Using Web Services Technology for Inter-enterprise Integration of Digital Time Stamping

A. Cilardo[1], A. Mazzeo[1], L. Romano[1], G.P. Saggese[1], and G. Cattaneo[2]

[1] University of Naples Federico II, Via Claudio 21, 80125 Napoli, Italy
{acilardo,saggese, mazzeo,lrom}@unina.it
[2] Finmatica SpA, Via Amato 15, 84131 Salerno, Italy
g.cattaneo@finmatica.com

Abstract. This paper describes the results of a research activity conducted cooperatively by an academic and an industrial party. It presents a practical solution for and an experience in the implementation of time stamping services and their exposition to the Internet for inter-enterprise integration. State-of-the-art time stamping algorithms and crucial issues related to their practical implementation are discussed. Focus is on integration problems which arise when a potentially large community of enterprises – which rely on a handful of heterogeneous technologies – is willing to access remote third-party time stamping services. We propose a practical architecture which provides time stamping services, both in terms of relative temporal authentication, based on a linear linking scheme, and of absolute temporal authentication, based on publishing mechanisms as well as on a trusted time source. The architecture has been implemented using the emerging Web Services technology. An integration experiment has been conducted to evaluate the effectiveness of the proposed solution.

1 Introduction

Now more than ever, electronic documents are the heart of businesses life. From accounting spreadsheets to e-mail, from secure filings to the intellectual property of the R&D department, business depends on the integrity of corporate data. In this scenario, a particularly challenging issue is how to guarantee the trustworthiness of electronic records, and in particular how to establish beyond a shadow of a doubt that vital electronic records have not been backdated. In many cases, it might be necessary to do that in a court of law.

In order to guarantee that the trustworthiness of long lived electronic documents can be verified over the years, it is crucial that reliable digital time stamping features be made available.

During the last years, especially in the context of legal regulation of digital signature techniques, the organizational and legal aspects of time stamping itself have become the subject of world-wide attention. Time stamping can be thought

R. Meersman and Z. Tari (Eds.): OTM Workshops 2003, LNCS 2889, pp. 960–974, 2003.

of as a set of cryptographic techniques enabling one to ascertain whether an electronic document existed at a certain time (Absolute Temporal Authentication), or was created before another document (Relative Temporal Authentication).

In the literature, time stamping schemes are classified into three distinct categories [14]: simple, linking, and distributed schemes.

In the *simple* scheme, a time stamp is generated by a trusted third party (the Time Stamping Authority, TSA) in such a way that it does not involve data included in other time stamps. The main weakness of this scheme is that the TSA has to be unconditionally trusted: if the TSA fraudulently alters the time parameter of a certain time stamp, nobody can detect the alteration. Also, if a leakage of the signature key of the TSA has occurred, fake time stamps can be forged at will. Some ten years ago, the only known time stamping methods relied on the use of TSAs which were based on the simple scheme. Thus, applications which needed digital time stamping had no choice but trusting the TSA unconditionally.

In 1991, the seminal publication [11] of Haber and Stornetta showed that the trust to the TSA can be greatly reduced by using the linking schemes, or alternatively the distributed schemes. Several papers were published during the last years, which further improved the original schemes [8,7,12,9]. The basic idea behind the *linking scheme* is to generate a time stamp which involves data included in other time stamps. A chain of time stamps is constructed, typically by using a one-way hash function. If an issuer is willing to deliberately alter or forge a certain time stamp, he[1] has to alter all the related time stamps. For this reason, it is more difficult for an issuer to manipulate a time stamp in the linking scheme than it is in the simple scheme.

Finally, in the *distributed scheme* multiple issuers (who could possibly be the users of the service themselves) independently generate a time stamp according to the simple scheme, each one using his own key and time source. The set of issuers designated to sign the time stamp is chosen randomly, in such a way that the submitter cannot determine it a priori. If the number of signing issuers is less than a specific predetermined number, they cannot produce a correct time stamp. This scheme relies on the difficulty for the submitter to collude with all of the issuers which are needed to complete the stamping process. However, the need for a large number of independent issuers makes the distributed scheme rather unpractical in most real-world scenarios, as compared to the linking scheme.

Although the linking and distributed schemes show that the provision of a trusted time stamping service does not necessarily imply blind trust in the TSA, the actual implementation of a time stamping server is is still a hard task, since:

- it uses algorithms and data structures which are extremely sophisticated,
- it requires a complex infrastructure for service delivery to heterogeneous clients, and
- it has challenging requirements in terms of availability and reliability.

