Secure Orchestration of Symbolic Grid Services

Technical Report

July 2008

SCIEnce e-Austria Team

Institute e-Austria Timişoara

Abstract. *One of the major goals of the SCIEnce European Project is to create a symbolic computational infrastructure built upon Grid. This infrastructure will allow exploring new ways of solving symbolic computing specific problems and potentially lead to solutions for problems that were previously computationally unfeasible. This document gives an overview of the general problems related to the security issues in Grid architectures, identifies general security solutions offered by the Globus Toolkit 4 middleware and investigates how these solutions can be applied for the infrastructure we aim to build.*

1 Introduction

Grid computing, even though a relatively new field in Computer Science, has been identified as one of the emerging technologies that will change the world. The Grid can exist everywhere in our daily lives even if we are or not aware of it. It exists in cell phones, PDAs, computer networks, or research fields such as physics, astronomy, chemistry, etc. Many technologies are converging to create these future Internet architectures including utility Grids, Web services, wireless devices, peer-to-peer collaboration, virtual clusters, business process automation, and enterprise portals. Due to this potential and already huge appliance in various technologies, present day security systems try to secure the Grid with regard to authentication and data encryption. These attempts try to keep the Grid safe from DDoS (Distributed Denial of Service), data capturing, false authentications, etc. The nature of Internet with its limited resources makes the Grid vulnerable to DDoS [1]. Bandwidth, processing power, and storage capacities are all targets of attacks. Each Grid has limited resources that can be exhausted by a sufficient number of users. Thus when the attacks are successful, the Grid can be out of service. The other security issues such that data capturing or false authentication can be more problematic as it addresses the content that is transmitted and used inside a Grid.

1.1 Classification of Grid Security issues

According to [2] the issues concerning Grid security can be generally grouped into four main categories:

- Protect applications and data from the system where computation executes;
- Stronger authentication needed (for users and code);
- Protect local execution from remote systems;
- Different admin domains/security policies.

However this is not the only possible classification. The paper [3] uses a classification (Figure 1) which groups the Grid security issues in three main categories:

- Architecture related: information security issues, service level security issues and resource level issues;
- Infrastructure related: host and network level security issues such as data protection, job starvation, and host availability;
- Management related: managing credentials and trust.

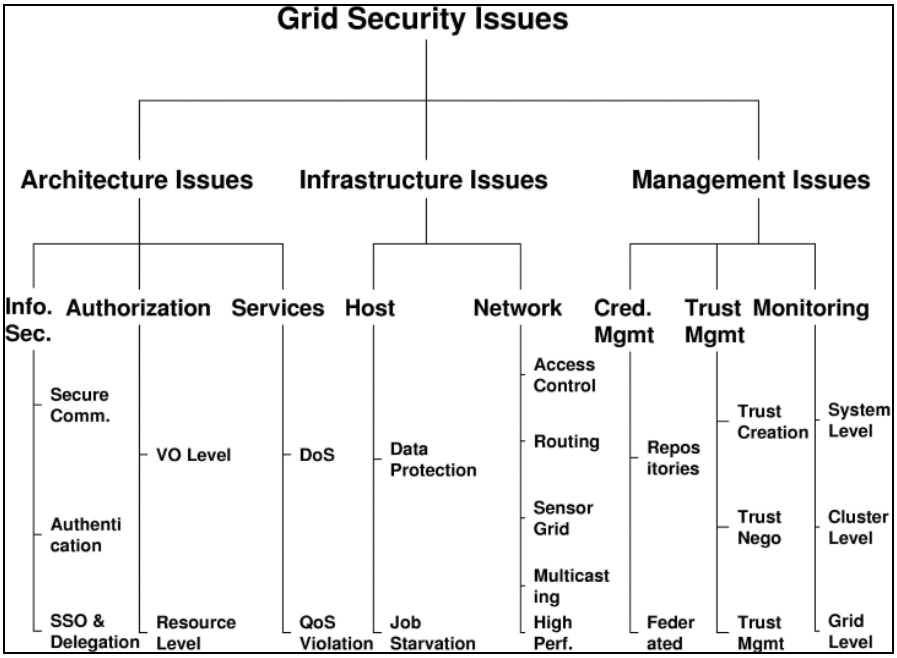

Fig. 1 Taxonomy of Grid security issues [3]

With regard to users and data one can notice the following important aspects related to security:

- **Authentication**: password-based, Kerberos authentication, SSL authentication, certification authorities;
- **Authorization**;
- **Integrity and Confidentiality:** symmetric and asymmetric cryptography, PGP (Pretty Good Privacy), SSL.

Authentication is the process of verifying identity of a participant to an operation or request. A principal is an entity whose identity is verified. It can be a local user or a user logged into remote system. The authentication method is different from traditional systems in Grid systems. The later ones require mutual authentication in order to ensure that resources and data not provided by an attacker, while the former ones authenticate client to protect the server. It is not advisable to send a non-

encrypted password over the network unless one is certain that it will not be read by any un-trusted process.

In what follows we mention shortly the most known **authentication systems.**

The **Kerberos** system is a protocol for authentication and key distribution. It is used with symmetric encryption systems and has a better performance than systems that require public keys or asymmetric cryptography. It is well suited for frequent authentication, is centrally administered and requires a trusted, on line certification authority (KDC).

The **Secure Sockets Layer** (SSL) is a widely deployed system used in every browser. The client authenticates the identity of the server, and sends a session key in order to set up an encrypted communication. The server has a certificate that contains the public key. The client can authenticate itself only if it possesses a certificate.

Certificates and **Certification Authorities** (CA) provide a binding between an encryption key and an authenticated identity. The CA is a third party that certifies or validates the binding. A certificate is an object which contains:

- Distinguished name of a principal;
- In asymmetric cryptographic systems: the public key of the principal;
- Optional attributes: authorizations, group memberships, email addresses, alternate names.

The most widely form of certificate is the **X.509 certificate** used in: Web browsers, secure email services, and electronic payment systems. Paper [4] describes a platform where one would like to use an X.509 certificate for accessing a Web service based on a Kerberos-authenticated identity. In order for this to be achieved one might use the WS-Trust [5]. This standard defines a protocol by which Web services in different trust domains can exchange security tokens or use in the WS-Security [6] header of SOAP messages. Clients use the WS-Trust protocols to obtain security tokens from Security Token Services.

Authorization is the process that determines whether a particular operation is allowed or not. In traditional systems it is based on the authenticated identity of the requester and other local information (ACLs). Grid systems require however a different approach by determining whether access to the resource is allowed or not

Integrity and Confidentiality has as main purpose the goal to protect data during the transmission on network. The protection can be achieved by using a cryptographic mechanism. Terms such as encryption, decryption, cipher-text or plain-text are vital in this process. Encrypted messages (using symmetric systems such as DES, triple-DES, IDEA, Blowfish, RC4, RC5, or asymmetric ones such as RSA or DSA) provide integrity and confidentiality by protecting data from eavesdroppers, and by making checksums to protect data integrity.

1.2 Addressing Grid Security issues

In order to address the Grid security requirements, [7] has divided them into project specific and generic requirements. Generic requirements include items that apply to

existing systems, but are stretched by scalability and administration complexity, and items that are new to highly distributed systems, such as remote delegation and distributed authorization. The generic requirements can be analyzed from the point of view of the main stakeholders: *users* and *resource providers*.

According [7] the **Users' requirements** are the followings**:**

- *limits*: to be able to place limits on the overall amount of resource consumed by particular groups or users;
- *speed*: to introduce new users or groups into a system quickly;
- *flexibility*: about the privileges provided to a group or individual;
- *privacy:* to constrain the flow and use of private data in a system, including data used for authentication;
- *security:* to be able to set up a Grid infrastructure with agreed security features (e.g. only allow data to flow to certain processing sites);
- *delegated action*: to allow the system to carry out a range of functions for the user when the user is not present.

The **resource managers' requirements** are the followings [7]**:**

- to be assured that their system is safe from software run by remote users;
- to ensure that their system is not misused, for example as a platform to attack other systems;
- to be able to charge for services;
- to have a chain of accountability for actions that allows technical and administrative investigation and resolution of problems.

