifying if there is the necessary compatibility, before beginning a real negotiation. A TNInit element can contain: a SignatureMaterial, for proving the possession of a private credential; a StrategyFamily, for identifying a supported family of strategies; a TokenFormat, for specifying the supported type of security token.

In the negotiation phase, the parties use an extensible TNExchange element. It can contain PolicyCollection and TokenCollection elements, for transporting policies and credentials, respectively, disclosed to the other party during the negotiation. Moreover, it can contain TokenType, RequestedSecurityToken and OwnershipProof, for conveying the requested token and other associated proofs.

IV. Implementation of a practical STS framework

Following the design of a generic Trust Negotiation protocol for Web services, a practical implementation has been realized. At this step, it is mainly an experimentation framework, for testing both the functionality and performance of the proposed protocol. However, part from prototype services and clients, most of its components are reusable for creating open SOA-based applications, especially in the case of dynamic service selection and composition.

The framework is available as part of the open source dDelega project [12], at https://github.com/tomamic/dDelega. dDelega is the result of ongoing work started with the development of a security layer for JADE, one the most widespread FIPA-compliant multi-agent systems [13].

In particular, it integrates a trust engine, in compliance to WS-Trust specifications. It also integrates an advanced rule engine for compliance checking against disclosure policies. These engines can be used by parties in a WS environment, by means of translator components that has been realized, in order to complete the integration. At a more basic level, the implementation exploits a number of frameworks developed under the Apache Foundation umbrella, including Axiom, Axis, Rahas, Rampart.

A. Integrating a modular trust engine

The trust engine must be able to evaluate which policies and credentials have to be inserted into the message at each round of the negotiation, on the basis of current state of negotiation and policies and credentials received at the previous round.

TrustBuilder2 (TB2) is a framework for trust negotiation, developed for providing a flexible and extensible tool in the context of research about this problem area. It is the second main version of the TrustBuilder tool and it has been developed at the DAIS (Database and Information Systems) Laboratory of the University of Illinois [14].

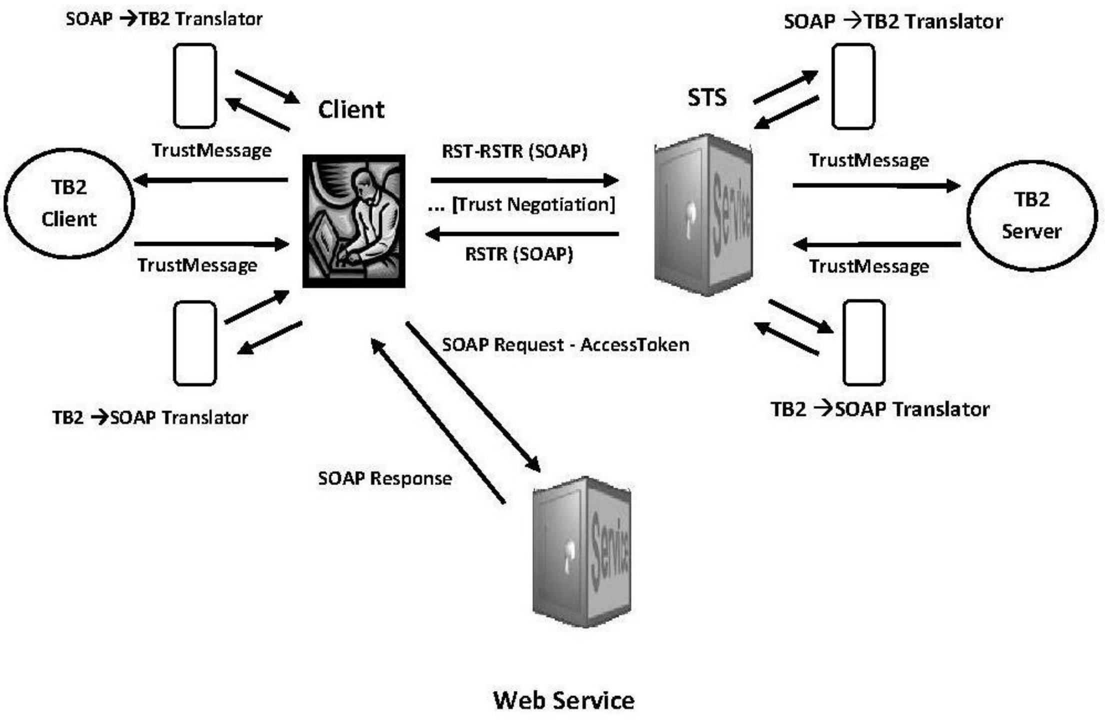

Figure 3. System architecture

TrustBuilder2 has not been realized for usage in the context of Web services, however his modular structure allows it to be extended for: (*i*) using different policy languages, (*ii*) implement different negotiation strategies, (*iii*) and provide support for different types of credentials.

