Maturing the Distribution of Supportive Tasks in Web Service Framework: Security and Reliability

Beytullah Yildiz
Department of Computer Engineering
TOBB Economics and Technology University
Ankara, Turkey
E-mail: byildiz@etu.edu.tr

***Abstract*—Security and reliability are the crucial features of distributed computing, of which one of the key enabling technologies is Web Service. In a service execution, the main task is carried out by endpoint logic, which is supported by additive functionalities and/or capabilities, called Web Service handlers. The handlers can be detached from the endpoint and distributed to suitable locations to improve availability, scalability, and performance. In this paper, security and reliability, which are among the most fundamental and essential requirements of the handler distribution, are investigated. The proposed environment contains a hybrid encryption scheme, digital signing, authentication, replication, and guaranteed message delivery. The benchmark results are presented to illustrate that the utilized reliability and security mechanisms for the handler distribution are reasonable and efficient.**

Keywords-Web Service; distributed computing; replication; reliability; security.

I. INTRODUCTION

Web Service is a technology providing seamless and loosely coupled interactions that help to build platform-independent distributed systems. Software standards and communication protocols offering common languages are at the foundation of Web Service. The strength of Web Service originates from its ability to hide platform-specific details of the implementations, to expose service interfaces, and to let these self-describing services be registered, published, and discovered dynamically across the Internet. Web Service utilizes the most basic distributed computing approach of client-server interaction. However, it also allows for the creating of complex service composed of many communicating services. Powerful Web Service applications can be assembled by combining the remote and local services.

A single Web Service application integrates endpoint logic and its handlers in a common framework. The main task is accomplished by the service endpoint logic. Supportive functionalities and capabilities, called Web Service handlers, are utilized to provide a full-fledged service. These capabilities might be related to security, reliability, orchestration, and logging, as well as any other necessary capabilities for a distributed system. These capabilities may help to compose a complex service as the handlers can deal with orchestration. They may also be utilized to provide high enough quality of services for a single client-server interaction as the handlers offer security or logging. A Web Service can employ several handlers in a single interaction; a chain of handlers can contribute to a service execution. A service can have a pair of handlers offering functions on both the server and client sides. Some handlers can be, in contrast, employed only in one side of the interaction.

Although handlers are required and inevitable in many cases, they may cause degradation in service quality if their numbers overload the service. A service endpoint with many handlers may suffocate in a single memory space. Hence, it is wise to use additional computing power. This raises the idea of distribution. There are different reasonable approaches for distribution of Web Service handlers. Some suggest that they can be deployed as services; others create a specific distributed environment for them. Distributing the handlers by using a designated setting provides a superior computing environment, especially when the concern is performance. On the other hand, the distribution requires certain features to ensure a suitable environment [1].

Security and reliability are among the most important criteria that need to be considered when a distributed system is being evaluated. Hence, this paper investigates reliability and message security for the distributed Web Service handlers and their effect on the system performance. A fundamental task of cryptography is to protect the secrecy of messages transmitted over public communication lines. For this purpose, in this research, an encryption scheme using a secret key is utilized to encode a message in such a way that an eavesdropper cannot make sense of it. To handle key exchange smoothly, the secret key is ciphered with a public key encryption algorithm. Moreover, a digital signature is used to verify the sender. Reliability mechanisms are also employed to attain a robust environment for Web Service handlers.

The rest of this paper is organized as follows: Section II provides information about related works on reliability and security. Distributed Web Service handler execution is briefly explained in Section III. Section IV investigates reliability. Section V gives details about message security. Finally, the paper is concluded in Section VI.

II. RELATED WORKS

Web Services, an ideal type of technology for distributed applications, benefit from several specifications for security and reliability purposes: WS-Security [2], WS-Trust [3], WS-Federation [4], and WS-ReliableMessaging [5]. These specifications provide the common language to develop secure, reliable, and interoperable interactions between clients and services.

WS-Security addresses security by leveraging existing standards and provides a framework to embed these mechanisms into a SOAP message. This happens in a transport-neutral fashion. WS-Security defines a SOAP header element, which contains the information defined by the XML signature that conveys how the message was signed, the key that was used, and the resulting signature value. Likewise, the encryption information is inserted into the SOAP header. WS-Trust explains the mechanisms to use security token and methods to establish trust relationships. It enables secure conversations between Web Services by defining how to issue, renew, and cancel the security tokens. WS-Federation defines mechanisms to let different security realms unite. Hence, authorized access to a resource handled in one realm can be provided to security principals whose identities are controlled in other realms.

In the Web Service reliability, the WS-ReliableMessaging specification offers an outline to ensure reliable message delivery between the sender and receiver for Web Services. The specification provides an acknowledgement-based scheme to guarantee that data are transferred between the communicating entities. Although it is mainly for point-to-point communication, the specification also supports service composition and transactional interaction.

Web Service specifications describe the syntax and do not define implementation mechanisms or APIs, which remain proprietary to individual vendors. Other than specifications, there are also works and research on security and reliability for Web services. Jayalath and Fernando describe a basic design and implementation approach for building security and reliability layers for Apache Axis 2 [6]. Moser et al. explain dependability features, including SOAP connection failover, replication, and checkpointing, in addition to reliable messaging and transaction management. Their paper also presents security technologies, including encryption and digital signatures for Web Services specifications, as well as other security technologies [7]. Pallemulle et al. present a middleware that supports interaction between replicated Web Services while providing strict fault isolation assurances [8]. Lei et al. propose greedy replica optimizers to improve reliability for a data-intensive online grid system [9]. Aghdaie and Tamir present a transparent mechanism that provides high reliability and availability for Web Services. Their paper explores fault tolerance even for the requests being processed at the time of server failure. The scheme can handle dynamic execution and does not enforce deterministic servers [10]. Zhang presents an integrated security framework based on the use of authentication, authorization, confidentiality, and integrity mechanisms for Web Services, and proposes a model to integrate and implement these security mechanisms in order to make Web Services robust [11]. Yamany et al. propose a metadata framework providing different levels to describe the available variations of the authentication, authorization, and privacy features. With the metadata, the security features are constructed to assist the service consumer and provider in reaching an agreement on how to meet their needs [12].