[1] Throughout this paper, he and she will be used interchangeably.

For these reasons, having each individual enterprise implement its own time stamping service within the enterprise is not an efficient nor a cost-effective solution.

Neither do major middleware technologies – such as Common Object Request Broker Architecture (CORBA) and .NET – currently provide time stamping services as a built-in facility. Some work has indeed been done on middleware-based time services for ditributed systems, but it addressed the issue of time consistency and not of temporal authentication [22,23].

For the above discussed reasons, a promising solution – and currently the only feasible one – is to have a limited number of independent parties, acting as digital notarization agents, provide time stamping services to a large community of users. Surety [21], the American company founded by the two pioneers of time stamping techniques, Haber and Stornetta, has provided such services since 1994. However, this scenario leads to the non trivial problem of how to integrate third-party time stamping services across enterprise boundaries.

This paper presents a web-based architecture for providing time stamping services suitable for an inter-enterprise integration approach. As far as the algorithm is concerned, we adopted a linear linking scheme, since this solution provides the best compromise in terms of security and performance. All time stamping requests falling in a given time window are gathered in a tree-like structure and compressed into a single data item to be signed and linked to the results coming from the previous time windows. The width of the time window represents the time resolution by which relative temporal order between two distinct stamps can be established. Absolute temporal order is achieved by having the time stamping authority add a time value to the time stamp token before the signature phase.

As to the architecture, we resorted to a multi-tier structure. The back-end server is in charge of the bulk of the computing activity. Currently, it is implemented in software. Work is ongoing to implement it in hardware, to boost the performance of the system. The middle-tier is in charge of leveraging the services provided by the back-end servers, in order to satisfy interoperability requirements which arise from the heterogeneity of the service requestors. To implement the middle-tier, we resorted to the emerging Web Services technology, which enables flexible interaction between applications across enterprise boundaries.

The rest of the paper is organized as follows. Section 2 gives an overview of the Web Services architecture. Section 3 briefly describes the linking schemes used to provide time stamping services. Section 4 presents the architecture of the system, and gives some details about its current implementation. Section 5 presents an integration experiment. The experiment is conducted across enterprise boundaries, with respect to a simple case study application. Section 6 concludes the paper with some final remarks.

2 The Web Services Framework

A *Web Service* [15] is to be thought of as an interface describing a collection of operations that are network-accessible through standardized Extensible Markup Language (XML) messaging. It is described using a formal XML notion, called its *service description*. The description covers all the details necessary to interact with the service, including message formats, transport protocols, and service location. The interface hides the implementation details of the service, allowing it to be used independently of the hardware or software platform on which it is implemented, and also independently of the programming language in which it is written. This allows and encourages Web Services-based applications to be loosely coupled, component-oriented, cross-technology implementations. Web Services fulfil a specific task or a set of tasks. They can be used alone or in conjunction with other Web Services to carry out a complex aggregation or a business transaction.

The Web Services architecture is based upon the interactions between three roles: *Service Provider*, *Service Registry*, and *Service Requestor*. The interactions involve the publish, find, and bind operations. Together, these roles and operations act upon the Web Services artifacts: the Web Service software module and its description. In a typical scenario, a service provider hosts a network-accessible software module (an implementation of a Web Service). The service provider defines a service description for the Web Service and publishes it to a service requestor or service registry. The service requestor uses a find operation to retrieve the service description locally or from the service registry, and uses the service description to bind to the service provider and invoke or interact with the Web Service implementation. Service provider and service requestor roles are logical constructs, and a service can exhibit characteristics of both. Figure 1 illustrates these operations, the components providing them, and their interactions.

The Web Services architecture consists of several layers. To perform the three operations of publish, find and bind in an interoperable manner, there must be a Web Services stack that embraces standards at each level. The foundation of the Web Services stack is the network. Because of its ubiquity, Hyper-Text Transfer

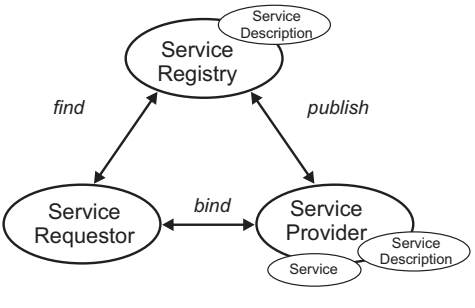

Fig. 1. The Service Oriented Architecture for Web Services.