In particular, after a proper translation we defined, it is able to evaluate policies expressed according to the WS-SecurityPolicy language. Starting from received policies and credentials, it is able to analyze them and take decisions about which credentials and policies to disclose, according to the chosen negotiation strategy. The framework uses a policy compliance checker, which has the duty of finding one or more minimal sets of credentials satisfying a given policy. In TB2, the main components of ATN are represented as interfaces, which can be implemented and extended to add new functionalities. They can be distinguished as:

- Strategy module: regarding negotiation strategies.
- Policy compliance checker: regarding the problem of finding a set of credentials satisfying a policy.
- Query interfaces: used to provide access to resources, including local policies and credentials.
- Credential chain module: used to build and validate chains of credentials, during the negotiation process.

TrustBuilder2 is designed according to a model of negotiation with two main phases. The first phase is characterized by the exchange of messages containing data structures, called InitBrick, for communicating the information needed to initialize a negotiation.

After this phase, the main negotiation rounds take place, characterized by the exchange of data structures called TrustMessage, i.e. objects containing policies and credentials to exchange during the negotiation.

In this research work TrustBuilder2 is used as a trust engine for automated trust negotiation. A mechanism has been realized for translating "TB2 messages", i.e. InitBrick and TrustMessage objects, into "WS-Trust messages", i.e. RSTR messages containing TNInit and TNExchange elements, which are exchanged in the context of a "negotiation and challenge framework", as defined by WS-Trust.

Policy and credentials are represented as abstract classes and credentials in TB2, in such a way to make the tool independent from the type of policies and credentials used. The authors of TB2 have also implemented the support for X.509 credentials; in fact, the implementation of this research work uses X.509 credentials. In TB2, credentials are organized in chains; i.e. when a credential, released by an authority, is sent, then the whole chain has also to be sent. In fact, TB2 does not process single credentials, but chains of credentials, through the *CredentialChainMediator* component, which uses algorithms to build and validate chains of credentials. This allows administrators to create decentralized authorities, valid for the different parties participating in a negotiation process; moreover, it allows TB2, when processing a chain, to verify the issuer of a credential released by an entity, starting from the verification of the root certificate of the chain.

Moreover, our implementation requires a user to specify, though configuration files, information about some components to be used by a client and a server, with respect to TB2 functionalities. This allows users to customize negotiation strategies, types of credentials and policy languages to be used in a certain application. The credential loader module can also be customized to load particular credentials into the system; it has access to a list of available credentials. The profile manager module uses the same customization to decide which class loader to use, according to the type of credentials used by the PolicyManager. A policy loader contains information for the PolicyManager, to decide which policy class loader to use.

B. Using a rule engine as a policy checker

A fundamental aspect of TB2 is the logic it uses for the functioning of its compliance checker component. In TB2, the problem of finding a set of credentials satisfying a policy is reformulated into the so-called "many pattern/many object match" problem, i.e., to find the objects matching the given patterns. Here, credentials are considered as objects and policies as patterns, in a problem which can be solved using a production rule engine. The rules of such engines have a standard format, with: an LHS (left hand side), the part of the rule defining the conditions; and an RHS (right hand side), the part of the rule defining the action to perform in the case when the conditions of the LHS are satisfied.

TB2 includes the Clouseau component, that is an expert system using the Jess (Java Expert System Shell) rule engine, which provides APIs for integration into a Java application. The rules, representing the policies of a trust negotiation process, define constraints on credentials. Jess implements the Rete algorithm [15], which allows to solve the "many pattern/many object match" problem. Using an engine of this kind in a trust negotiation process requires to introduce rules for representing policies, which specifies the patterns. The knowledge base, instead, is determined by acquired credentials. An inference can be realized by finding a set of credentials satisfying the policy, which is exactly the duty of the policy checker in TB2. Thus, a policy checker is nothing more than an expert system based on production rules.

Jess does not support natively any object for representing credentials or policies. Instead, to use credentials in Jess and to insert them into its working memory, it is necessary to define their format explicitly. Then, through JessComplianceChecker class, an assert command must be constructed and executed. This requires quite cumbersome code, for constructing thse command as an "assert(...)" string, starting from the object representing the credential.