Security and reliability are not the concerns of only Web and Grid Service technologies. Other distributed computing applications also offer necessary reliability and security mechanisms. Fault tolerance approaches such as replication, recovery techniques, self-reconfiguration of systems, and dynamic binding are applied in various studies to improve reliability. Many research projects and applications utilize Secure Sockets Layer (SSL)/Transport Layer Security (TSL) protocols. Others prefer to utilize symmetric or asymmetric crypto systems. Some research projects offer a hybrid approach combining symmetric and asymmetric key encryption algorithms to offer superior solution.

Vaca et al. propose an automatic identification of faults by means of model-based diagnosis, which helps to establish particular fault tolerance mechanisms such as replications and checkpoints [13]. Zhao et al. design a scheduling algorithm offering reliability satisfying the user's requirement without exceeding the system capacity. The authors explain how to achieve the minimum number of replicas for each task while satisfying the user's reliability requirement with the minimum resources. They also target acceptable performance for execution time [14].

Desmedt et al. demonstrate a scheme that uses a public key to encrypt a random key, which is used to encrypt the actual message with a symmetric encryption algorithm [15]. Ramachandran et al. use a public/private key model for securely communicating messages [16]. Rizvi et al. present an implementation of a secure application syndicating symmetric and asymmetric key algorithms to minimize the execution time and maximize the security [17]. Ramaraj et al. describe a hybrid encryption based on AES and RSA for online transactions [18]. Palanisamy et al. propose the use of a symmetric key algorithm to encrypt and decrypt data and RSA for the symmetric key's encryption/decryption [19]. Damiani et al. discuss the applicability of outsourced Database Management System solutions to the cloud and provide an outline for management of confidential data in public clouds. It utilizes symmetric and asymmetric encryption for privacy and signing [20].

Kemathy et al. investigate a component-based security solution for XML messaging [21]. Ammari et al. provide architecture securing XML messages by encrypting flagged XML parts, each with a different type of encryption depending on data sensitivity and the defined importance level [22].

III. DISTRIBUTION

In this paper, the distribution of Web Service handlers is explored by utilizing a Message-Oriented Middleware (MOM), which is more narrowly focused on messaging. The MOM is designed to be as simple as possible while still offering robust support for messaging. Unlike SOAP messaging, MOM message headers and basic routing information are not contained in XML. This allows more efficient processing, since XML parsing is slow compared to the speed at which routing decisions are made in a specialized messaging system.

In general, the same proven security and reliability mechanisms from the related works are utilized. However, the present research differs in the use of a designated messaging system to improve efficiency and in having an end-to-end solution to complete the distributed handlers' secure and reliable execution in this setting.

The distributed handler execution is organized by a specialized management tool, which contains the orchestration engine explored in [23]. The constructs of the orchestration engine answer the wide range of the handler's execution configuration, such as serial, parallel, and conditional processing. In addition to orchestration, the distribution manager employs an efficient execution engine to meet the performance requirements. The details of the manager are provided in [24]. The engine utilizes a MOM to distribute the tasks to the handlers. The execution manager is so efficient that the overhead justifies the distribution, as investigated in [25]. This paper extends the required security and reliability mechanisms for the handler distribution explained in [1].

The execution of a message in the distributed environment is shown in Figure 1. The incoming requests to the Web Service are delivered to the distributed handler manager by a Web Service Container such as Apache Axis. The manager stores the requests, called messages, in the Message Execution Queue. The messages are sent to the distributed handlers and the responses are received after the successful handler execution. The manager ensures that each message is executed without being interrupted. Every message execution contains one or more stages. Several distributed handlers may construct a single stage for which handlers concurrently process the message. The manager awaits the completion of the handler executions before starting the delivery of the message to the next stage. This procedure continues until all stages of a message are completed. At the end, the successfully obtained output returns to the Web Service Container. All of the messages in the Message Execution Queue are executed concurrently.

Since the handlers are located in separate memory spaces, the message on the wire should be secured against unauthorized access. An adversary can see the important information and/or modify the message. Moreover, the handler distribution manager and the handlers should authenticate themselves to prevent an adversary from intervening in the interaction. Hence, the following issues need to be addressed for the purpose of security:

- Eavesdropping: Potential hackers who have access to the network are able to read the messages.
- Message modification: The message travelling between the handlers and distribution manager can be modified by an unauthorized person.
- False messages: It is fairly easy to produce false messages and send them as if from an actual computing node.
- Message replay: Similar to message modification, the message formed by a handler or the distribution manager can be saved by others and sent again.
- Repudiation: As messages can be forged, there is no way of validating that a message has been sent by a particular node.

The reliability of a handler itself is also essential for successful execution. The message must reach the distributed handlers and be executed without failure. Hence, the reliability of the message delivery is critical. Additionally, the system must have mechanisms in place to deal with the failure of the handlers.

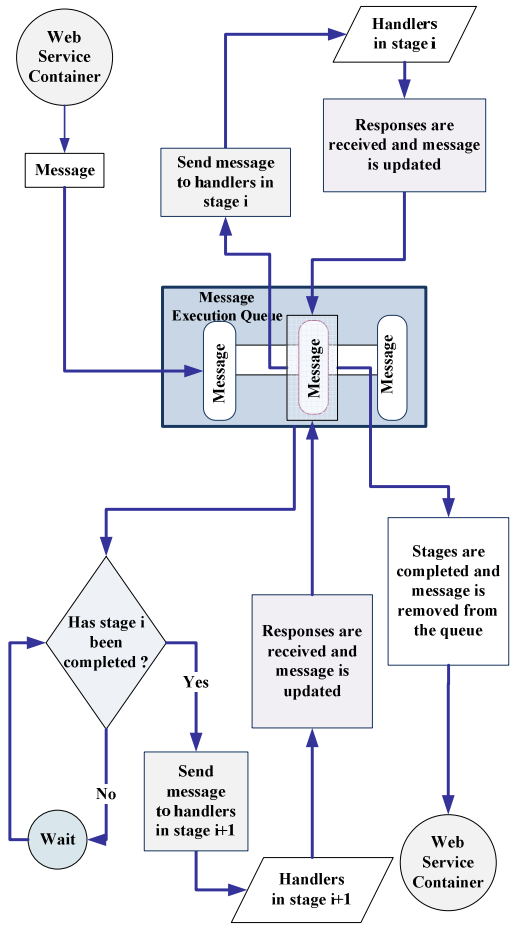

Figure 1. Executing the messages in the distributed Web Service handlers.