Globus Toolkit designers [8] attempted to address some of these requirements, but have had to balance early delivery against security features, which has lead to several heavily cited issues such as: mapping of global to local user IDs, management, firewalls, proxies, uniformity. These concerns will be addressed in a more general matter in the next section.

The common factor in these problems is **scalability** and **flexibility,** particularly in authorization and policy management, the use of a user's identity as the basis for delegation in distributed systems has venerable roots in existing security practice. However, it is the fundamental source of the cited scalability and flexibility problems. One must accept that in the case of the Grid, it will be impossible to base authorization on individual user identity. In this case it is straightforward to group users into roles. This scheme uses the idea of **federation** to group objects with similar properties. As a result the problem of (users X machines) is factored into manageable parts. The concept of federation can be applied more generally than identification and authorization, to solve a wider range of security issues:

- limit the number of bipartite agreements in which participating resource providers would need to engage
- allow security (and other QoS) properties to be offered by different **distributors**, allowing the tailoring of security to users' needs.
- the notion of a session pseudonym can be extended to a **ticket (**or capability), allowing a user federation to establish arbitrary agreements with a distribution federation about the rights that may be granted by any particular ticket.

From the Web Service perspective the *WS-Federation* standard [9] describes how to use *WS-Trust*, *WS-Security* and *WS-Policy* together to provide federation between security domains. As a conclusion one might say that it is possible to factor Grid security requirements using the federation concept which is capable of solving many of the problems perceived in present day Grid. The Grid can also be secured by using network technologies such as: *IPSec* and *IPv6***,** *VPN* **(**Virtual Private Networks), *Firewalls* or *GSS-API* can also help the Grid to become more secure. Next we will address some *network issues***,** some of them mentioned when we discussed that Globus has left open some issues concerning amongst others firewalls, proxies.

One of the requirements for Grid networks which have not been addressed in this section is the access control and isolation of the traffic flowing through the network [10]. One of the most important technologies for access control is through *firewalls* where suspicious traffic is controlled and resources are protected. The main purposes of firewalls are: *security, enforcing policies and auditing*.

When dealing with networks we must also talk about routing. It is one of the most important components of any networking infrastructure. Routing mechanisms and protocols are designed to route packets through the network in a resource efficient manner. They also take care of network failure, network level congestion, etc. One of the main ways of attacking a router is by table “poisoning”. It is a type of attack where the routing updates are maliciously modified by the adversaries resulting in creation of wrong routing tables. Table "poisoning" can also lead to DDoS.

1.3 WS- Security Specifications*

In the previous paragraphs we have briefly mentioned some WS-* security specifications such as **WS-Trust** or **WS-Federation**. In the following lines we will describe more such standards from the same security family.

WS-Security - is a communications protocol providing a means for applying security to Web Services. The protocol contains specifications on how integrity and confidentiality can be enforced on Web Service messaging. The WSS protocol includes details on the use of SAML and Kerberos, and certificate formats such as X.509. It also describes how to attach signatures and encryption headers to SOAP messages. In addition, it describes how to attach security tokens, including binary security tokens such as X.509 certificates and Kerberos tickets, to messages. WS-Security incorporates security features in the header of a SOAP message, working in the application layer. Thus it ensures end-to-end security.

WS-SecureConversation [11] - is a Web Service specification, created by IBM and others, that works in conjunction with WS-Security, WS-Trust and WS-Policy to allow sharing of security contexts. The purpose of WS-SecureConversation is to allow secure conversations between sites using Web Services for communication.

WS-Federation [9] - defines mechanisms for allowing disparate security realms to broker information on identities, identity attributes and authentication.

WS-Policy [12] - is a specification that allows Web services to use XML to advertise their policies (on security, Quality of Service, etc.) and for Web service consumers to specify their policy requirements. It represents a set of specifications that describe the

capabilities and constraints of the security (and other business) policies on intermediaries and end points (for example, required security tokens, supported encryption algorithms, and privacy rules) and how to associate policies with services and end points.

WS-Trust [13] - specification and OASIS standard that provides extensions to WS-Security, specifically dealing with the issuing, renewing, and validating of security tokens, as well as with ways to establish, assess the presence of, and broker trust relationships between participants in a secure message exchange. Using the extensions defined in WS-Trust, applications can engage in secure communication designed to work within the Web services framework.

WS-Privacy is related to the privacy preference specification for Web Services.

Finally, **WS-Authorization** is managing policies about Web Services.

1.4 Web Service Security

In Service Oriented Architecture (SOA) technology, Web Services are emerging as the enabling technology that bridges decoupled systems across various platforms, programming languages, and applications. However their introduction comes with a new level of complexity added to the environments where they are deployed. We have previously seen the security requirements for a Grid and in this section we will briefly address them with regard to WSs.

Paper [14] states that the goal of securing a WS can be divided into three sub-goals:

- Providing mechanisms and tools for securing the integrity and confidentiality of messages as well as the guarantee of message delivery.
- Ensuring that the service acts only on message requests that comply with the policies associated with the services.
- Ensuring that all information required by a party in order to discover and use services is correct and authentic.

It is also stated that the ultimate goal of WS security standards is to make interoperable different security infrastructures and to reduce the security management cost. This task is however not trivial as it requires for the different security standards to provide a common framework and protocols for exchanging security information.

The threats that could arise in the context of WS have in general counterparts in the form of Grid threats and can be grouped as follows [14] :

- Message alteration: message information is altered by handling the information created by the sender of the information and mistaken by the receiver for the sender's intention. This type of attack is easily performed due to the use of intermediaries and transformation mechanisms in Web Services.
- Confidentiality: makes information within the message visible to unauthorized users.
- Falsified messages: occurs when an attacker constructs false messages and sends them to a receiver who believes them to have originated from a party other than the sender.

- Man in the middle: the term is applied to a wide variety of attacks that have little in common except for their topology. This type of attack occurs when a third party poses to be the other participant to the real sender and receiver in order to mislead both of them. Software engineers have to examine their developed applications on a case-by-case basis for susceptibility to anything a third party might do.
- Principal spoofing: in this threat a message is sent in a form in which it appears to be from another principal.
- Forged claims: a message is sent in which the security claims are falsified in an effort to gain access to otherwise unauthorized information.
- Replay of message parts: involves a message to be send and which includes portions of another message in an effort to gain access to unauthorized information or to cause the receiver to take some action. This technique can be applied in a wide variety of cases. All designs must be carefully inspected from the perspective of what could an attacker do by replaying messages or parts of messages.
- Replay: means that a whole message is resent by an attacker.
- Denial of service: a form of an amplifier attack where the attacker does a small amount of work forcing the system under attack to do a large amount of work. As a result the attacked system could fail completely or provide a degraded service.

The main objective of securing a Web Service is to create an environment where message-level transactions and business processes can be conducted securely in an end-to-end fashion.

The paper [15] proposes a solution which implements message-level security (MLS). It is based on the observation that much of the MLS-related overhead comes from the XML nature of the digital signatures and encryption. The authors of the paper came up with the idea of using S/MIME (Secure/Multipurpose Internet Mail Extensions) over HTTP (the protocol used by Web Services for transferring messages) for transferring SOAP messages. The MLS is usually handled using signatures and/or encryption in the XML content of SOAP messages, using W3C standards XML-Encryption and XML-Signature. The authors have also created two implementations of the S/MIME-over-HTTP message level security one for the gSOAP toolkit and other for the Apache Axis Toolkit. Reports on the performance of Web Service security based on XML-Signature and XML-Encryption, as detailed in [16] and [17], found it to be slow when compared to SSL, mainly due to very expensive canonicalization of XML. However SSL has also some shortcomings, namely it cannot provide digital signatures and non-repudiation.

1.5 Legal Issues in Secure Grid Computing

When dealing with Grid security one must also take into consideration the legal issues that might appear. As a result a legal framework is necessary in order allow safe transactions among the organizations and individuals that dynamically form the Grid.

According to [18] privacy and intellectual property requirements are magnified in the globalized Grid environment due to the distributed nature of the data and computing resources. In a Grid data is transferred from users and resources and vice versa.