Protocol (HTTP) is the *de facto* standard network protocol for Internet-available Web Services. Other Internet protocols can be supported, including Simple Mail Transfer Protocol (SMTP) and File Transfer Protocol (FTP). The next layer, XML-based messaging, represents the use of XML as the basis for the messaging protocol. Simple Object Access Protocol (SOAP) is the chosen XML messaging protocol. The service description layer is actually a stack of description documents. Web Service Definition Language (WSDL) is the *de facto* standard for XML-based service description. This is the minimum standard service description necessary to support interoperable Web Services. WSDL defines the interface and mechanics of service interaction. Additional description is necessary to specify the business context, qualities of service, and service-to-service relationships. The WSDL document can be complemented by other service description documents to describe these higher level aspects of the Web Service. For example, business context is described using Universal Description Discovery and Integration (UDDI) data structures in addition to the WSDL document. Service composition and flow are described in a Web Services Flow Language (WSFL) document.

Because a Web Service is defined as being network-accessible via SOAP and represented by a service description, the first three layers of this stack are required to provide or use any Web Service. The simplest stack would thus consist of HTTP for the network layer, the SOAP protocol for the XML messaging layer, and WSDL for the service description layer. This is the interoperable base stack that all inter-enterprise or public Web Services should support. Web Services – especially intra-enterprise or private ones – can support additional/alternative network protocols and distributed computing technologies.

3 Linking Time Stamping Schemes

As already mentioned in the introductory section, the *simple* time stamping scheme relies only on a trustworthy third party (the Time Stamping Authority, TSA) which is in charge of certifying the time by signing the stamp. As we have already pointed out, this time stamping system is rather weak, since it relies on an unconditionally trusted third party, which is wholly liable for the issued stamps. Moreover, in the event that the secrecy of the issuer's signature key is compromised, there is no way to distinguish a genuine stamp from a forged one.

Linking schemes overcome these drawbacks. All the proposed time stamping linking schemes realize one-way dependencies by means of the so-called *collision resistant one-way hash functions*. These include families of functions which compresses bit-strings of arbitrary length $*$ to bit-strings of a fixed length l ($h : \{0,1\}^* \rightarrow \{0,1\}^l$), and which satisfy the following properties:

1 The functions h are easy to compute, and it is easy to pick a member of the family at random.
2 It is computationally infeasible, given one of these functions h, to find a *collision* for h, i.e. a pair of distinct strings x, x' such that $h(x) = h(x')$.

The practical importance of such functions has been known for some time, and researchers have used them in a number of schemes. Hash functions have a number of good properties which are well suited for all kinds of time stamping schemes.

In the time stamping linking scheme, hash functions are mainly used to produce time dependencies between issued time stamps, based on the following consideration: if h is a collision-resistant one-way hash function, and the values $h(x)$ and x are known to a supervisor P at a moment t, then someone (possibly P himself) used x to compute $h(x)$ at a moment prior to t. The TSA combines requests from individual clients which arrive within a given time window, along with certain values related to stamps issued in the past. With such a scheme it is hard to produce fake time stamps because forging a single stamp means forging all verifiable dependencies. Moreover, the need for a trusted TSA can be greatly reduced with the linking schemes by periodically publishing the values used to create the dependencies. Actually, if one can demonstrate the dependency of a stamp on some widely accepted data (for example, a hash value weekly published on a printed newspaper), the TSA is no longer involved in the verification process. The audit can be accomplished at any time and allows the verifier to demonstrate not only whether, but also *when* the linking process was altered and thus to distinguish genuine stamps from unreliable ones (with a certain time resolution depending on the publishing rate).

An additional advantage is that one can hash the document x to be time stamped, and submit only the hash value $y = h(x)$ to the time stamping authority. For the purpose of authentication, stamping y is equivalent to stamping x; however, only a small, fixed-length message is to be submitted to the time stamping authority, greatly reducing the bandwidth problems which would arise if the whole document x were to be processed. Resorting to hash functions solves a privacy issue as well, since the content of the document to be time stamped need not to be revealed to the Time Stamping Service provider. The originator of the document computes the hash values himself, and sends them to the time stamping service. The plain document is only needed for verifying the time stamp. This is very useful for many reasons (like protecting something that one might want to patent). Depending on the design goal for an implementation of time stamping, there may be a single hash function used by everybody, or different hash functions for different users. With a secure signature scheme available, when the time stamping authority receives the hash value, it builds the time stamp token, then signs this response and sends it to the client. By checking the signature, the client is assured that the time stamping authority actually did process the request and that the hash was correctly received.