IV. RELIABILITY

Software reliability is described as the probability that the software functions without failure under given conditions during a specified period of time [26]. Reliability is also measured in terms of percentage of failure circumstances in a given number of attempts to compensate for variations in usage over time [27]. For Web Services, although reliability is viewed by some researchers as a non-functional characteristic [28], Zhang and Zhang describe one of the more comprehensive definitions of Web Service reliability as a combination of correctness, fault tolerance, availability, performance, and interoperability, where both functional and non-functional components are considered [29].

In this paper, reliability will be investigated in two categories: the reliability originating from the handler replication and the reliability coming from the utilization of a reliable messaging system.

A. Replicating handlers

Replication is critical for reliability, mobility, availability, and performance of a computing system. There are basically three replications: data, process, and message. These concepts are extensively explored in [30]. Data replication is the most heavily investigated one. However, the other replications are also very important in the distributed systems, especially for Service-Oriented Architecture.

The process replication is particularly of main interest in this paper because the intention is to investigate the replication of the handlers. There exist two main approaches in this area. The first one is modular redundancy [31]. The second approach is called primary/standby [32]. Modular redundancy has replicated components that perform the same functionalities. All replicas are active. On the other hand, the primary/standby approach utilizes a primary process to perform the execution. The remaining replicas wait in their standby state. They become active when the primary replica fails.

The processes can be classified into two categories: no consistency and consistency. The first category is the simplest one; the processes are stateless. They do not keep any information for the processed data. Therefore, consistency is not an issue between the processes. Replicated instances can be allowed to run concurrently. On the other hand, replicas may enter an inconsistent state if the process is not atomic and stateful. Inconsistency has been extensively investigated in [33].

Replication is a very important capability where a handler is inadequate. Sometimes, a handler may not be able to answer the incoming requests. The tasks may line up such that the overall performance degrades. This is similar to a shopping center, where customers are waiting in line to be served. The solution is to add one or more persons to serve when necessary. Similarly, adding a replica to help with the execution contributes to the overall performance.

In addition to the performance, a replica can be leveraged for fault tolerance. It is possible that a handler crashes. The replication contributes to the continuity of the execution and improves the availability and reliability. Without using handler replication in the case of an error, the whole computation cannot continue. The computation becomes more resilient with handler replication. The execution continues as long as at least one replica of every handler has not failed.

For n handlers with the replication factor of R, the execution can be successful for R-1 failures per handler. The maximum allowable number of errors is:

$$\sum_{i=1}^{n} R_i - 1 \quad (1)$$

where n is the number of Web Service handlers and R_i is the replication number of the *i-th* handler. The execution cannot continue even in a single handler fault, where $\forall i \in N: R_i = 1$.

In the distributed Web Service handler execution environment, a variation of the primary/standby approach is utilized. Dynamic binding ideas are employed for the replicated handlers using the primary/standby approach. Dynamic binding is a technique that allows services to be linked at run-time [34] [35]. The execution manager decides at run-time which distributed handler is invoked: the primary handler or a replica. The handlers are prioritized. The handler with the highest priority is assigned to execute a message. The other replicas wait until their priorities become highest. The system is able to change the priority during the execution. When a fault occurs, the handler priority is minimized.

If the replicated handlers were executed concurrently, a checkpoint mechanism must have been utilized. The checkpoint mechanism is based on the idea of saving the state of the system. In fault detection, the execution of the system is recovered from the checkpoint where the state was saved. The recovery mechanism is only launched when a fault occurs. Compensation handlers (Rollback) are specific computing nodes that limit and confine the effects created by a faulty handler. Compensation handlers allow the execution of the faulty replicated handlers to be rolled back to a specific point.

The checkpoint approach presents some drawbacks. It necessitates the introduction of additional elements into the distributed handler execution design. It requires extra time to check each important point, and recovery processes need to be activated when the rollback occurs. In fact, establishing the correct and minimal checkpoint and recovery structure is a highly complex task. The checkpoint solution is not, in short, suitable in view of the fact that the rollback mechanism could introduce a very high overhead in the case of a fault. Hence, the primary/standby approach is preferred for the distributed replica handlers.

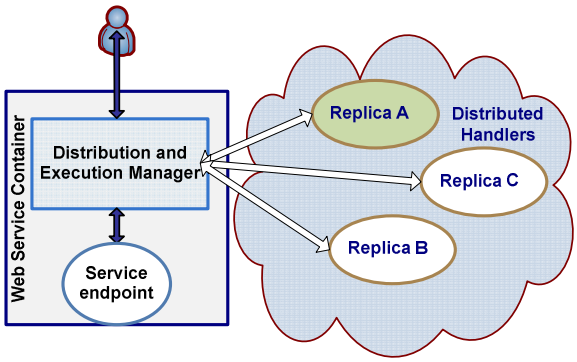

Figure 2. The execution of replicas using the primary/standby approach.

The replicas are never allowed to be executed concurrently, except in the case of stateless handlers. Even though they are allowed to run in a parallel manner, they cannot join the processing of the same message. The messages have to be different so that the parallel execution does not cause inconsistency. Figure 2 depicts a replicated handler processing the incoming message while the other replicas await their turns.

When only one of the several replicated handlers is executed, as shown in the square in Figure 3, the following formula works to compute the reliability value:

$$R_{RH} = \sum_{i=1}^{n} P_i \, R_i \qquad (2)$$

where R_{RH} is the reliability of the replicated handlers' execution, P_i is the execution probability of handler i, and $\sum_{i=1}^{n} P_i = 1$.