During these transfers data is either cached or duplicated so that the best transfer times are to be achieved with minimal costs. Paper [18] also states that Copyright Law protection has proved to be ineffective in protecting the vendor software in Grid environment therefore software license agreements are utilizing contractual provisions that are more restrictive than copyright law in order to protect software usage.

Most of the software and middleware used in the Grid is based on open source licensing. One of the most commonly used **licenses** of this type is **GPL** (GNU General Public License). Any software licensed under it can be provided at a very low or no cost at all. GPL licensees are restricted from using GPL-covered code in proprietary derivatives in order to preserve access to the software for all users. Through this the Grid discourages any commercial use because it is difficult for vendors to produce proprietary software.

Globus on the other hand is using **BSD licenses** (Berkeley Software Distribution) that allow individuals or organizations to make modification or enhancements to the code without contributing that code back to the open source community and thus encouraging a commercial support from the industry.

Currently many companies sell Grid software "per processor" that makes Grid solutions not affordable since Grid computing is based on a dynamic resource allocation. The issues arise because the Grid user is paying full price for software that is not fully exploiting the machine that is shared in a virtual environment. Instead they would need a dynamic pricing that will reflect their actual Grid usage. These issues should be addressed by new licensing schemes and there should be clearly stated whether they should be considered to be a lease agreement by the client or a service agreement by the Grid provider.

Data privacy issues are becoming critical as more and more industry and academic organizations are connecting their resources and deploying them into the Grid. Data is usually protected by data protection laws but this nonetheless increases concern of the clients regarding the misuse of their information. As mentioned in the paper [18] there is a need to balance competing interests of privacy inside the Grid environment. The Grid client has the right to access his data held by the Grid broker and where appropriate to have such data corrected or erased. A controversial point concerning the Grid broker's operation is that, while the user can request and obtain access to his data from the Grid broker, it is not clear whether this access includes revealing information regarding where the data reside and other information regarding the Grid nodes that store and process the data.

Software licenses used in a Grid environment should constraint the Grid provider and the Grid nodes to only make authorized use of the software. The contract should restrict software modifications and at the same time should also clarify that the necessary exchange of data within the Grid environment for efficient computing is not misuse of data under applicable data protection laws.

2 Globus Security

The previous section introduced the most important security issues that must be addressed in the context of computer-to-computer communication in general with emphasis on the Grid security issues. Globus Toolkit GSI (Grid Security

Infrastructure) component tries to address these issues and to offer solutions for secure communication. While basic mechanisms are provided, a satisfactory security level in Grid applications can only be achiever if the security mechanisms are used wisely and with extreme caution. This section is an introduction of the basic mechanism offered by Globus [19], [20].

The central concept of the GSI is the security based on X.509 security certificates and proxy security certificates derived from the X.509 certificates. A Grid Virtual Organization(VO) can thus be easily created based on trusted Certificate Authorities (CA) that signs the certificates. Authentication with a VO can be achieved through:

- X.509 certificates
- username and password
- anonymous authentication

It must be noted though that the most popular and preferred authentication method is based on security certificates. The information that is usually stored within a security certificate are:

- User's name
- The public key
- The identity of the CA signing the certificate.
- The digital signature that assures that the certificate was not altered

While the public key is sent over the wire, the private key is kept secret. The tuple formed by the certificate and the private key are usually referred, in the context of public key encryption using security certificate, as the credential of the user.

Secure communication may be used with different purposes. It may be intended to guarantee message integrity or it may also need to assure the confidentiality of data transmitted over the wire. For this reasons, GSI supports several secure conversation schemes. The type of security mechanism enforced through mechanisms supported by GSI is present at two levels of communication:

- Transport level: TLS encryption applied to all communication that is sent over the wire. This encryption level assures integrity and potentially authentication of the communicating entities.
- Message level: encryption is applied only at message level. If only integrity of messages is required, the messages are sent as plain text (i.e. not encrypted), and digital signatures are attached to the messages to assure integrity. If confidentiality is also required, messages are sent over the wire as encrypted text. In either case, the SOAP envelope is still not encrypted.

Message level exchange with GSI is an implementation of the WS-SecureConversation and WS-Security OASIS standards. The former is used when multiple calls must be issued between the client and the remote service. Secure conversation creates in the initialization phase a security context that is reused with later calls. This way, the computational cost introduced by public key encryption is reduced. This type of conversation makes also possible delegation of credentials from one entity to another.

Authentication is of great importance not only to assure that the originator of messages is who it claims to be. Because messages represent, in Service Oriented

Architectures, request of services and responses to the requested services, authentication assures that proper authorization schemes can be applied. As we will see further on, authorization plays an important role in the overall security mechanisms implements by GSI.

Communication between two entities is done with mutual public key exchange. In the process of communication, the two entities can use one of the following communication schemes:

- Mutually authenticated: when both entities present authentication information
- Server-side only authentication: when the server presents authentication information but the client does not provide any information
- Completely unauthenticated: when no authentication is provided.

With Globus GSI, the communication that takes place between a client and the Globus container storing Grid services can fall only in one of the first two categories because the container always provides authentication information.

To sum up, the most important security features that he GSI provides are:

- command-line tools;
- Java classes to easily integrate security with services;
- Security components.

2.1 General Security Options

The heart of a secured Grid services is the security descriptor describing the security configurations applicable for a service or for the invoking client. The security descriptor file can be used to configure the security options at server side at client side. Some of the security options that can be specified inside the security descriptor file can be applied for both client and server side levels, while several options are specific either to client component or to server components. The run-as identity and security features applicable for the Globus container, for the services that run in the container and even for the resources managed by services can be fine-tuned using the security descriptor. Every such level can be configured to use different authentication mechanisms. Also, the run-as identity dictates the type of access that the service is entitled to. This feature is important in the context of credential delegation when a service may be allowed to execute actions on behalf of the invoking client.

The general structure of the security descriptor is:

```
<securityConfig xmlns="http://www.globus.org">
      <-- security options -->
</securityConfig>
```

Credentials, i.e. the tuple formed by the files that hold the X.509 certificate and the private key, may be specified within the security file, using the `credential` node :

```
<securityConfig xmlns="http://www.globus.org">
      <credential>
            <key-file value="key_file"/>
            <cert-file value="certificate_file"/>
      </credential>
</securityConfig>
```

In the above example, the `key_file` and the `certificate_file` values specify the path to the key files that must be used when communicating to third parties.

NOTE. Credentials may also be specified at the server level because the Globus container, the service or the resource may have their own identity. The scope of the identity applicable at the server side dictates that if credentials are not specified for a certain level, these are assumed from the higher lever. The way that identity is resolved is presented in the following figure:

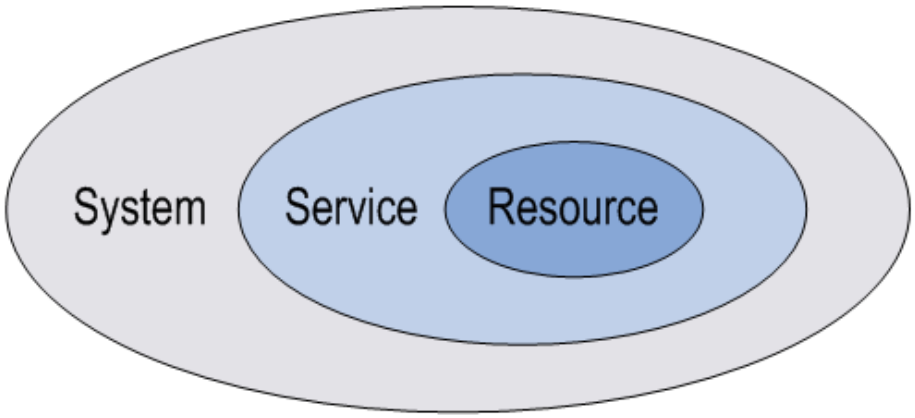

Fig. 2 Identity scope at server side

When a proxy certificate must be used, for example due to security or commodity reasons, instead of the X.509 certificate, the `proxy-file` node can be used to specify the corresponding location of the proxy certificate file:

```
<securityConfig xmlns="http://www.globus.org">
      <proxy-file value="proxy_file"/>
</securityConfig>
```

For both client and server security descriptors, the node `reject-limited-proxy` can be used to reject time limited authentication proxies from the communication partner:

```
<securityConfig xmlns="http://www.globus.org">
      <reject-limited-proxy value="true"/>
</securityConfig>
```

2.2 Server Side Security Options

In order to specify the security options that apply at container level, the WSDD file associated with the container must contain a parameter definition that declares the location of the security descriptor:

```
<parameter
      name="containerSecDesc"
      value="path/to/global/security/descriptor.xml"/>
```

Here, the attribute `value` should be a relative path to the file containing the security rules that apply.