The most widely known linking schemes are the Haber and Stornetta's ones. The first scheme proposed by Haber and Stornetta is referred to as *linear* linking scheme [11]. In order to diminish the need for trust, the time stamping authority links all time stamps together into a chain using the collision-resistant hash function h. In this case the time stamp for the nth submitted document y_n is $s = sig_{TSS}(n, t_n, ID_n, y_n, L_n)$, where t_n is the current time stated by the time

stamping authority, ID_n is the identifier of the submitter and L_n is the linking information defined by the recursive equation

$$L_n = (t_{n-1}, ID_{n-1}, y_{n-1}, h(L_{n-1})).$$

There are several complications with the practical implementation of this scheme. The major one is that the number of steps needed to verify the one-way relationship between two time stamps is linear with respect to the number of time stamps between them. Hence, a single verification may be as costly as it was to create the whole chain. It has been shown that this solution has impractical trust and broadcast requirements.

Haber and Stornetta proposed a *tree-like* scheme [12], based on Merkle's authentication trees [16,17], in which the time stamping procedure is divided into rounds. Since the tree-like scheme is feasible, we used this solution to implement our time stamping system.

4 System Architecture and Implementation

We designed and implemented an architecture for providing time stamping services with emphasis on interoperability, since our main goal was to enable inter-enterprise integration of time stamping services.

We adopted a multi-tier architecture, since this facilitates a development approach exploiting clean separation of responsibilities, and makes the proposed solution more flexible. The middle-tier components and the back-end servers are located in the first and second tier of the architecture, respectively.

The rest of this section is organized as follows. Subsection 4.1 thoroughly explains the back-end structure and the mechanisms by which the system provides relative temporal authentication and absolute temporal authentication. Such mechanisms are based on a linear linking scheme, and on publishing mechanisms as well as on a trusted time source, respectively. Subsection 4.2 provides implementation details on the middle-tier, including the specific technologies we have used for the development and for the deployment of the system prototype.

4.1 The Back-End

The back-end satisfies the application functional requirements, in that it implements cryptographic routines for the provision of time stamping functions. The overall structure of the back-end server is depicted in Figure 2.

The basic functions the back-end time stamping system provides are: creation of a time stamp, verification of existing time stamps, publication of global hash values for auditing and independent verification.

In the following, we briefly describe the roles of individual back-end components.

The *Proxy component* is in charge of handling communication with the back-end.

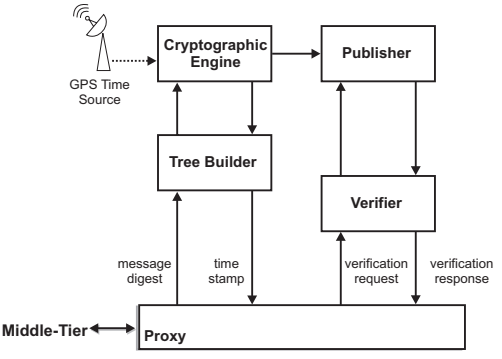

Fig. 2. The organization of the back-end.

The *Tree Builder* component accepts a sequence of input time stamping requests and builds a single time stamping request which is submitted to the Cryptographic Engine. The Cryptographic Engine adds the time value and executes the linking process. Building tree structures results in a performance improvement for the system, since the load which is delivered to the Cryptographic Engine is reduced to a fraction of the one which the Cryptographic Engine would experience if requests were sent directly to it. The Tree Builder component basically uses the Haber and Stornetta's tree-like scheme based on Merkle's authentication trees [16,17]. Each tree corresponds to a round of the time stamping process and includes time stamping requests falling in a time window of about ten seconds. Intra-round relative temporal authentication is not provided. Basically, Merkle's authentication tree is a method of providing short proofs (logarithmic in the number of inputs) that a bitstring x belongs to the set of bitstrings $\{x_1, x_2, \ldots, x_n\}$. During a round a binary tree is constructed as follows (see Figure 3): the leaves are labeled with message digests extracted from the time stamping requests obtained during the round, every non-leaf is labeled with a message digest computed using a hash function h over a concatenation of the labels of its children.