The reliability of the parallel handlers with the AND junction and the reliability of the serial handlers can be formulated as:

$$R_s = \prod_{i=1}^{n} R_i \qquad (3)$$

where R_s is the reliability of the handlers' execution and R_i is the reliability of handler i.

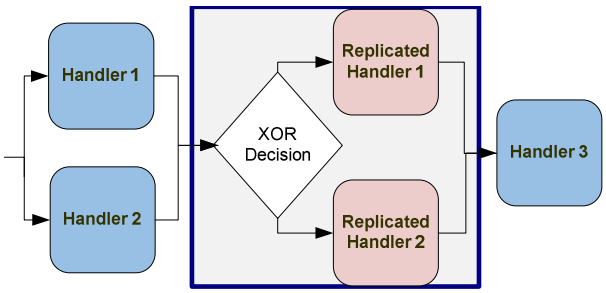

Figure 3. A sample configuration for the handlers' execution.

By using formulas 2 and 3, the reliability of the handlers' execution in Figure 3 can be formulated as:

$$R_s = \prod_{i=1}^{2} R_i \, . \sum_{R_i=1}^{2} P_{R_i} R_{R_i} \, . R_3 \qquad (4)$$

$$R_s = \prod_{i=1}^{3} R_i \, . \sum_{R_i=1}^{2} P_{R_i} \, R_{R_i} \qquad (5)$$

where R_s is the reliability of the handlers' execution. R_i is the reliability of the ith replica and $P_{R_i} = 1$ for only one replicated handler, which is executed; the value is 0 for the remaining handlers.

B. Reliable messaging

The distributed handler mechanism benefits from two different sources for reliable message delivery: a messaging broker and its own execution mechanism.

The messaging system, NaradaBrokering, provides message-level reliability. It also offers supportive functionalities to the messaging and a very reasonable performance [36]. The messages can be queued up to several thousands and are gradually delivered to their destinations to provide flow control for the messaging. Additionally, the system has a Reliable Delivery Service (RDS) component that delivers the payload even if a node fails [37].

RDS stores all the published events that match up with any one of its managed templates, which contain the set of headers and content descriptors. This archival operation is the initiator for any error correction, which is caused by the events being lost in transit to their targeted destinations and also by the entities recovering either from disconnect or a failure. For every managed template, RDS also maintains a list of entities for which it facilitates reliable delivery. RDS may also manage information regarding access controls, authorizations, and credentials of the entities that generate or consume events, which are targeted to this managed template.

When an entity is ready to start publishing events on a given template, it issues a discovery request to find out the availability of RDS, which provides the archival environment for the generated template events. The publisher will not circulate template events until it receives a confirmation that RDS is available.

The publisher ensures that the events are stored by RDS for every template event that it produces. After successful delivery of the event to RDS, the event is archived and a message is sent to the publisher to verify that the message was received by RDS successfully. Otherwise, a failure message with the related event id is sent back to the publisher. After verification, the suitable matching engine is utilized to compute the destinations associated with the template event.

A subscriber registers with RDS. A sequence number linked with the archival of the interaction is recorded. The number can be also described as an epoch, which signifies the point from which the registered entity is authorized to receive events conforming to the template. Once a template event has been archived, RDS issues a notification. The notifications allow a subscribing entity to keep track of the template events while facilitating error detection and correction. Upon receipt of the notification, the subscribing entity confirms the reception of the corresponding template event.

When an entity reconnects to the broker network after failures, the entity retrieves the template events that were issued and those that were in transit before the entity's leaving. After the receipt of the recovery request, RDS scans the dissemination table starting at the sync related to the entity and then generates an acknowledgment-response invoice event outlining the archival sequences that the entity did not previously receive. Accordingly, the missing events are provided to the receiver.

In addition to this, a reliable mechanism for the Web Service handler execution environment is built on top of the reliable messaging that NaradaBrokering provides. The distributed Web Service handler mechanism is able to repeat the execution of a specific handler in the event of a failure. Failure is declared when a response is not received from a distributed handler. There are several possible reasons behind an unsuccessful response. For example, the communication link may be broken, or the handler may not have successfully processed the message because of either an error or a crash. The distributed Web Service handler mechanism checks the possibilities by sending the message several times to its destination. In each attempt, it waits for a specific amount of time. This duration is either assigned or calculated by the system. After several unsuccessful attempts, the message processing may switch to a replica, depending on its priority. As discussed previously, handlers can populate their replicas to improve availability and reliability.

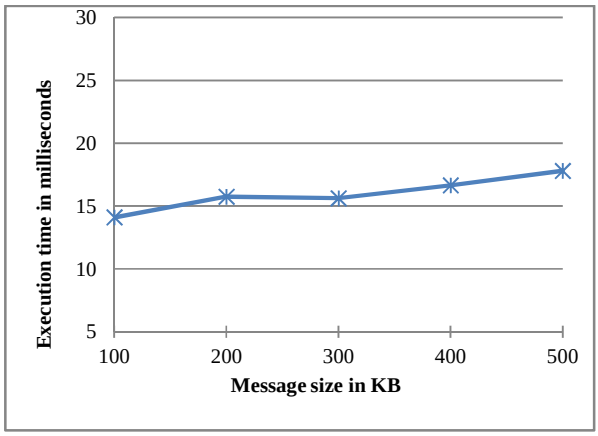

Figure 4. The cost of the reliability mechanism for the distributed Web Service handlers.

For a reliable messaging benchmark, two HP DL 380 G7, 2 x Xeon Six Core, 2.93 GHz, 48 GB memory physical machines are utilized. The machines are virtualized to create four 4-core and 16 GB memory machines and one 8-core 32 GB memory machine. These machines are connected to each other via a LAN and share a common storage system. Virtual machines use Windows Server 2008 R2 64-bit operating systems. The cost of the reliable mechanism of the messaging for the distributed handlers is shown in Figure 4. The cost contains the time needed for reliability procedures to send the tasks to the distributed Web Service handlers or receive the responses back. The time for the handlers' executions and the time for the messaging are excluded to illustrate only the reliability cost for varying message sizes. Figure 4 shows that the message size does not affect the cost of the reliability of the messaging very much. The cost is very reasonable when reliability is a necessity for the distribution.