Secure Conversation establishes a security context that is valid for predefined period of time. The node `context-lifetime` can be used to specify the lifetime of the security context:

```
<securityConfig xmlns="http://www.globus.org">
      <context-lifetime value="seconds"/>
</securityConfig>
```

Denial of Service is a type of attach that is meant to crash the remote service, preventing the clients of such services to access them. Usually, this type of attack involves requests to the service that require small computational effort for the attacker and a large computational effort on the service side. Encrypted messages prevent an attacker to modify the messages, but this does not stop them from issuing a valid call for many times. GSI implement a mechanism that keeps track of the latest messages that were received at the server-side. If the same massage is replayed more than once in a predefined time interval, only the first message is considered and the rest of them are discarded. The replay time interval in which the fake messages are dropped can be controlled through:

```
<securityConfig xmlns="http://www.globus.org">
      ..
      <replay-attack-interval value="milliseconds"/>
      ..
</securityConfig>
```

One method to specify authorization for a remote client to access a service is to create a Grid-map-file that contains a list of the authorized users. To specify that a certain Grid map file should be considered, the security descriptor must contain a reference to its location:

```
<securityConfig xmlns="http://www.globus.org">
      <gridmap value="Gridmap_file"/>
</securityConfig>
```

As described earlier, the authentication methods available with the GSI implementation are those specified by WS-Secure Conversation, WS-Security and TSL standards. The required type of authentication that must be used for a certain service or for accessing a certain resource can be specified within the security descriptor using the XML nodes:

- `<GSISecureConversation/>`
- `<GSISecureMessage/>`
- `<GSITransport/>`

The above nodes must be specified within an auth-method node. The following example states that the WS-Secure Message is used:

```
<auth-method>
      <GSISecureConversation/>
</auth-method>
```

while the following section states that both Secure Conversation and TLS encryption must be used in conversation:

```
<auth-method>
      <GSISecureConversation/>
      <GSITransport/>
</auth-method>
```

If authorization method must be tuned at operation level, the security descriptor may state the method for which the rule applies to. The following example states that Secure Conversation is applicable to the operation `operation_name`:

```
<method name="operation_name">
      <auth-method>
            <GSISecureConversation/>
      </auth-method>
</method>
```

The methods of authentication presented so far specify whether security enforced at message level is individual for every call or a security context must be created. They also control if the encryption should be applied on the entire SOAP message or not. Additionally it is possible to specify if communication should ensure that message integrity is preserved or that privacy is also required. For every message level encryption communication scheme there are two possible alternatives:

- integrity: ensures that the message received was not corrupted by a third party. This requires attaching to the message a digital signature based on the public key of the sender. It is marked by adding an empty `<integrity/>` child node to one of the `<GSISecureConversation/>`, `<GSISecureMessage/>`, `<GSITransport/>`
- privacy: ensures that the message cannot be read and cannot be corrupted by a third party. This involves an encryption of the original message combined with a digital signature ensuring integrity. It can be requested by adding an empty `<privacy/>` child node to one of the `<GSISecureConversation/>`, `<GSISecureMessage/>`, `<GSITransport/>`

It must be noted that enforcing privacy will automatically provide integrity of the messages.

In the interaction between the client and the server, the server side component participating in the conversation is always authenticated. The identity of the server side components is mandatory. As depicted in Figure 2, identity specified at fine grained level has priority over the higher level specified identity. This means that the identity options specified at resource level will have priority over the ones specified at service level, which in turn, overrides the one specified at the Globus container level.

An operation level security descriptor may specify that a certain run time identity must be considered. The available identities that may be user are the identity of the container, the identity specified at resource level or the identity of the client. The identity assumed at run-time is also referred to as the *invocation subject*. The *invocation subject* is particularly relevant in the credential delegation process where it will be used in relation with third party services. The valid choices are: `<caller-`

`identity/>`, `<system-identity/>`, `<service-identity/>`, `<resource-identity/>`. For example, the following specifies that the method should use during execution the resource level identity:

```
<method name="divide">
      <run-as>
            <resource-identity/>
      </run-as>
</method>
```

If the `run-as` node is not a child of a `method` node, it indicates that the run-as directive is the default behavior that applies to operations for which the `run-as` property is not specified.

2.2.1. Adding a security descriptor to a service

Globus container uses a predefined security descriptor that specifies the security rules that apply to the container. Changing the security must be done by adding to the service deployment descriptor a reference to the security descriptor. An example is shown below:

```
<service
      name="examples/security/first/MathService"
      provider="Handler"
      use="literal" style="document">
      <!-- Other parameters -->>
      <parameter name="securityDescriptor"
            value="path/to/security/descriptor.xml"/>
</service>
```

2.2.2. Adding a security descriptor to a resource

In order to add security constraints at resource level, the resource implementation must meet several requirements. Since resources are created dynamically, the link to the specific security descriptor file is possible using the GSI Java API at design time. The class implementing the resource must also implement the *SecureResource* interface. This interface requires an implementation for a method that returns an in-memory representation of the security descriptor.

A more elaborated alternative is to alter the security settings using only the GSI API. This alternative does not require that a security descriptor is specified but the process of altering the security settings using only the GSI API is far more complicated and represents a less flexible approach.

2.3 Client Side Security Implementation

Client side security can be specified using the provided GSI API. To apply security settings to a communication stub of a remote port type exposed by a Grid service, several stub properties must be set/altered. The above code obtains a reference to a port type stub and modifies its properties so that Secure Conversation with no authorization is used:

```
stubObject = locator.get*Port(endpoint);
((Stub)stubObject)._setProperty(Constants.GSI_SEC_CONV,
```

```
                                    Constants.ENCRYPTION);
((Stub)stubObject)._setProperty(Constants.AUTHORIZATION,
                          NoAuthorization.getInstance());
```

The first line is meant to retrieve the stub object of the remote port type. The stub object reference properties are further configured to obtain the desired results.

Because the above example does not use encryption at transport level, the client will be able to interact only with aGlobus container that does not use TLS encryption, started with the additional *–nosec* option.

The above example specifies the security policy that the client uses explicitly adding security properties to the stub of the remote portType. This approach is not efficient when the same client must be used to call services with different security policies. A more flexible alternative is to specify an external security descriptor.

Adding security descriptor file to alter the security options of the client requires that the location of the descriptor file is provided:

```
String secDescFile = "path/to/security_descriptor.xml";
((Stub) stub_object).
      _setProperty(Constants.CLIENT_DESCRIPTOR_FILE,
      secDescFile);
```

Secure message communication with integrity may be achieved at client side at design time using the API provided by GSI Java package or it can be set at run time through a security descriptor. Using Java code:

```
((Stub) stub_object )._setProperty(
      Constants.GSI_SEC_MSG,
      Constants.SIGNATURE);
```

Using a security descriptor that is loaded as shown above:

```
 <securityConfig xmlns="http://www.globus.org">
      <GSISecureMessage>
            <integrity/>
      </GSISecureMessage>
</securityConfig>
```

To achieve Secure Message encryption (i.e. Secure Message with privacy) the client must execute a more complicated initialization procedure if the design time initialization approach is considered. When using Secure Conversation the task of exchanging public keys between the client and the server is done automatically. Secure Message procedure does not involve a communication procedure beforehand sending the actual call. Thus, to use Secure Message, the client must already have certificate of the container that it's going to interact to.