Hence, the label of the root of the tree depends on all the leaves, i.e. aggregated bitstrings. For every leaf it is possible to prove this dependence by exhibiting some more vertices of the tree; the minimal collection of such extra nodes is called an *authentication path*. Figure 3 shows also an example of

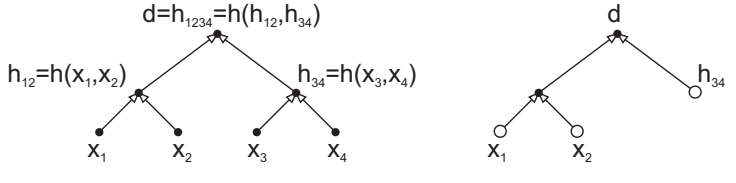

Fig. 3. Merkle's authentication tree and the authentication path for bitstring x_1.

a Merkle's authentication tree where in order to prove that the root value d is dependent on the input x_1 it is enough to add vertices x_2 and h_{34} and compute $h_{12} = h(x_1, x_2)$ and $d = h_{1234} = h(h_{12}, x_{34})$.

The adopted hash function is obtained as a concatenation of two well-known hash functions, i.e. SHA1 [18] and RIPEMD160 [10]:

$$h(x) = \text{SHA1}(x) \,\&\, \text{RIPEMD160}(x)$$

We chose a compound hash function in order to strengthen the security of our time stamping server: in the event that one of the two hash functions is broken, the system will not be compromised and old time stamps will remain valid as long as the other hash function is secure.

The *Cryptographic Engine* component performs two distinct actions:

1. It freezes the time stamping request sequence by linking the consecutive rounds based on a linear scheme. Direct dependencies are created by computing message digests using hash functions.
2. It adds the time value to the time stamp to be formed. Such a value is provided by a trusted time source (such as a GPS clock).

Time stamps are signed by the server to assure their authenticity and instant verifiability.

The *Publisher* component computes a digest for the entire time window of related time-stamping requests, and makes it publicly available by some auditable media. In the current implementation such digests are computed and published daily.

Please note that the two actions performed by the Cryptographic Engine component and by the Publisher provide three different forms of time authentication: the linking process (action 1 of the Cryptographic Engine component) provides the fine-grain relative temporal authentication between time stamps; attaching the absolute time value to the stamps (action 2 of the Cryptographic Engine component) allows users to gain fine-grain information on the time elapsed between two published root values; the publication on a publicly available auditable media (action performed by the Publisher) provides coarse-grain absolute temporal authentication; publishing (action performed by the Publisher) is also one of the central means of retaining the long-term proof value of the time stamps, regardless of the trustworthiness of the time stamping service provider.

Note also that the absolute time is to be thought of as an additional piece of information provided by the system with respect to the basic linking scheme. Obviously, this additional feature relies on the security of the system private key. However, erroneous or fraudulent behaviors would be immediately detected and proved by means of the linking process. The *Verifier* component allows any external user to check the validity of old time stamps. An instant verification on a time stamp is always possible without interacting with the time stamping server by simply checking the digital signature which was provided with the stamp. However, as already mentioned, this method requires an unconditionally trusted time stamping server and critically relies on the security of the provider's

private key. The Verifier component instead provides the verifier with all the data needed to reconstruct a hash chain from the time stamp in question to the closest published hash value, so that the time stamping provider is not required to be trusted and the verification process is as trustworthy as the publishing media are.

As far as the implementation of the system prototype is concerned, the back-end code runs on a IBM Server X-370 with four Pentium Xeon 900 MHz processors, running a Linux kernel 2.4.12. The code is written in C++ language and is based on Crypto++ [24] library for cryptographic functions and on Snacc [25] based efficient C++ routines and data structures to support BER encoding and decoding of ASN.1 data structures.

4.2 The Middle-Tier

The middle-tier is in charge of leveraging services provided by the back-end server, in order to satisfy interoperability requirements which arise from the heterogeneity of the clients. It also decouples the implementation of the service (i.e., the back-end) from its interface. To this aim, the middle-tier wraps the services provided by the back-end. It forwards client requests to the back-end server, and delivers the servers replies to the clients.

Interactions between the middle-tier and the back-end take place via the Gateway (at the middle-tier side) and the Proxy (at the back-end side).

The middle-tier is also responsible for anonymity of requestors. Actually, an important property of a time stamping service is that it should never bind clients to time stamp requests.

The middle-tier was implemented using the the emerging Web Services technology to facilitate service access from any platform. The structure of the middle-tier is shown in Figure 4.