V. Message Security

Security is one of the most important issues for computing systems. Critical data can be seen or altered by an unauthorized person. This is increasingly important if the data are transferred through the network, which is a more vulnerable environment.

Local computing does not expose its data to the outside world very much, but this is not the case for distributed computing. The computation is shared between nodes, which may be physically dispersed in the distributed environment. The transmission of the data among the nodes may expose critical information to dangerous vulnerabilities. Hence, the transportation channels between the computing nodes must be secured in addition to the security of the nodes.

NaradaBrokering, which is utilized for messaging, has a security framework that is able to support secure interactions between the distributed handlers [38]. The security infrastructure consists of a Key Management Center (KMC), which provides a host of functions specific to the management of keys in the system. The KMC stores the public keys of the interacting entities. It also provides authentication and authorization mechanisms to offer an enhanced environment for secure messaging.

Authentication is an elementary security requirement to prove that an entity possesses a claimed identity [39]. The basic tool that a person can use to prove a claimed identity is generally something that the person knows (e.g., a password), something that the person has (e.g., an authentication token), or a biometric property (e.g., fingerprints or iris recognition). Different mechanisms can be used for cryptographic authentication. Keyed hash functions or symmetric ciphers that utilize a specific key are among the examples. The key is only available to the entity to be authenticated and the verifier. One requirement for authentication mechanisms using the key is obviously the protection of the applied key. The leakage of the key causes

the collapse of the security mechanism. Another requirement is that the key should preferably be unique for the interacting entities. An attack is confined with this specific communication if the key of an interaction gets compromised.

Asymmetric cryptography can also be used for authentication. A proper procedure based on elliptic curves is described in [40]. A cryptographic operation is performed, to be authenticated using the entity's private key. The verifier checks the received response using the corresponding public key by performing a cryptographic verification operation on the received value. NaradaBrokering utilizes an authentication mechanism for the publishers and subscribers, which are the computing nodes for the distributed handler execution. For the authentication, the publisher or subscriber sends its signed request by using private key. The broker verifies this request by using the public key of the entities.

The KMC incorporates with an authorization module to manage the usage of the messaging. Every topic has an access control list that authorizes the subscribers. Similarly, an access control list exists for the publishers. After verification of the signature, the publisher or subscriber is permitted to access the entity according to the relevant access control lists.

The message traveling between the computing nodes is described in Figure 5. It contains a unique id, properties, and a payload. The unique message id is a distinctive name for a message. The handler execution mechanism may host many messages being executed at one moment. Hence, an identifier is necessary to achieve the correct executions; a Universally Unique Identifier (UUID) generated id is assigned to every message. The generator assures that there will not be more than one of the same id in the system. Thus, the design gives enough assurance that the message executions are not blended.

```
<context>
  <id>4099d6dc-0b0e-4aaa-95ff-2e758722a959</id>
    <properties>
     <encKey> bèø3DKUQ …</encKey >
     <sender>
       <senderId>12345… </ senderId >
       <signed>šZQSİÑU,k…</signed >
     </sender >
      ….
  </properties>
  <payload>
     ….
  </payload>
</context>
```

Figure 5. The message format for the distributed Web Service handlers.

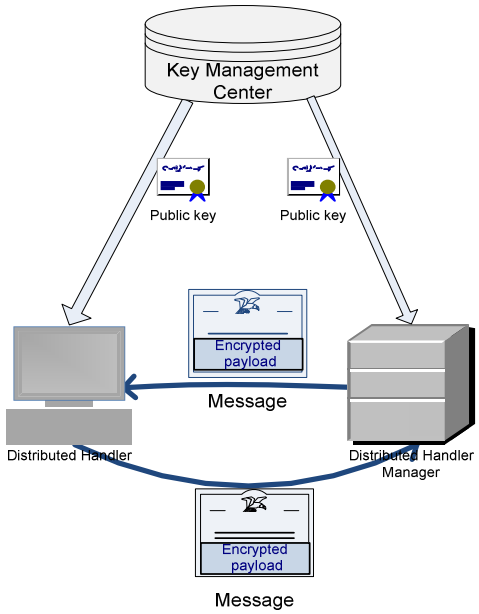

Figure 6. Security mechanism for a distributed handler execution.

The second important part of the message format is the properties section. This part conveys the required additional information to the computing nodes. The information can be specific to a single handler or generic for all handlers. There is a property that contains the encryption key. It is a symmetric key that is created for a single message. The key size is usually selected to be large enough to provide the necessary security. On the other hand, it must be kept in mind that generating larger keys is time-consuming. The average time taken for key generation for different bit sizes is presented in [41]. Therefore, the same symmetric key can be utilized to send a group of messages for a period of time. Additionally, the properties section contains the sender's signature to prove the sender's authenticity. The sender signs its unique id with its private key. Both the sender's id and signature are added to the properties section. The last part of the distributed handler message format is the payload, which contains the encrypted message.

The performance of the asymmetric key encryption is worse than the performance of the symmetric key encryption [42]. It can take about 1000 times more CPU time to process an asymmetric encryption or decryption than a symmetric encryption or decryption. Nevertheless, an important advantage of asymmetric ciphers over symmetric ciphers is that no secret channel is necessary to exchange the keys. The receiver needs only to be confident about the authenticity of the public key provider. Asymmetric ciphers also cause fewer key management problems than symmetric ciphers. Only $2n$ keys are needed for n nodes to communicate securely with each other in an asymmetric key encryption system. However, in a system based on a symmetric cipher, $n(n - 1)/2$ secret keys are needed for secure interaction among n nodes. Because of these features,

asymmetric ciphers are typically used for non-repudiation, for authentication through digital signatures, and for the distribution of symmetric keys. Thus, the asymmetric key algorithm is able to support the solving of the key exchange and management problem of the symmetric keys. On the other hand, symmetric ciphers are used for bulk encryption.