Instead, in this work we have customized the TrustBuilder2, extending it for using a different rule engine as a policy checker. In particular, we used the *Drools* rule engine [16] for the policy checker component instead of Jess, supported by the currently available version of TB2. Drools is based on the so-called ReteOO algorithm, i.e., an adaptation of the Rete algorithm for object oriented systems. In Drools there are two main storage areas: a Production Memory, where rules are stored, and a Working Memory, where known facts are stored. For trust negotiation, the Production Memory can be used for storing the policies as rules, while the Working Memory can be used for storing the credentials as facts. An important advantage with respect to Jess is that facts in Drools are represented as Java objects, which can be put directly into the Working Memory. This has allowed us to develop a policy checker with a much leaner code than the Jess policy checker. Moreover, the tool is completely open-source, at the contrary of Jess; it is continuously updated, with the addition of new features, and it has the attentions of a vast and lively community of developers.

C. Initial evaluation

The ATN process, as described in the previous sections, was analyzed from the point of view of performance. The evaluation regarded the influence of the various components of the system and the conversions required by those components for communicating. For this tests, a scenario has been realized, in which:

- the client requests a token;
- the STS sends a policy requesting a chain of credentials;
- the client, on the other hand, protects one of the credentials in the chain with a policy, which he discloses to the STS;
- then, the STS discloses the credentials satisfying the client's policy;
- thus, the client discloses the credential chain initially requested by the STS;
- finally, the STS sends the requested token.

Including the initialization phase, the whole process takes 4 rounds, in which both the client and the STS send a message to the other party.

4 rounds, with Enc & Sign	6.0s
4 rounds, w/o Enc & Sign	4.8s
3 rounds, w/o Enc & Sign	4.0s
3 rounds, 1 credential requested by STS	3.0s
4 rounds, TB2, no WS	1.2s

Table 1. Initial performance results

As shown in Tab.1, the execution times vary around a mean value of 6 seconds, including the signature and encryption of SOAP messages, and 4.75 seconds without any signature and encryption. Considering instead a minimal negotiation process of three rounds, the execution time is around 4 seconds. Decreasing the credentials required by the policy from 3 to 1, the execution time does not vary proportionally, but it is reduced only by around 1 second. This means that a significant part of the computation load is absorbed by TB2, for the evaluation of policies, in addition to the basic workload imposed by the WS-* stack [17][18].

These qualitative results are in accordance with those conducted by some authors of TB2 [19], which report that almost half of the total time of execution is used by the policy checker. Another significant comparison is with the execution of a negotiation using only the TB2 tool, in which a TB2Client and a TB2Server communicate directly, through a dedicated socket, without any conversion, signature or encryption: in the same scenario with 4 rounds, as described above, the process takes 1.2s in TB2, against the 6s required by the whole Web services infrastructure implemented in this work.

It is worth noting that more efforts may be dedicated to the optimization and fine tuning of various components the system. Thus, performance may be improved in many aspects. For example, the inclusion of policy statements into Drools is now a process involving various steps and conversions. In future releases of the framework, this process will be streamlined, enabling a more direct inclusion of policies and improving efficiency.

V. CONCLUSIONS

This paper presented the design and implementation of a generic Trust Negotiation framework for Web services. It allows users to create trust automatically, by incrementally disclosing credentials. Modular applications can integrate services provided in an open environment, on the basis of peer-to-peer trust relationships. Interoperability among such services is guaranteed by the conformance to standard protocols for Web services. The realized ATN system is composed of various components and requires various format conversions for messages, policies and credentials. For these reasons, the complete execution of a negotiation process is quite costly and imposes a significant computational overhead. Thus, it is advisable to release tokens which can be used for accessing a number of cohesive services in a given time interval, without repeating the negotiation.

Besides using the framework in generic Web-based applications, further research work will also investigate the possibility of using an Automated Trust Negotiation protocol in distributed social platforms [20]. In fact, especially in the case of location-aware applications, unknown users may need to establish some level of trust before interacting, when meeting at a certain place or at a certain event.

In this sense, the framework described in this work will provide a solid ground for further analysis in different application scenarios, above all for its generality and modularity, which permit to exploit a powerful trust engine and a well known rule engine with very different kinds of protocols and credentials.