From the client prospective, the system appears as a Web Service provider. The service requestor can access the middle-tier via any transport protocol available over the Internet, provided that a protocol listener for the specific protocol has been implemented (in particular, the current implementation provides an HTTP listener, and FTP listener, which are two of the transports recommended in RFC3161 [6]). Requests and responses are exchanged through SOAP messages. The entry point for requestors' messages consists thus of a SOAP engine (see Figure 4) which coordinates the SOAP message flow through the subsequent components. The engine is also responsible for ensuring that the SOAP

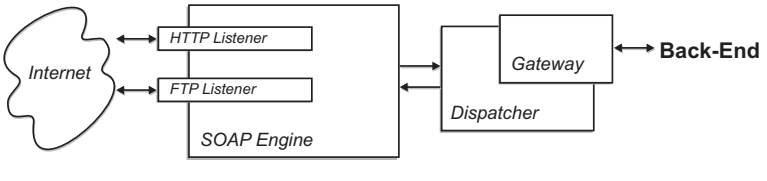

Fig. 4. The organization of the middle-tier.

semantics are followed. Clients are not aware of the implementation details of the service provided by the back-end components. All they know is which services are available and what their interface is like.

The *Dispatcher* component is responsible for acting as a bridge between the SOAP processor and the functional components. It identifies the Java objects to which to delegate the execution of individual activities, and invokes the appropriate methods on such objects.

The *Gateway* component is in charge of data conversion from XML and SOAP data structures to ASN.1 data structure (and viceversa). ASN.1 data structures are compliant with indications contained in RFC3161. An abstract of the time stamping request and of the time stamping token is reported in Fig. 5.

```
TimeStampReq ::= SEQUENCE  {
   version                 INTEGER  { v1(1) },
   messageImprint          MessageImprint,
   reqPolicy               TSAPolicyId              OPTIONAL,
   nonce                   INTEGER                  OPTIONAL,
   certReq                 BOOLEAN                  DEFAULT FALSE,
   extensions              [0] IMPLICIT Extensions  OPTIONAL  }

TimeStampResp ::= SEQUENCE  {
    status              PKIStatusInfo,
    timeStampToken      TimeStampToken     OPTIONAL  }
```

Fig. 5. Abstract of data structures for the time stamping request and response.

The time-stamp token is the data item which contains the certified time information.

All layers of the presented multi-tier architecture work in a pipelined fashion to achieve high throughput.

The prototype version of the system we have built, is deployed within an IBM WebSphere Application Server version 4, on top of a Dell PowerEdge 1400SC with two 1400MHz Pentium III processors, running a Linux Red Hat kernel 2.4.18-3 with dual processor support.

5 Integration Experiment

To test the effectiveness of the proposed solution, we integrated a generic business application with the time stamping server we have implemented. The prototype application handles purchase order documents, whose submission time is to be certified.

Figure 6 shows the class diagram of the front-end code of the case study application.

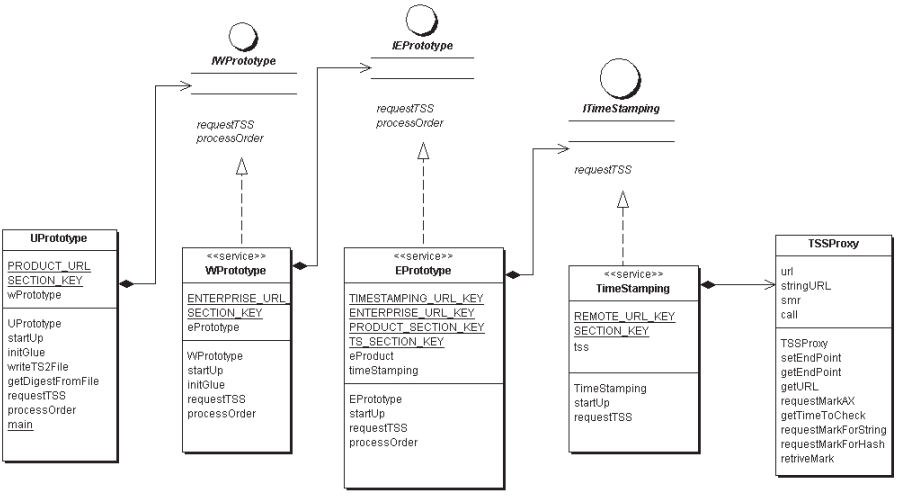

Fig. 6. Main classes for the experimental set-up.

The `UPrototype` class includes the utilities which performs the hash functions. Its input parameter is the path of the file to be processed, and its output parameter is the path of the file where the time-stamping token is to be stored.

The `WPrototype` class acts as a wrapper of the inner layers.