To use the best part of the algorithms, a hybrid approach is utilized. Figure 6 demonstrates a secure messaging architecture for the distributed handlers. The payload of the message is encrypted by a symmetric cipher algorithm. Advanced Encryption Standard (AES) is used for the encryption. AES is a natural choice for the symmetric key algorithm because it has been analyzed extensively and used worldwide. The cryptography scheme is a symmetric block cipher that encrypts and decrypts blocks of data. The AES key generation algorithm takes a random seed as an input. A 256-bit session key is created and passed within the properties section of the message to the other computing node for decryption.

The RSA algorithm is utilized for the asymmetric key encryption. The sender encrypts the symmetric session key with the 2048-bit public key of the receiver to present the confidentiality. The RSA algorithm requires two large prime numbers as the input along with a random seed. All of these inputs, which are created randomly, are provided for key generations. The created private keys are then kept locally, and the public key is stored in the KMC.

With the commonly used RSA implementations, doubling the RSA key length means that encryption will be more than two times slower and decryption will be almost four times slower, as shown in Table I. In general, RSA encryption is much faster than RSA decryption. The fast encryption relies on the use of a short public exponent. The RSA algorithm is commonly used in this way. It is possible to have an RSA public key with a long public exponent, which will make encryption as slow as decryption. However, because a long public exponent does not improve security, short public exponents are widespread. Some well-known RSA implementations do not even support long public exponents. Hence, the decryption exponent is typically huge, whereas the encryption exponent is small.

TABLE I. PUBLIC KEY ENCRYPTION AND DECRYPTION RESULTS.

	Encryption time in milliseconds		Decryption time in milliseconds	
Plain text size in KB	1024-bit	2048-bit	1024-bit	2048-bit
100	262	470	4703	14906
200	563	892	8844	29594
300	784	1298	13703	44542
400	1048	1735	17766	59064
500	1298	2391	22454	73737

TABLE II. COMPERISON OF CIPHER TEXT SIZE IN PUBLIC KEY ENCRYPTION.

Plain text size in KB	Cipher text size in KB	
	1024-bit	2048-bit
100	109	105
200	218	209
300	329	314
400	438	418
500	547	522

The key length of an RSA key specifies the number of bits in the modulus. A larger key increases the maximum number of bytes that we can encrypt in a block at once, as well as the security of the encryption. On the other hand, with every doubling of the RSA key length, decryption becomes much slower. Key length also affects the speed of encryption, but the speed of decryption is usually of greater concern.

Moreover, depending on the padding scheme, the cipher text size increases in RSA encryption. This is another factor that is taken into consideration while using this system. Table II shows the text sizes for 1024-bit and 2048-bit key encryptions. When using a 1024-bit RSA key with PKCS #1 padding, it is not possible to encrypt a string that is longer than 117 bytes. Increasing the size of the RSA key to 2048 bits will allow the encrypting of 245 bytes of data, but longer RSA keys are expensive and they take more time to generate and operate. On the other hand, the AES encryption algorithm does not show the size increase in the encrypted text even though it also does block ciphering with 128 bytes.

The size of cipher text can be calculated with the following formula:

$$c = p + b - (p \bmod b) \quad (6)$$

where c is the cipher text size, p is the plain text and b is the block size.

As mentioned earlier, the authentication of the sender and receiver and authorization to access the message are established by the security mechanisms of the messaging broker. Figure 7 shows the tasks happening between the sender and receiver for a single interaction. The sender generates the symmetric session key for a message or a group of messages. The payload containing the message is encrypted with this symmetric session key with the AES algorithm. The sender looks up the receiver's public key in the KMC. The RSA algorithm is used to encrypt the symmetric session key with the receiver's public key. Hence, only the node that has the right private key can decrypt the session key to get the encrypted payload. At the same time, the sender authenticates itself by signing its id with its private key.

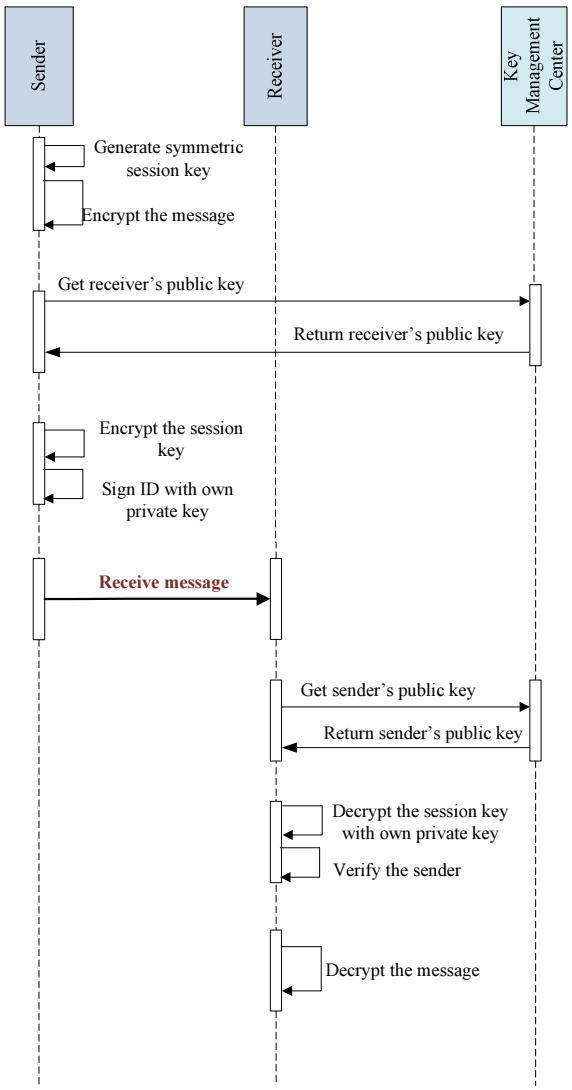

Figure 7. Interactions while sending a message between computing nodes.