The corresponding code is:

```
String pathToPublicKey = "/etc/Grid-ecurity/containercert.pem";
Subject subject = new Subject();
X509Certificate cert =
     CertUtil.loadCertificate(pathToPublicKey);
EncryptionCredentials serverCred =
      new EncryptionCredentials(
```

```
            new X509Certificate[] { cert });
      subject.getPublicCredentials().add(serverCred);
      // Set stub options
      ((Stub)stub_object)._setProperty(
            Constants.GSI_SEC_MSG,
            Constants.ENCRYPTION);
      ((Stub)stub_object)._setProperty(
            Constants.PEER_SUBJECT,
            subject);
```

Fortunately, using Secure Message with privacy is easier when using an external security descriptor that is loaded at run time. The security descriptor must contain, among the other security specifications, the path to the certificate of the remote container:

```
<GSISecureMessage>
      <privacy/>
            <peer-credentials
            value="/path/to/container/certificate.pem"/>
</GSISecureMessage>
```

At the client side, using Transport encryption can only be specified at design time, and not through a security descriptor. For integrity the properties of the stub must include:

```
((Stub) stub_object )._setProperty(
      Constants.GSI_TRANSPORT,
      Constants.ENCRYPTION);
```

while for privacy the property of the stub must be set to:

```
((Stub) stub_object )._setProperty(
      Constants.GSI_TRANSPORT,
      Constants.SIGNATURE);
```

A *static* section of the client class must also declare that this class is using transport level security through the section:

```
static {
      Util.registerTransport();
}
```

2.4 Authorization

As described in the previous section, the access tot resources remotely available raises two issues. The first, a technical issue, regards the mechanism to specify that a certain user is allowed to access a certain service. The level of access has in subsidiary a more philosophical problem. It must be decided if the user can be allowed to access the functionality. Security policy, legal issues and data privacy are the most important topics that drive the authorization scheme used.

Simply put, authorization refers to the matter of who is authorized to perform a certain task.

Available options for enforcing **Server-Side Authorization** at service level are:

1. None: This authorization type is marked using `<authz value="none"/>` declaration. Practically no authorization policy is used;

2. Self: A client will be allowed to use a service if the client's identity is the same as the service's identity. In this case the authorization node should have for the `value` attribute the value `"self"`

3. Gridmap: authorization is achieved by searching the identity of the client in a predefined gridmap file. The gridmapf file contains all the identities of the clients that are authorized to access the service. The security descriptor should contain an `authz` node with its `value` attribute set to `"gridmap"` and a `gridmap` node with and attribute `value` specifying the location of the gridmap file:

```
<authz value="gridmap"/>
<gridmap value="location/to/gridmapfile"/>
```

4. Identity authorization: The client of the service should have a pre-specified identity. The `value` attribute should have the value set to `"identity"` and the WSDD file should declare a parameter that states the identity of the allowed user:

```
<parameter name="idenAuthz-identity"
      value="/O=... /OU=.../CN=..."/>
```

5. Host authorization: this type allows defining a certain authorized client host. The `value` attribute should be set to `"host"` and the WSDD should declare a parameter that specifies the host:

```
<parameter name="hostAuthz-url"
      value="host-name.com"/>
```

6. SAML Callout authorization: This authorization scheme is achieved by calling an external authorization service that uses SAML(Security Assertion Markup Language).

Client side authorization refers to what services the client is entitled to access. The options that can be used are : "none", "self", identity, host. The authorization types are similar with the ones specified at server side.

GSI provides an infrastructure to easily plug in our own authorization mechanisms. See the one implemented in [21].

2.5 Credential Delegation

The classic example for credential delegation is the one in which a caller, having the identity *A*, is requesting a service from another entity identified as *B* that has to make additional calls to third party services to be able to solve the call originated from *A.* If the request sent to by *A* to be *B* imposes that *B* has to call a third party entity *C*, with the authorization issues considered, the questions is what is the identity that *C* should see the call coming from. Since *B* is working for *A*, the most logical scenario is that *C* should consider the call as coming from *A.* In order to achieve this behavior, delegation of credential must be used. Through credential delegation, *B* receives the right to act on behalf of *A*.

Credential delegation may be achieved only when WS-SecureConversation is used. Since private keys corresponding to the X.509, proxy certificates must remain secret, in the scenatio explained above, *B* creates a public key that is validated by a digital signature of *A.* The new created public key will be used in all interactions between *B* and *C.*

We have explained in a previous sub-section how the *invocation subject* can be modified to specify a different identity that the default one specified at container level. In our case, for credential delegation, it is important that the identity of the service is set to *caller identity.* This way, in accessing a third party service, the identity that will be used for authorization purposes will be that of the client *A,* and not the identity of *B.* For this mechanism to work, the *invocation subject* identity must also be assumed as own identity of *B* in every conversation with *C.* This can be specified using the GSI API:

```
SecurityManager.getManager().setServiceOwnerFromContext();
```

For the delegation process to be functional, the client *A* must also specify that a credential delegation must be done to *B*. This can be also specified as a property of the communication stub at the client side by GSI Java code:

```
((Stub) stub_object)._setProperty(
      GSIConstants.GSI_MODE,GSIConstants.
      GSI_MODE_FULL_DELEG
      );
```

or directly in the security descriptor:

```
<GSISecureConversation>
      <integrity/>
      <delegation value="full"/>
</GSISecureConversation>
```

The values of the `value` attribute may be full or `limited`. Depending on the security configuration at server side, a service may be configured to accept only full delegation certificates. It the limited certificates delegation is used at client side, the delegation is considered invalid.

Another convenient method of delegating credentials is to use the delegation service implemented by Globus. It acts like a certificate storage facility. The client must specify that it want to delegate credentials to a specific service located in the same Globus container as the delegation service. Then, the service that wants to use the delegated credentials must contact the delegation service and retrieve the proper certificates.

3 The Symbolic Grid Service Architecture

Exposing functionality of CASs as Web and Grid services requires finding solutions for issues such as the right technology to be used for interfacing with external world, a data model that allows interoperability and the way that these systems can be enabled to access and to be accessible in a distributed environment. With such services available the next step is to create a computational infrastructure that is able to

combine the functionality of available services to solve compound problems by orchestrating remote services.

The architecture described in the following tries to offer an answer to the problems stated above by integrating, on one hand, solutions already present in research and software industry and, on the other hand, to offer specific solutions to the domain of symbolic Grid computing. Since SOA, and Grid services in particular, seems to be the solution for large-scale computations, the efforts were concentrated in this direction.

CAS application specialists aim to solve problems that require high computer power unavailable in the context of a local machine. Since CASs focuses on particular areas in symbolic computing, another goal is to enable integration of functionality of different CASs into Grid architectures.

Possible roles that a CAS can play in the interaction with other systems are: partner in CAS-to-CAS interactions, client in CAS to Web/Grid service interactions, and CAS server as its functionality is exposed as Web/Grid services. The aim of developing a distributed system as the one presented here motivates the particular interest accorded to the last two roles. The first one is better handled by the SymGrid-Par component [22].

To enable the functionality mentioned above two main components were identified. An important requirement is to enable the user to discover and to access the functionality mentioned above in a transparent way.

Once the symbolic services are available, one may want to model complicated scenarios that require the ability to combine provided functionality. CAGS [23] offers the ability to access remote services. The functionality of CAGS is demonstrated in the context of the GAP system through a special GAP [24] package. The tool is conceived as an interface between the CAS or any other type of software system, and the computing infrastructure of Web and Grid services.

Since the main tools for symbolic computing are CASs, an important computation power gain is to enable those systems to be remotely accessible as computational servers. In this context, user transparency means that remote CAS functionality must be available on a per function basis. This solution must take into account the security and legal issues that are generated by exposing this functionality. These requirements were met by the CAS Server component [25] but further security mechanism must be enforced not only at CAS server level but also with the component responsible for orchestrating these servers.

CAS Servers assure that the functionality provided by CASs installed on the server can be invoked using a Web/Grid service interface. The control over the exact functionality that is being made available for remote calls is controlled through a local registry that verifies that a certain CAS is installed on the server and that a certain function is available to be invoked.

Having as a ground basis the ability to expose CAS functionality as Web or Grid services, it can now be imagined combining functionality of several CAS servers. Services can be composed to serve for complex scenarios. Solving a complex problem

can be usually achieved by combining results of smaller problems that compose the original problem.

The ability to execute mathematical workflows is based on the functionality of several software components. The main components include a client component at the client side, and a server side system that includes a client manager component, a workflow engine needed to execute the workflows and several CAS Servers (Figure 3). More details about the functionality of these components as well as implementation details are offered further on.