The `EPrototype` class contains the business logic for processing the purchase order data.

The `processOrder` method applies the specific proprietary business logic.

The `requestTSS` method calls the remote time stamping service.

The `TimeStamping` class represents the local component which requests and binds the Web service providing time stamping functionalities.

Figure 7 shows the sequence diagram for the request and delivery of a time stamp.

TSSProxy component acts as a local entry point to the remote service provider. The requestor submits her or his document to the system and passes the return path for the time stamp token. The system takes care of performing the hash functions and calling the remote web service. Once the token is received, the requestor can add the certified time to the order and go on with the remaining order processing operations.

We explicitly note that the interaction between the TSSProxy and the remote time stamping server is asynchronous. The parameter which regulates this interaction is the time-to-check-back, i.e. the time that the TSSProxy has to wait between the moment when the request is forwarded to the time stamping service, and the moment when the response will be available.

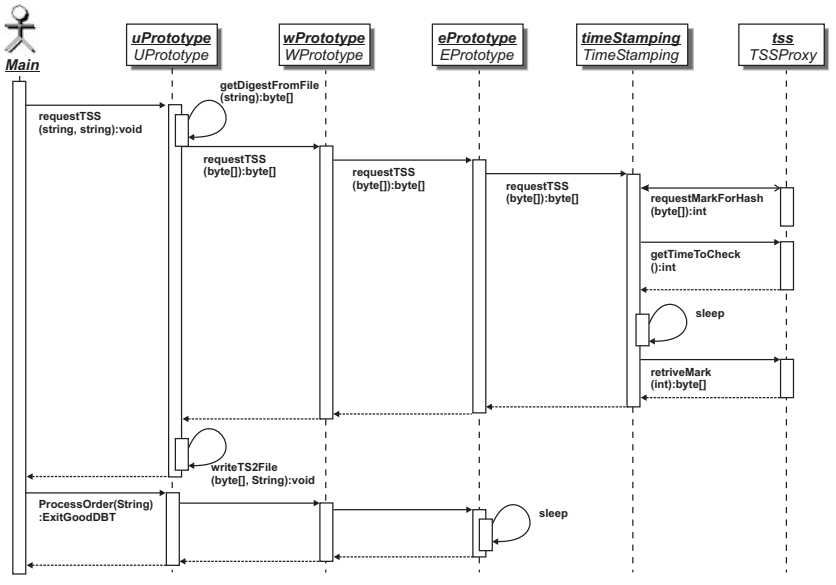

Fig. 7. Main success scenario for the request and delivery of a time stamp.

6 Conclusions

This paper described the results of a research activity conducted cooperatively by an academic and an industrial party. It presented a practical solution for and an experience in the implementation of time stamping services and their exposition to the Internet for inter-enterprise integration. The paper discussed state-of-the-art time stamping algorithms and crucial issues related to their practical implementation. Focus was on integration problems which arise when a potentially large community of enterprises – which rely on a handful of heterogeneous technologies – is willing to access remote third-party time stamping services.

We proposed a practical architecture for providing time stamping services across enterprise boundaries. The architecture relies on a multi-tier structure. The back-end tier provides the basic time stamping functionalities and is based on the Haber and Stornetta linking scheme. The time stamping process is divided into rounds based on tree-like structures. The system provides both relative temporal authentication, based on the linking scheme, and absolute temporal authentication, based on publishing mechanisms as well as on a trusted time source. The middle-tier is in charge of leveraging services provided by the back-end server, in order to satisfy interoperability requirements which arise from the heterogeneity of the clients. It has been implemented using the emerging Web Services technology. Messages are compliant to RFC3161. An integration experiment has been conducted to evaluate the effectiveness of the proposed solution. We integrated a generic business application with the time stamping server we have implemented. Experimental results have shown that the Web

Services technology is well-suited for delivering time stamping services to the users of an inter-enterprise infrastructure.

Acknowledgements. Authors are grateful to Annalisa Caso, Sonia Del Vacchio, Francesco Festa, and Gerardo Maiorano for the numerous fruitful technical discussions, and for developing the bulk of the code.

This work was partially funded by: the University of Naples Federico II, Sintel SPA, the Consorzio Interuniversitario Nazionale per l'Informatica (CINI), the National Research Council (CNR), and the Ministry of University and Research (MIUR), within the framework of projects: SP1 "Sicurezza dei documenti elettronici", "Oltre la Firma Digitale" (OFD), "Gestione in sicurezza dei flussi documentali associati ad applicazioni di commercio elettronico", and "Middleware for advanced services over large-scale, wired-wireless distributed systems" (WEB-MINDS).