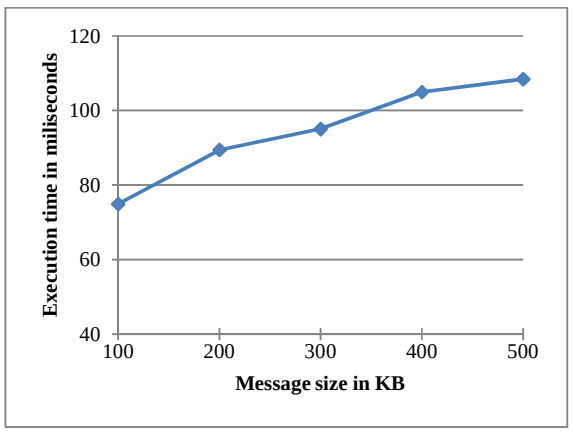

Figure 8. The cost of the security mechanism for the distributed Web Service handlers.

When the subscriber receives the message, it identifies and verifies the sender by using the sender's public key, which is retrieved from the KMC. The session key carried within the "encKey" tag is then decrypted by the receiver's private key. The retrieved session key is used to decrypt the payload to get the original message. In this interaction, the senders and receivers are either the distributed Web Service handlers or the distribution manager.

The benchmarks showing the cost of the aforementioned security mechanism and the results of the tables containing public key encryption are determined in the same environment as the reliability benchmark, as discussed in Section IV.B. Figure 8 shows the cost of the secure environment for varying payload sizes. The signing of the sender's id to present the authentication causes very small overhead. Instead of asymmetric key encryption, the usage of the symmetric key to encrypt the messages provides reasonable execution time. As stated earlier, even though asymmetric key encryption solves the problem of key exchange, it does not accomplish the message encryption and decryption for large sizes at an affordable cost. In short, a hybrid approach using both asymmetric and symmetric ciphers helps to improve security at a reasonable cost.

VI. CONCLUSION

Although the distribution of Web Service handlers provides many advantages in terms of scalability, availability, and performance, it necessitates a reliable and secure atmosphere. The instruments explained in this paper for secure and reliable handler distribution and the support tools of the utilized messaging broker grant the necessary features for this atmosphere. Utilized reliability mechanisms deal with the distributed computing node failures by using replication and ensure the message delivery. The hybrid security approach advances the environment by offering a solution for the key exchange problem of the symmetric encryption and by reducing the cost of the asymmetric cipher algorithm. The design also delivers the authentication and authorization mechanisms for the distributed handlers. The benchmark results show that the costs originating from the utilized instruments are acceptable and affordable. In short, the design of the distributed execution with the security and reliability tools offers a satisfactory environment for Web Service handlers. Moreover, it should be kept in mind that a secure and reliable environment must be employed in many mission-critical tasks.

REFERENCES

[1] B. Yildiz, "Reliability and message security for distributed web service handlers," Proc. the Seventh International Conference on Internet and Web Applications and Services (ICIW 2012) , Stuttgart, Germany, May 2012, pp. 17-22.

[2] Web Service Security (WS-Security), Available: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss, [Retrieved: May 20, 2013].

[3] Web Service Reliable Messaging (WS-ReliableMessaging), Available:http://public.dhe.ibm.com/software/dw/specs/ws-

rm/ws-reliablemessaging200502.pdf, [Retrieved: May 20, 2013].

[4] Web Service Trust (WS-Trust), Available:http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/errata01/os/ws-trust-1.4-errata01-os-complete.html, [Retrieved: May 20, 2013].

[5] Web Service Federation (WS-Federation), Available:http://docs.oasis-open.org/wsfed/federation/v1.2/os /ws-federation-1.2-spec-os.html, [Retrieved: May 20, 2013].

[6] C. M. Jayalath and R. U. Fernando, "A modular architecture for secure and reliable distributed communication," Proc. the Second International Conference on Availability, Reliability and Security (ARES07), Washington, DC, USA, 2007, pp. 621-628, DOI=10.1109/ARES.2007.7.

[7] L. E. Moser, P. M. Melliar-Smith, and W. Zhao, ''Building dependable and secure web services,'' Journal of Software, vol.2, no.1, 2007, pp. 14-26.

[8] S. L. Pallemulle, H. D. Thorvaldsson, and K. J. Goldman, "Byzantine fault-tolerant web services for n-tier and service oriented architectures," Proc. the 28th International Conference on Distributed Computing Systems (ICDCS '08), Washington, DC, USA, 2008, pp. 260-268. DOI=10.1109/ICDCS.2008.94.

[9] M. Lei, S. V. Vrbsky, and Z. Qi, "Online grid replication optimizers to improve system reliability," Proc. IEEE International Parallel and Distributed Processing Symposium (IPDPS 2007), March 2007, pp.1-8.

[10] N. Aghdaie and Y. Tamir, "CoRAL: a transparent fault-tolerant web service," Journal of System Software, vol. 82, no. 1, January 2009, pp. 131-143, DOI=10.1016/j.jss.2008.06.036.

[11] W. Zhang, "Integrated security framework for secure web services," Proc. the Third International Symposium on Intelligent Information Technology and Security Informatics (IITSI '10), Washington, DC, USA, 2010, pp. 178-183, DOI=10.1109/IITSI.2010.8.

[12] H. F. EL Yamany, M. A. M. Capretz, and D. S. Allison, "Quality of security service for web services within SOA," Proc. Congress on Services (SERVICES '09). Washington, DC, USA, 2009, pp. 653-660, DOI=10.1109/SERVICES-I.2009.95.

[13] A. J. Varela-Vaca, R. M. Gasca, D. B. Nunez, and S. P. Hidalgo, "Fault tolerance framework using model-based diagnosis: towards dependable business processes," International Journal on Advances in Security, issn 1942-2636 vol. 4, no. 1 & 2, year 2011, pp.11-22.

[14] L. Zhao,Y. Ren, Y. Xiang, and K Sakurai, "Fault-tolerant scheduling with dynamic number of replicas in heterogeneous systems", Proc. 12th IEEE International Conference on High Performance Computing and Communications (HPCC), Melbourne, September 2010, pp. 434-441, DOI=10.1109/HPCC.2010.72.

[15] Y. Desmedt, R. Gennaro, K. Kurosawa, and V. Shoup, "A new and improved paradigm for hybrid encryption secure against chosen ciphertext attack," Journal of Cryptology, vol. 23, iss. 2, January 2010, pp. 91-120, DOI=10.1007/s00145-009-9051-4.

