10th IEEE High Assurance Systems Engineering Symposium

CROWN-C: a High-Assurance Service-Oriented Grid Middleware System

Paul Townend[1], Nik Looker[1], Dacheng Zhang[1], Jie Xu[1], Jianxin Li[2], Liang Zhong[2], Jinpeng Huai[2]

[1]School of Computing
University of Leeds
Leeds, LS2 9JT, UK

{pt, nlooker, dcz, jxu} @ comp.leeds.ac.uk

[2]Dept. Computer Science
Beihang University
Beijing, 100083, P.R. China

{lijx, zhongl, huaijp} @ act.buaa.edu.cn

Abstract

Service-orientation is a highly useful means of developing flexible, agile, and dependable software systems, and is a paradigm that has been increasingly adopted into Grid Computing middleware. However, service-orientation brings with it new challenges in the fields of dependability and security that need to be addressed by the High Assurance Systems community in order to provide sufficient support to enable service-based Grid applications to offer non-trivial Quality of Service guarantees. This paper discusses some of the new dependability and security challenges introduced by service-orientation, and for the first time introduces CROWN-C – a Grid middleware system that features specific enhancements designed to support the development and assessment of highly secure, dependable, service-oriented Grid systems and applications. The architecture of the new middleware is discussed, and the architecture and functionality of each dependability and security enhancement is described, alongside the results of experimental evaluations of each enhancement. Future work is then discussed.

1. Introduction

Service-orientation is emerging as a highly useful means of developing flexible, agile, and dependable software systems. A service can be defined as "*a mechanism to enable access to a set of one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description.*" [1], whilst [2] defines a service-oriented architecture as:

"*an application architecture within which all functions are defined as independent services with well-defined invokable interfaces, which can be called in defined sequences to form business processes.*"

Service-orientation aims to facilitate the development of complex, dynamic, inter-organisational systems as well as to greatly simplify the process of integrating existing legacy systems, and has a profound impact on the software development process. As services are loosely-coupled and can be invoked through well-supported standards, it becomes possible to construct *demand-centric* [3] applications by dynamically discovering and binding to autonomous system components at run-time. This process has been described as *ultra-late binding*, and can be used to create so-called *agile computing systems* with dynamic execution conditions and resource demands [4]; specific functionalities of applications can be altered in part simply through modifications to the composition of services that they use. Service-orientation can also drive down the costs of developing applications, as the loosely-coupled nature of services often means that services can be reused in different applications or easily replicated for gains in dependability, and it is speculated that the availability of multiple functional-equivalent services may reduce the expense (in terms of both development time and cost) and improve the feasibility of developing *fault-tolerant* systems [5].

The *Grid Computing* community is reorienting itself toward the service-oriented paradigm (and thus creating so-called *service-oriented Grids*) in order to help solve the fundamental problem of coordinated resource sharing and problem solving in dynamic, multi-organisational *virtual organizations (VOs)*. A VO is formed by whenever an application or workflow is created that features autonomous services owned by multiple organizations, every participating organisation makes available (i.e. shares) some proprietary services and part of its own knowledge.

1530-2059/07 $25.00 © 2007 IEEE
DOI 10.1109/HASE.2007.56

However, service-oriented Grids bring about many new challenges not faced by traditional distributed systems research, in areas such as security, dependability and dependability assessment; these are of especial interest when developing *high-assurance systems* (systems that require levels of safety and dependability that far exceed normal enterprise needs, and invariably experience high consequences in the event of a system failure). The new challenges concern not only the composition of service workflows, but also the network environments that the services are deployed upon, which are ever more spatially disparate, heterogeneous, and spread across many administrative domains.

This remainder of this paper introduces some of the new challenges discussed above, before introducing *CROWN-C*, a new Grid middleware system with specific functionality enhancements to support the development and assessment of highly secure and dependable service-oriented Grids. Each of these enhancements is discussed in detail, and the empirical results of evaluations of each enhancement are presented, before the paper concludes with a look at future work on CROWN-C.

2. New issues due to service-orientation

Many of the new challenges faced by the High Assurance Systems community due to the advent of service-orientation and Grids can be related to the concepts of *dependability* and *security*. Dependability is defined in [6] as "*that property of a computer system such that reliance can justifiably be placed on the service it delivers. The service delivered by a system is its behaviour as it is perceived by its users.*" It is important to state that this definition of dependability is not simply a synonym for reliability; rather, reliability is just one attribute of the overall concept.

Traditionally, dependability is a global concept, and subsumes the attributes of reliability, availability, safety, integrity, maintainability, and confidentiality. However, these attributes are becoming increasingly differentiated; for example, [7] distinguishes between dependability and security attributes - as shown in Figure 1 - in order to highlight the main balance of interest given to these attributes by their respective communities. Nonetheless, even if such a differentiation is made, it does not follow that the dependability or security communities have no interest or activity in the other listed attributes, and our research takes into consideration both of these concepts.