As a first step, the system implements the functionality that allows executing simple workflows that follow a given pattern. The system is thus able to execute scenarios where computing a final result is based on computing results for several subtasks.

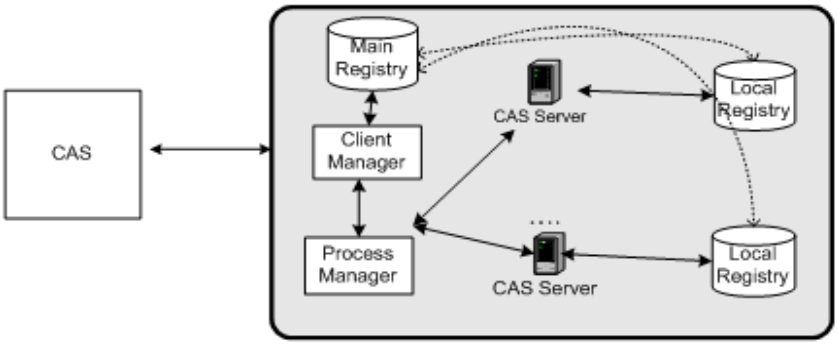

Fig. 3 Symbolic Services Architecture

The computational request formulated by the client within the client CAS component is translated and sent, through a Web service interface, to the server. At the server level the request is managed by the client manager component. Since there is no available mechanism to choose dynamically the right CAS that should be used for computing a certain task, the user must indicate the CAS to be used for every task.

Several steps are needed to transform the workflow described within the CAS to a format that complies with a standard orchestration engine. The workflow that results at the client side is not complete because the client is not, and should not be aware of the URL addresses of the services that will compute the sub-tasks. Thus, the client component sends an incomplete workflow to the client manager component.

The management of the request involves finding the right servers to solve the request, storing information about the status and the result of the request and offer the client mechanisms to obtain results, at a later time, in an asynchronous way. To complete the workflow specification with the necessary details, the client management component interrogates the Main Registry and uses selection algorithms that take into account details regarding the functionality and the load of the CAS Servers.

The information stored in the Main Registry duplicates information retrieved from the CAS Servers integrated into the system. Every CAS Server has a corresponding Local Registry that contains information about CASs and exposed functions.

The research conducted in the Web service composition area is quite vast [26] and resulted in the development of tools that are able to manage complicated workflows. The system is currently configured to be used in conjunction with the ActiveBPEL engine. Once client manager completes the BPEL document, the workflow is deployed into the BPEL engine.

Since interoperability is an important aspect, client-server interactions as well as server-to-server component interactions are encoded in XML format. The system is prepared thus to embed mathematical expressions encoded using the OpenMath standard.

With several changes at the client and component management level, the system can be enabled to offer support for more complicated scenarios, dynamically generated. Among the changes needed, at the client level must be implemented richer functionality for expressing workflows. Also, the client management must be able to dynamically deploy workflows into the BPEL workflow engine.

3.1 General Use Cases

The use cases presented here are intended to capture the functional requirements of the system that we aim to build. While most of the required components are already implemented, integrating these components to a real-life system still require several issues to be solved, and amongst them, security. The main purpose of the system is to enable CAS users to access computational power offered by Grid architectures. In conjunction to this, several administrative functionalities must be offered.

Main types of users that will interact with the system are:

- Regular users
- Administrators

The interaction with the system is carried out by accessing the system using a CAS or by using third party software packages. While access to computations must be offered mainly from within the CAS systems, administrative tasks are likely to be supported through external software packages, including standard functionality offered by software tools such as Globus Toolkit 4.

3.1.1. User functionality

As noted above, the regular user will interact with the system mostly using CAS provided functionality. Security concerns must address security at the client level on one hand and security at the computation site that the user accesses on the other hand. The client must be confident that accessing our system does not represent a security gap. This may be achieved by integrating standard security solutions already provided by Globus GSI and related standards.

The system itself must be protected from the external user actions. While in some cases providing a username and a password may suffice, when interacting with Grid services better security policies must be taken into account potentially by enforcing authentication mechanism such as those provided by X.509 certificates.

Several issues regarding regular user access include:

- Logging and access to remote functionality. For a uniform access to resources, authentication must be handled in a centralized manner. From this point on, all actions of the user in direct relation with the system must be governed by the user access rights dictated by the authorization policy.
- Access rights must state whether a certain user can access certain services, may run computations on certain servers…etc. A good idea may be to restrict users to access remote file systems by preventing direct access to any operating system file system functionality. Since we are using OpenMath objects to describe SCSCP calls, we have to deal with situations that expose security gaps described using OpenMath (if applicable).
- Treat software license related issues.

CAGS user scenario

CAGS tool offers means to access Grid and Web services. It covers service discovery process, learning the list of operations for services and enabling user to access functionality offered by services. Since access to services is sometimes secure in the case of Web services and it presumes security mechanisms with most of Grid services, CAGS tool takes into account these issues. The solution is though an ad-hoc one in the sense that it does not offer a centralized solution. General patterns of using CAGS are:

```
1. Enable needed security features ( optional )
2. Get the list of Grid/Web services hosted at a certain address
3. Find out the signatures of operations exposed by a certain
   service
4. Call the desired operation
```

and for general access to services:

```
1. Enable security features ( mandatory )
2. Request service :
   2.1 Launch a Globus Job using WS GRAM
   2.2 Transfer files from one machine to another using RFT
```

for special Globus Toolkit 4 services. Enabling security features stated above refers to providing a *proxy certificate* to be used in conjunction with those services. The management of proxy certificates is achieved currently directly by the user. A plus of this approach is the possibility of using different certificates depending on the service the user wants to access. The drawback of this approach is that the user must be aware of such proxy certificates and needs to deal with them.

Accessing workflow functionality

While security mechanisms related to this feature are not implemented yet, it is expected that the interaction with the composed service resulted from a workflow deployment is similar to accessing a regular Web service. The deployment step requires that the user describes the workflow to be executed. As a consequence of this, the user is not to add new workflows. The general use case for workflows may be abstracted in several steps:

```
1. Describe the workflow at client side
2. Deploy the workflow and start its execution
3. Obtain the results at a later time
```

```
(Optional)Obtain real time information during the execution of the
workflow.
```

The user specific access rights policy must control the types of information that is provided to the user. As an example, providing information about the host that carry out the computation may represent a security risk.

3.1.2. Administrative functionality

Administrators of the system must be provided with full functionality that assures management of the system. Management of the system imposes using several external tools and software packages. Among the facilities that an administrator must have we mention:

- Access to computational logs that enable administrators to learn information about regular user actions. This may include information about logging history, tasks and related information
- Management of access policies

Access management to CAS Server services

The CAS Server component allows exposing CAS functionality as Web/Grid services through a generic operation. The function that needs to be called and its parameters are supplied by the client of the service using OpenMath objects. A basic security mechanism is implemented by a local registry. This registry contains information about the CAS systems that are installed on the server and contains information about what functions are available to be called by a remote client. This way, on a local basis, it can be controlled what kind of operations can be accessed.

A software package that is already implemented offers an administrator the possibility to edit information within registries. The informational structure of the local registry system permits storing information regarding the CASs installed:

- The name, version, location path for CAS
- For every CAS, the methods exposed and their signature, package to be loaded and a short description

This system can be integrated with a larger security scheme that could enable access on a per user basis.

4 Secure Symbolic Components Composition Architecture

In this section we present the design of our solution for enabling secure access to symbolic computing components. Our design is based on the system architecture presented in Section 3.

Our main goal is to allow the users to use our system in a secure manner. Since our symbolic computing components are exposed as Grid Services running in a Globus Toolkit container, all security features are provided through the Grid Security Infrastructure (GSI). Although, GSI implements three security mechanisms (GSI Secure Message, GSI Secure Conversation and GSI Transport), we have chosen to use

only GSI Transport as it suffices our needs. As transport level security in GSI uses public key cryptography, it guarantees privacy, integrity and authentication.