REFERENCES

[1] A. Poggi. Developing Ontology Based Applications with O3L. *WSEAS Trans. on Computers* 8(8): 1286-1295, 2009.

[2] Winsborough, W. H., Seamons, K. E., & Jones, V. E. (2000). Automated trust negotiation. In *DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings* (Vol. 1, pp. 88-102). IEEE.

[3] Winslett, M., Yu, T., Seamons, K. E., Hess, A., Jacobson, J., Jarvis, R., & Yu, L. (2002). Negotiating trust in the Web. *Internet Computing, IEEE*, 6(6), 30-37.

[4] A. Negri, A. Poggi, M. Tomaiuolo, P. Turci. Dynamic Grid Tasks Composition and Distribution through Agents. *Concurrency and Computation: Practice and Experience*, 18(8):875-885, 2006.

[5] Venanzi, M., Piunti, M., Falcone, R., & Castelfranchi, C. (2011, July). Facing openness with socio-cognitive trust and categories. *In Proceedings of the Twenty-Second international joint conference on Artificial Intelligence* – Vol. One (pp. 400-405). AAAI Press.

[6] Li, N., Mitchell, J. C., & Winsborough, W. H. (2005). Beyond proof-of-compliance: security analysis in trust management. *Journal of the ACM* (JACM), 52(3), 474-514.

[7] Yu, T., Ma, X., & Winslett, M. (2000, November). PRUNES: an efficient and complete strategy for automated trust negotiation over the Internet. In *Proceedings of the 7th ACM conference on Computer and communications security* (pp. 210-219). ACM.

[8] Bertino, E., Martino, L. D., Paci, F., & Squicciarini, A. C. (2010). Standards for web services security. In *Security for Web Services and Service-Oriented Architectures* (pp. 45-77). Springer Berlin Heidelberg.

[9] Lee, A. J., & Winslett, M. (2008, June). Towards Standards-Compliant Trust Negotiation for Web Services. In *Trust Management II: Proceedings of IFIPTM 2008* (Vol. 263, p. 311). Springer.

[10] Yu, T., Winslett, M., & Seamons, K. E. (2001). Interoperable strategies in automated trust negotiation. In *Proceedings of the 8th ACM conference on Computer and Communications Security* (pp. 146-155).

[11] Nadalin, A., Goodner, M., Gudgin, M., Barbir, A., & Granqvist, H. (2009). WS-Trust 1.4. *OASIS* (February 2009).

[12] Tomaiuolo, M. (2013). dDelega: Trust Management for Web Services. *International Journal of Information Security and Privacy (IJISP)*, 7(3), 53-67. ISSN:1930-1650.

[13] Poggi, A., Tomaiuolo, M., & Vitaglione, G. (2005). A Security Infrastructure for Trust Management in Multi-agent Systems. *Trusting Agents for Trusting Electronic Societies, Theory and Applications in HCI and E-Commerce*, LNCS, vol. 3577, R. Falcone, S. Barber, and M. P. Singh, Eds. Berlin, Germany: Springer, 2005, pp. 162-179.

[14] Lee, A. J., Winslett, M., & Perano, K. J. (2009). Trustbuilder2: A reconfigurable framework for trust negotiation. In *Trust Management III* (pp. 176-195). Springer Berlin Heidelberg.

[15] Forgy, C. L. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial intelligence*, 19(1), 17-37.

[16] Sottara, D., Mello, P., & Proctor, M. (2010). A Configurable Rete-OO Engine for Reasoning with Different Types of Imperfect Information. *IEEE Transactions on Knowledge and Data Engineering*, 22(11), 1535-1548.

[17] Novakouski, M., Simanta, S., Peterson, G., Morris, E., & Lewis, G. (2010). Performance analysis of ws-security mechanisms in soap-based web services (No. CMU/SEI9-2010-TR-023). *Carnegie-Mellon University, Software Engineering Institute*.

[18] Rodrigues, D., Pigatto, D. F., Estrella, J. C., & Branco, K. R. (2011). Performance evaluation of security techniques in web services. In *Proc. of the 13th International Conference on Information Integration and Web-based Applications and Services* (pp. 270-277). ACM.

[19] Lee, A. J. (2008). *Towards practical and secure decentralized attribute-based authorization systems*. ProQuest.

[20] Franchi, E., Poggi, A., Tomaiuolo, M. (2013). Supporting Social Networks with Agent-Based Services. *International Journal of Virtual Communities and Social Networking (IJVCSN)*, 5(1), 62-74. ISSN:1942-9010.