Many of the major challenges to the provision of high-assurance service-oriented Grids are due to issues arising from the autonomicity of individual services within virtual organisation workflows. Table 1 relates the problems posed by this autonomicity to the attributes of dependability and security.

New challenges emerge in the area of testing too; when viewed from the point of view of dependability, there is an implicit trust of middleware. Service-oriented middleware products are used as a basis for large undertakings, but there is little or no information on dependability assessment of the middleware itself, even though it greatly impact the security and dependability of a system constructed upon it. Furthermore, there is an implicit distrust of third party services when constructing service-oriented systems; organisations much prefer to create their own services, for which they have QoS statistics. Dependability assessment techniques must be performed to both ensure that robust middleware is developed, and also that trust issues with third party services are overcome.

In the area of security, the demand-centric nature of applications executing on service-oriented Grids leads to often unpredictable workflows and business processes, and on some occasions, the actual execution of a business process can be "one-of-a-kind". As each organisation within a VO has its own security mechanisms and policies to protect its local resources, a composite application has to operate amongst multiple heterogeneous *security realms*. A security realm is a group of principals (people, computers, services etc.) that are registered with a specified authentication authority and managed through a consistent set of security processes and policies.

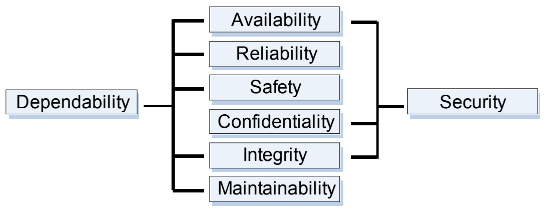


Figure 1. Dependability and Security attributes in [7].

Because organisations and services can join a collaborative process in a highly dynamic and flexible way, it cannot be expected that every pair of collaborating security realms always have a direct cross-realm authentication relationship. A possible solution to this problem is to locate some intermediate realms that serve as an authentication-path between the two separate realms that are to collaborate. However,

Table 1. Some dependability attributes and their link with autonomous services

Dependability Attribute	Autonomicity issue
Availability	A participant within a VO may have no control over the availability of services provided by other participants; service availability may be dynamic and unpredictable [8].
Reliability	A participant within a VO may have little or no knowledge of the reliability (including the performance) of services provided by other participants; this is especially a problem in long running interactions [5].
Confidentiality	A participant within a VO may have little or no knowledge of the access procedures in place to protect the confidentiality of data shared with services provided by another participant. [9].
Integrity	A participant within a VO may have little or no knowledge of the security procedures in place to protect the integrity of services provided by another participant. [10].

the overhead of generating an authentication-path for two distributed realms is non-trivial. The process can involve a large number of extra operations for credential conversion, and require a long chain of invocations to intermediate services. Moreover, such authentication paths may not exist between security realms in many cases.

In addition to this, traditional access control methods based on the identity of each user in a VO do not scale as the number of users and services increase, especially when the population of users and services is highly dynamic. It is a fundamental but challenging problem to dynamically build mutual trust between service requesters and providers coming from different security domains, whilst preserving their privacy in open Grid environments.

Although much work has been performed in both the Web Services and Grid Computing communities to address some of these issues on an individual level, no major service-oriented Grid middleware system has focused on integrating specific dependability and security technologies in order to provide a fully-integrated environment for the assessment and deployment of secure and dependable high-assurance applications and systems.

In light of this, the University of Leeds (UK) and Beihang University (China) are collaborating to develop a Grid middleware system that features integrated tools for the assessment and deployment of high-assurance systems. The collaborative project is known as COLAB (COllaboration between Leeds And Beihang) and involves extending the CROWN (China Research and development environment Over Wide-area Network) Grid middleware developed at Beihang University with service-oriented security and dependability technologies developed at the University of Leeds. The result of this collaboration is a high-assurance service-oriented Grid middleware system known as CROWN-C (CROWN-COLAB).

The remainder of this paper presents the architecture of CROWN-C, and discusses each integrated dependability and security enhancement. Empirical results of the assessments of each enhancement are presented, and future work is discussed.

3. CROWN-C

According to [11], the three major issues addressed by Grid computing are:

- Co-ordination of resources that are not subject to central control
- Standard, open, general-purpose protocols and interfaces
- Delivery of non-trivial Quality of Service (*QoS*)

In order to effectively utilize and deliver these features, some form of middleware is required. One of the most popular middlewares to attempt this is the *Globus Toolkit* (http://www.globus.org/toolkit), a reference implementation developed by the *Globus Alliance* (http://www.globus.org) - a "*community of organizations and individuals developing fundamental technologies behind the Grid*". Although originally oriented towards High Performance Computing clusters, the Globus Toolkit has now evolved to offer a service-oriented approach, based on the Open Grid Services Architecture (*OGSA*) and the Web Services Resource Framework (*WSRF*) standards, developed by the *Open Grid Forum* (http://www.ogf.org).