References

1. Clarke, F., Ekeland, I.: Nonlinear oscillations and boundary-value problems for Hamiltonian systems. Arch. Rat. Mech. Anal. **78** (1982) 315–333
2. Clarke, F., Ekeland, I.: Solutions périodiques, du période donnée, des équations hamiltoniennes. Note CRAS Paris **287** (1978) 1013–1015
3. Michalek, R., Tarantello, G.: Subharmonic solutions with prescribed minimal period for nonautonomous Hamiltonian systems. J. Diff. Eq. **72** (1988) 28–55
4. Tarantello, G.: Subharmonic solutions for Hamiltonian systems via a $\mathbb{Z}_p$ pseudoindex theory. Annali di Matematica Pura (to appear)
5. Rabinowitz, P.: On subharmonic solutions of a Hamiltonian system. Comm. Pure Appl. Math. **33** (1980) 609–633
6. Adams, C., Cain, P., Pinkas, D., Zuccherato, R.: Internet X.509 Public Key Infrastructure Time Stamp Protocol (TSP) Available at http://www.ietf.org/rfc/rfc3161.txt (2001)
7. Bayer, D., Haber, S., Stornetta, W., S.: Improving the efficiency and reliability of digital timestamping. Proceedings Sequences II: Methods in Communication, Security, and Computer Science. Springer-Verlag (1993) 329–334
8. Benaloh, J., de Mare, M.: Efficient Broadcast time stamping. Technical Report 1, Clarkson University Department of Mathematics and Computer Science (1991)
9. Buldas, A., Laud, P., Lipmaa, H., Villemson, J.: Time Stamping with Binary Linking Schemes. Advances in Cryptology CRYPTO '98 (1998)
10. Dobbertin, H., Bosselaers, A., Preneel, B.: RIPEMD-160, a strengthened version of RIPEMD, Fast Software Encryption. LNCS 1039, D. Gollmann, Ed., Springer-Verlag (1996) 71–82
11. Haber, S., Stornetta, W., S.: How to timestamp a digital document. J. of Cryptology, **3** (1991) 99–111
12. Haber, S., Stornetta, W., S.: Secure Names for Bit-Strings. Proceedings of the 4th ACM Conference on Computer and Communications Security. (1997) 28–35
13. Housley, R.: Cryptographic Message Syntax (CMS). Available at http://www.ietf.org/rfc/rfc2630.txt (1999)

14. International Organization for Standardization and International Electrotechnical Commission: ISO/IEC Standard 18014: Information technology - Security techniques -Time stamping services (2002)
15. Kreger, H.: Web Services Conceptual Architecture. IBM Software Group. Available at http://www-3.ibm.com/software/solutions/webservices/pdf/WSCA.pdf (2001)
16. Merkle, R., C.: Protocols for public key cryptosystems. Proceedings of the 1980 IEEE symposium on security and privacy, IEEE Computer Society Press (1980) 122–134
17. Merkle, R., C.: A certified digital signature. Advances in cryptology - Crypto'89, Springer-Verlag **435** (1989) 218–238
18. National Institute of Standards and Technology: Secure Hash Standard, FIPS PUB 180-1. Federal Information Processing Standards Publication. Available on-line at http://www.itl.nist.gov/fipspubs/fips180-1.htm (1995)
19. Rivest, R.: The MD5 Message-Digest Algorithm, RFC 1321. MIT LCS & RSA Data Security Inc (1992)
20. Sneed, H., M.: Encapsulating legacy software for use in client/server systems. Proceedings of Working Conference on Reverse Engineering (1996) 104–119
21. Surety, Inc., Herndon, Virginia. www.surety.com
22. Zhao, W., Moser, L., E., Melliar-Smith, P., M.: Design and Implementation of a Pluggable Fault Tolerant CORBA Infrastructure. Proceedings of the International Parallel and Distributed Processing Symposium (2002) 343–352
23. Zhao, W., Moser, L., E., Melliar-Smith, P., M.: Design and implementation of a consistent time service for fault-tolerant distributed systems. Proceedings of the International Conference on Dependable Systems and Networks (2003) 341–350
24. Crypto++ Project: http://www.eskimo.com/ weidai/cryptlib.html
25. Snacc website: http://www.fokus.gmd.de/ovma/freeware/snacc/entry.html