[16] K. Ramachandran, H. Lutfiyya, and M. Perry, "Chaavi: a privacy preserving architecture for webmail systems," Proc. the Second International Conference on Cloud Computing, GRIDs, and Virtualization, 2011, pp. 133–140.

[17] S. S. Rizvi, A. Riasat, and K. M. Elleithy, "Combining private and public key encryption techniques for providing extreme secure environment for an academic institution application," International Journal of Network Security & Its Application (IJNSA), vol.2, no.1, January 2010, pp. 82-96.

[18] E. Ramaraj, S. Karthikeyan, and M. Hemalatha, "A Design of security protocol using hybrid encryption technique (AES-Rijndael and RSA)," International Journal of the Computer, the Internet and Management, vol. 17, no. 1, January 2009, pp. 78-86.

[19] V. Palanisamy and A. M. Jeneba, "Hybrid cryptography by the implementation of RSA and AES," International Journal of Current Research, vol. 3, iss. 4, April 2011, pp.241-244.

[20] E. Damiani, F. Pagano, and D Pagano, "iPrivacy: a distributed approach to privacy on the cloud," International Journal on Advances in Security, vol. 4, no. 3&4, 2011, pp. 185-197.

[21] K. Komathy, V. Ramachandran, and P. Vivekanandan, "Security for XML messaging services: a component-based approach," Journal of Network and Computer Applications, vol. 26, iss. 2, April 2003, pp. 197-211, DOI=10.1016/S1084-8045(03)00003-1.

[22] F. T. Ammari and J. Lu, "Advanced XML security: framework for building secure XML management system (SXMS)," Proc. the Seventh International Conference on Information Technology: New Generations (ITNG '10), Washington, DC, , 2010, pp. 120-125, DOI=10.1109/ITNG.2010.124.

[23] B. Yildiz, G. Fox, and S. Pallickara, "An orchestration for distributed web service handlers," Proc. International Conference on Internet and Web Applications and Services (ICIW08), June 2008, Athens, Greece, pp. 638-643.

[24] B. Yildiz, "Distributed handler architecture," Ph.D. Dissertation. Indiana University, Bloomington, IN, USA. Advisor: Geoffrey C. Fox. 2007.

[25] B. Yildiz and G. Fox, "Measuring overhead for distributed web service handler," Proc. the Third IEEE International Conference on Computer Science and Information Technology (ICCSIT 2010), July 2010, pp. 566-570.

[26] H. Zo, D. Nazareth, and H. Jain, "Measuring reliability of applications composed of web services," Proc. 40th Annual Hawaii International Conference on System Sciences (HICSS '07), 2007, pp. 278- 288.

[27] J. D. Musa, Software reliability engineering, McGraw-Hill, New York, NY, 1999.

[28] A. Arsanjani, B. Hailpern, J. Martin, and P. Tarr, "Web services: promises and compromises, "ACM Queue, 1 (1), pp. 48-58, March 2003.

[29] J. Zhang and L.-J. Zhang, "Criteria analysis and validation of the reliability of Web Services-oriented systems," Proc. the IEEE International Conference on Web Services (ICWS'05), Orlando, Florida, July 2005, pp. 621-628.

[30] A. Helal, A. Heddaya, and B.K. Bhargava, "Replication techniques in distributed systems," Advances in Database Systems, vol. 4, 2002, pp. 61-71, DOI: 10.1007/0-306-47796-3_3.

[31] P.A. Lee and T. Anderson, Fault tolerance: principles and practice, Springer-Verlag New York, Inc. Secaucus, 1990.

[32] P. P. W. Chan, M. R. Lyu, and M. Malek, "Making services fault tolerant," Proc. 3rd International Conference on Service Availability (ISAS'06), Berlin, Heidelberg, 2006, pp. 43-61, DOI=10.1007/11955498_4.

[33] P.T.T. Huyen and K. Ochimizu, "Toward inconsistency awareness in collaborative software development," Proc. 18th Asia Pacific Software Engineering Conference (APSEC), Dec. 2011, pp. 154-162.

[34] A. Erradi and P. Maheshwari, "Dynamic binding framework for adaptive web services," Proc. the 2008 Third International Conference on Internet and Web Applications and Services. Washington, DC, USA, 2008, pp. 162–167.

[35] U. K¨uster and B. K¨onig-Ries, "Dynamic binding for BPEL processes - a lightweight approach to integrate semantics into web services," Proc. 4th International Conference on Service

Oriented Computing (ICSOC06), Chicago, Illinois, USA, 2006, pp. 116–127.

[36] S. Pallickara and G. Fox, “NaradaBrokering: a distributed middleware framework and architecture for enabling durable peer-to-peer grids,” Proc. the ACM/IFIP/USENIX International Conference on Middleware (Middleware '03), 2003, pp. 41-61.

[37] S. Pallickara and G. Fox, "A scheme for reliable delivery of events in distributed middleware systems," Proc. the IEEE International Conference on Autonomic Computing (ICAC'04), New York, NY, May 2004, pp. 328-329.

[38] S. Pallickara, M. Pierce, G. Fox, Y. Yan, and Y, Huang, "A Security framework for distributed brokering systems," Available:http://www.naradabrokering.org, [Retrieved: May 20, 2013].

[39] R. Falk and S. Fries, “Advances in protecting remote component authentication,” International Journal on Advances in Security, issn 1942-2636 vol. 5, no. 1 & 2, year 2012, pp. 28-35.

[40] M. Braun, E. Hess, and B. Meyer, “Using Elliptic Curves on RFID Tags,” International Journal of Computer Science and Network Security, vol. 2, February 2008, pp. 1-9.

[41] K. Ramachandran, H. Lutfiyya, and M. Perry, “A Privacy preserving solution for web mail systems with searchable encryption,” International Journal on Advances in Security, issn 1942-2636 vol. 5, no. 1 & 2, year 2012, pp. 26-45.

[42] C. Narasimham and J. Pradhan,“Evaluation of performance characteristics of cryptosystem using text files”, Journal of Theoretical and Applied Information Technology, vol. 4, iss. 1, 2008, pp. 56-60.