OGSA is a service-oriented architecture, which adopts the notion of a service as a unified resource encapsulation format to provide better extensibility and interoperability between Grid resources, whilst WSRF refines the service interface and interoperating protocols of OGSA; this makes OGSA a Web Service compatible implementation framework, which facilitates the merging of Grid and Web service technology. However, as the new challenges discussed in Section 2 show, there is still a great deal of work that needs to be done in order to support the very demanding levels of Quality of Service required by high-assurance systems. CROWN-C is a Grid middleware system that attempts to provide a middleware to enable such systems.

The CROWN-C Grid middleware is an extension of the CROWN (China Research environment Over Wide-area Network) Grid middleware, which itself is based upon the Globus Toolkit. The major differentiators between the Globus Toolkit and CROWN are that CROWN places more emphasis on Grid resource management and dynamic management mechanisms at the design stage, and provides a new security architecture with distributed access control mechanisms and trust management mechanisms.

Furthermore, CROWN features integrated *provenance* recording capabilities; the provenance of a piece of data is the *documentation of the process that led to a given data element's creation*. This concept is well established, although past research has referred to the same concept with a variety of different names (such as *lineage* and *dataset dependence*). Provenance in relationship to workflow enactment and service-oriented architectures (SOAs) is discussed in [12]; in a workflow-based SOA interaction, provenance provides a record of the invocations of all the services that are used in a given workflow, including the input and output data of the various invoked services, and their location. The recording of provenance is essential in many areas; for example, in the pharmaceutical industry, there is a legal requirement in many countries to record the provenance of *in-silico* experimentation. The particular provenance recording and querying technology integrated into CROWN is the *PreServ* service-oriented provenance scheme, developed at the PASOA project at the University of Southampton, UK (http://www.pasoa.org).

The features of CROWN are discussed in more detail in [13]. For CROWN-C, the following areas were targeted for improvement:

- Fault-tolerance support tools
- Fault-injection based assessment facilities
- Advanced multi-party authentication tools
- Automated trust negotiation tools

The specific technologies integrated into CROWN to address such issues are *FT-Grid* to support fault-tolerance, *CROWN-FIT* to support fault-injection based assessment, *Grid-MPA* to address multi-party authentication issues, and *ATNService* to provide automated trust negotiation. An overall architecture of CROWN-C is shown in Figure 2, with the enhancements on top of the CROWN Grid middleware shaded. Each of these tools are discussed in the following Sections.

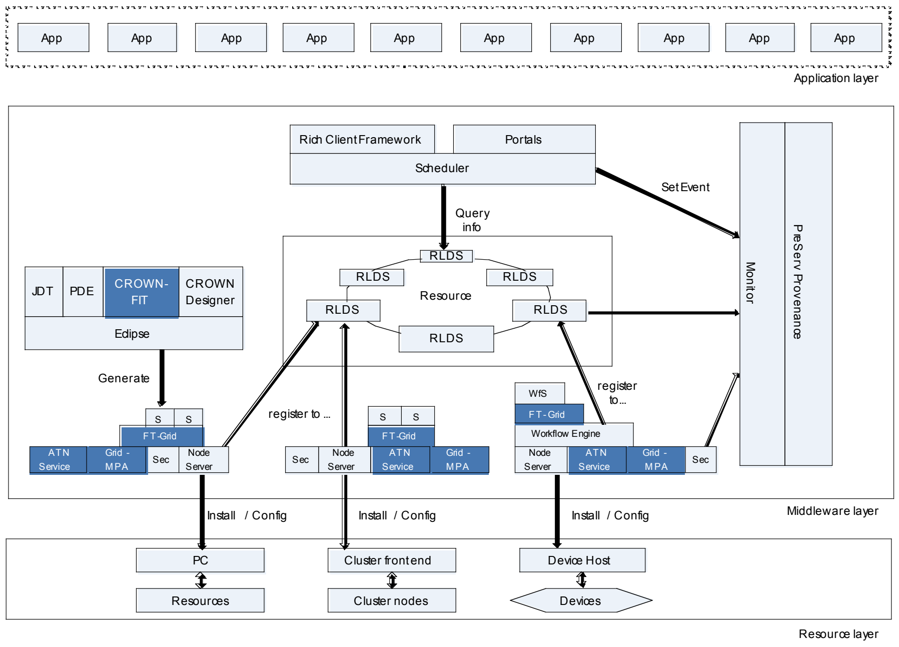


Figure 2. CROWN-C Architecture

4. FT-Grid

Given the issues raised in Section 2 (and specifically Table 1), it is important to investigate methods for overcoming the dependability challenges caused by service autonomicity, especially given the importance of guaranteeing Quality of Service levels in a demand-centric world. A traditional way to increase the dependability of distributed systems (both software and hardware) is through the use of fault-tolerant techniques. [14] describes the function of fault-tolerance as