For authentication purposes, each user must have an X.509 certificate. This certificate could be stored on the user desktop system, but this solution makes them vulnerable to theft by keystroke loggers, trojans or viruses. Instead of this, we have chosen to use the MyProxy [27] online credential repository. MyProxy is manages X.509 security credentials (private keys and certificates) and allows proxy generation from stored credentials via a TLS secured network connection. Since proxy certificates expire (their usual lifetime is 12 hours), the user must periodically create a new one. The generated proxy is then stored in the repository and is accessible via (username, password) combination, where the password is chosen by the user when he generates the proxy and has the same lifetime as the proxy certificate. Besides its ability to support storing and retrieving user credentials without exporting the private key, enforcing security, another advantage emerges from the mobility support provided by MyProxy.

Therefore, before the user starts accessing the system, he must request the MyProxy server to store his credentials. After that, he will just have to provide to the composition system the username and password needed for fetching a proxy certificate for him from the credential repository (see Figure 4). All communication between client and MyProxy server is achieved through a private (encrypted) SSL channel. The same security measure is applied for communication between the Computer Algebra System or Portal and the ClientManager Grid service.

Further, using the username and password provided by the user the ClientManager will contact the MyProxy server and fetch the user proxy certificate from the server, will create the WS-BPEL workflow and deploy it into the ActiveBPEL workflow engine for execution. The workflow engine needs the user's proxy certificate for encryption and for signing its request for the Grid services to be invoked. Normally, both the ClientManager and the ActiveBPEL engine are installed on the same machine (e.g. a cluster headnode in a Grid), so that the user's proxy certificate, retrieved by the ClientManager, would also be available to the workflow engine.

In order to integrate ActiveBPEL with the Grid Security Infrastructure implemented by Globus Toolkit 4 we have to make several configurations to the engine. ActiveBPEL uses Axis and thus one way to enable the ActiveBPEL engine to run a process using the proxy certificate of the user that requested its execution its through Axis handlers. This means that we need to declaratively add some security handlers into ActiveBPEL's message chains. These handlers are located in the <requestFlow> or <responseFlow> flow of the configuration files for ActiveBPEL.

ActiveBPEL takes on the role of both Server and the Client. When a Client (e.g. Computer Algebra System) calls a BPEL process deployed in ActiveBPEL, the engine plays the Server role, as a service provider. Security checks needs to be enforced whenever a request message is sent to the engine. When a part of the process calls an external service, ActiveBPEL plays the role of a Client to that Service. Thus, messages that originate from ActiveBPEL will need all credentials added prior to being sent. In both cases, the correct handlers must be placed in the message chains.

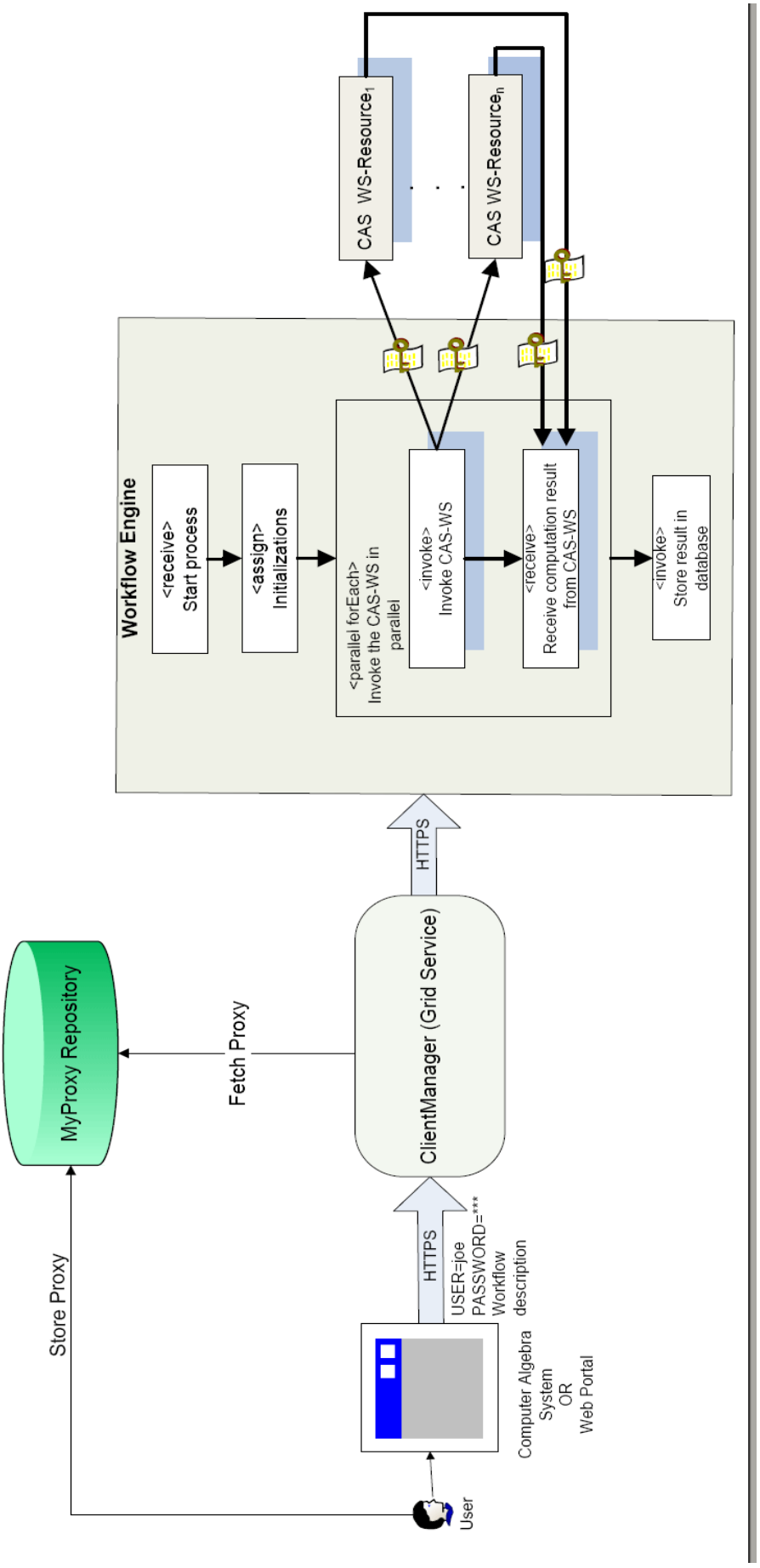

Fig. 4 Secure Symbolic Components Composition Architecture

When a process is invoked, the message is passed to the Globus' security and message handlers added to the Axis chain of handlers and these handlers automatically encrypt and sign messages. The response from the services is also handled by Axis handler chain which decrypts and checks the received message.

When a process is invoked, the message is passed to the Globus' security and message handlers added to the Axis chain of handlers and these handlers automatically encrypt and sign messages. The response from the services is also handled by Axis handler chain which decrypts and checks the received message.

In our implementation of the composition of symbolic components, a user defines a workflow for his computation. During the execution of the workflow a number of Grid services will be invoked. Because for each invocation of a Grid service operation from the workflow engine, a new WS-Resource is created at the CAS Server, and one security requirement is that a user must not be able to control another user's computation as long as it is running, we needed to enforce security at WS-Resource level.

To configure security in an individual resource, we must specify the security requirements in a security descriptor and implement the resource as a Globus Secure Resource. In the security descriptor we must specify that only the user that created the resource is authorized to operate on it and that the resource will be accessed using GSI Transport:

```
<securityConfig xmlns="http://www.globus.org">
      <auth-method>
            <GSITransport/>
      </auth-method>
      <authz value="self"/>
</securityConfig>
```

Enabling security at resource level, assures that only the user that requested the computation will be able to stop it or to pause/resume it or change it in any way. If another user will try to call any operation on the WS-Resource, his access will be denied.

Another approach which gives finer control over the security mechanisms used between the workflow engine and the CAS server involves extending WS-BPEL language to allow us to specify which security method must be used when invoking a Grid service operation, who is authorized to call the service or if credential delegation should be employed. Such approach was introduced for BPEL 1.1 and is presented in [28].

In this second approach the proxy certificate of the user is obtained similarly to the first solution making use of the MyProxy credential repository, but this time the ActiveBPEL engine will be modified such it must be capable of processing additional tags needed for invoking a Grid service operation. For example, the WS-BPEL tag <*invoke*> will have an additional <*security*> tag:

```
<invoke ...>
      <security
```

```
        method="GSITransport |
                GSISecureMessage |
                GSISecureConversation"
        level="privacy | integrity"
        authz="none | self | host | anyString"?
        peer-credentials="filename"?
        anonymous="true | false"?
        delegation="none | full | limited"?
        />?
</invoke>
```

In the example above the security is enforced at operation level, but another possibility would be to enforce it at Grid service partner level, by setting the <security> tag inside the <*partnerLink*> tag, in which case security settings will be the same for all operations invoked on that partner:

```
<partnerLink ...>
        <security
        method="GSITransport |
                GSISecureMessage |
                GSISecureConversation"
        level="privacy | integrity"
        authz="none | self | host | anyString"?
        peer-credentials="filename"?
        anonymous="true | false"?
        delegation="none | full | limited"?
        />?
</ partnerLink >
```

For example, if we want all operations on all partners to be invoked using the GSI Transport mechanism the following settings should be made in the BPEL process:

```
<partnerLink …>
        <security
              method=″GSITransport″
              level=″privacy″
              peer-credentials=″x509up_tmp1001″/>
</partnerLink>
```

In the case when symbolic Grid services are accessed using the CAGS middleware, security is also enforced using the proxy certificate of the user. This certificate could be stored locally on the machine of the user, but it could also be obtained using the MyProxy credential repository. After the user is authenticated, the invoked Grid service will check if the user is authorized to call the operation using a special gridmap file. When using the CAGS middleware any of the GSI security mechanisms could be used, enabling thus delegation and single sign-on.

Also, the access to administrative services is controlled using proxy certificates for authentication and gridmap file, so that it is possible that different users are responsible for administering different sections of the registry (e.g. one person could manage registries at several sites).

5 Conclusions and future steps

Implementing computational systems for real life applications that use as a communication infrastructure open computer networks must address the security

threats present in such environments. While several security standards are already defined, there are still open issues that must be solved. Besides the technical security problem, moral and legal issues must also be addressed.

Globus GT4 middleware provides mechanisms that allow building safe Grid infrastructure by implementing several security standards. Still, these solutions are not versatile enough to provide easy solutions in the context of Grid service orchestration.

Since Grid services are state-full Web services, the most convenient approach for orchestrating Grid services is to use the solution already adopted for Web Service orchestration: executing workflows described using standard orchestration languages. While composing unsecured Grid services can be achieved using already existing workflow engines, there is no such engine that entirely supports GSI security features.

In Section 4 we have identified two possible solutions that will allow us to add security to the system we are building. The former it's easier to implement while the latter has the advantage to enable finer grained security based on the client's identity and their associated authorization rights. In the near future we plan to investigate which of the two solutions that we have identified is better suited for the system we aim to build. As a subsequent task, we plan to address the problem of compatibility between secured Grid services and workflow engines, including the problem of credential delegation.

References

[1] Y. Xiang, W. Zhou,– "Protect Grids from DDoS Attacks," GCC 2004, LNCS 3251, pp. 309–316, 2004.

[2] B. C. Neuman, "Security, Accounting, and Assurance," in *The GRID: Blueprint for a New Computing Infrastructure* (edited by Kesselman and Foster), Morgan, Kauffman Publishers, pp. 395-420, 1999.

[3] A. Chakrabarti, "Taxonomy of Grid Security Issues," in *Grid Computing Security*, ed. Springer, pp. 33-47, 2007.

[4] M. Ahsant, M. Surridge, T. Leonard, A. Krishna, O. Mulmo, "Dynamic Trust Federation in Grids," in Trust Management, Springer - Verlag, vol.3986, pp. 3-18, 2006.

[5] S. Anderson, J. Bohren, et al., "Web Services Trust Language (WS-Trust) v1.1", March 2007, http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.pdf

[6] C. Kaler, Web Services Security (WS-Security) v1.1, Feb. 2006, http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

[7] H. Chivers, "Grid Security: Problems and Potential Solutions," University of York, UK, Department of Computer Science, Yellow Report YCS-2003-354 available at http://www.cs.york.ac.uk/ftpdir/reports/

[8] Globus Toolkit 4.0, www.globus.org

[9] C. Kaler, A. Nadalin, "Web Services Federation Language (WS-Federation) v1.1," July 2003, http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-fed/WS-Federation-V1-1B.pdf?S_TACT=105AGX04&S_CMP=LP

[10] A. Chakrabarti, "Grid Network Security," in *Grid Computing Security*, ed. Springer, pp. 159-192, 2007.

[11] S. Anderson et al., "Web Services Secure Conversation Language (WS-SecureConversation)," Feb. 2005, http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-secon/ws-secureconversation.pdf

[12] A. S. Vedamuthuet al., "Web Services Policy 1.5 – Framework," Sept. 2007, http://www.w3.org/TR/ws-policy/

[13] A. Nadalin et al., "WS-Trust 1.3," March 2007, http://docs.oasis-open.org/ws-sx/ws-trust/v1.3/ws-trust.html

[14] A. Barbir, C. Hobbs, E. Bertino, F. Hirsch, L. Martino, "Test and Analysis of Web Services," in *Test and Analysis of Web Services*, Springer, pp. 395-440, 2007.

[15] O. Krajícek, M. Kuba, M. Procházka, D. Kouril, "Message Level Security For Grid Services Using S/MIME," in *Distributed and Parallel Systems*, Springer, pp. 123-131, 2007.

[16] S. Shirasuna et al., "Performance Comparison of Security Mechanisms for Grid Services," in *Grid*, pp. 360-364, Fifth IEEE/ACM International Workshop on Grid Computing (associated with Supercomputing 2004), Pittsburgh, PA, 2004.

[17] H. Liu, S. Pallickara, G. Fox, "Performance of Web Services Security," *Proceedings of the 13th Annual Mardi Gras Conference*, February 2005, Baton Rouge, Louisiana, 2005.

[18] I. Kafeza, E. Kafeza, F. Wai-Hon Chan, "Legal Issues in Secure Grid Computing Environments," in *Securing Electronic Business Processes*, Vieweg, pp. 448-454, 2006.

[19] Globus GSI, http://www.globus.org/toolkit/docs/4.0/security/.

[20] B. Sotomayor, L. Childers, "Globus Toolkit 4: Programming Java Services," ed. Elsevier, 2005.

[21] F. J. García Clemente, "Distributed Provision and Management of Security Services in Globus Toolkit 4," *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE (OTM 2006)*, LNCS vol. 4276, pp. 1325 – 1335, 2006.

[22] K. Hammond, A. Al Zain, G. Cooperman, D. Petcu, P. Trinder, "SymGrid: A framework for symbolic computation on the Grid," In *Proc. Euro-Par 2007 Parallel Processing,* Lecture Notes in Computer Science, Springer vol. 4641, pp. 447−456, 2007.

[23] A. Cârstea, M. Frîncu, G. Macariu, D. Petcu, K. Hammond, "Generic access to Web and Grid-based symbolic computing services: the SymGrid-services framework," In *Procs. International Symposium on Parallel and Distributed Computing*, IEEE Computer Society Press, pp. 143−150, 2007.

[24] GAP Group, *Groups, Algorithms & Programming*, http://www.gap-system.org.

[25] A. Cârstea, M. Frîncu, A. Konovalov, G. Macariu, D. Petcu, "On service-oriented symbolic computing," In *Procs. International Conference On Parallel Processing and Applied Mathematics* 2007, LNCS, Vol. 4967, Springer-Verlag, pp. 843-851, 2008.

[26] S. Dustdar, W. Schreiner, "A survey on Web services composition," *International Journal of Web and Grid Services*, vol. 1, No. 1, pp. 1−30, 2005.

[27] J. Basney, M. Humphrey, V. Welch, "The MyProxy online credential repository: Research Articles," *Software — Practice & Experience*, Vol. 35 , Issue 9 (July 2005), pp. 801 – 816, 2005.

[28] T. Dörnemann, M. Smith, B. Freisleben, "Composition and Execution of Secure Workflows in WSRF-Grids," *8th IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, pp. 122-129, 2008.