"…*to preserve the delivery of expected services despite the presence of fault-caused errors within the system itself. Errors are detected and corrected, and permanent faults are located and removed while the system continues to deliver acceptable service.*"

Many fault-tolerant techniques use some form of diversity to achieve their aim; of particular interest in a service-oriented context is *design-diversity fault-tolerance.* Design-diversity is a general way of allowing a system to operate successfully in the presence of a design fault by constructing the entire system from a number of diverse designs (i.e. software redundancy) derived from a common specification. An example of such a system is shown in Figure 3. This particular system uses a so-called *n-version design* approach, whereby multiple (in this case *n=3*) functionally-equivalent *channels* are invoked in parallel (although sequential invocation is conceptually possible). An adjudication mechanism compares their outputs and can either forward a consensus result or else fails with no agreement. In addition to consensus voting, a variety of other adjudication techniques may be applicable (a survey of various voters for diverse redundant components is given in [15]).

In the context of demand-centric, service-oriented Grid applications, the technique of design-diversity shows much promise as it may be possible to create such systems at much lower cost than would traditionally be the case. This can be achieved by dynamically binding to pre-existing, functionally-equivalent services at run-time. However, in this context, a new problem that is not encountered in traditional distributed systems occurs. This is due to the possibility that multiple demand-centric services may, during the execution of their dynamic workflows, invoke identical (and potentially faulty) 'common' services and hardware. This has the effect of reducing channel diversity, and increasing the likelihood of *common-mode failure* occurring (whereby multiple channels give similar incorrect outputs), thus potentially causing the adjudication mechanism to forward an incorrect result.

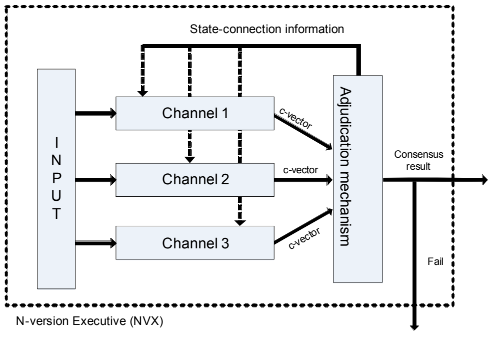


Figure 3. *N*-Version Design system.

This is problematic, as in many high-assurance and safety-critical systems, forwarding on an incorrect result is often far more dangerous than announcing a failure (and possibly moving the system into a safe state). The *common service problem* is shown in Figure 4. The *FT-Grid* tool [5], developed at the University of Leeds and integrated into CROWN-C, attempts to reduce the impact of this problem, through the use of the *PreServ* tool (already integrated into COLAB-C, as discussed in Section 3) to assist in the derivation of topological information about system workflows.

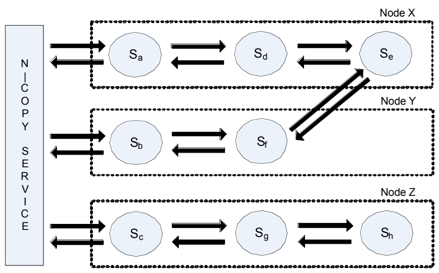


Figure 4. The common service problem

FT-Grid allows a user to manually search through any number of public or private UDDI repositories (should more than one UDDI server be specified, then FT-Grid will collate and return matching services from all UDDI servers specified), select a number of functionally-equivalent services, choose the parameters to supply to each service, and invoke those services in parallel. FT-Grid can then perform voting on the results returned by the services, with the aim of filtering out any anomalous results. Topological data is derived from analysis of the provenance records of service workflows generated by PreServ; by achieving topological awareness, FT-Grid may adapt to dynamic, demand-centric services in order to limit the common service problem, through techniques such as weighted voting [15].

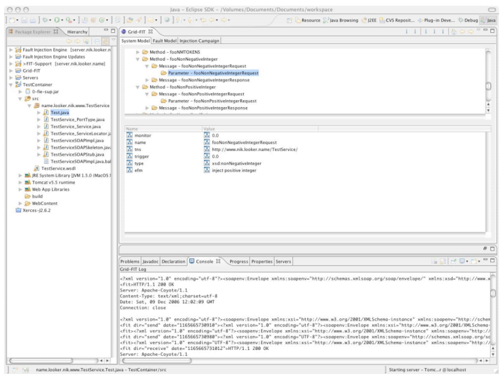
Table 2. Summary of FT-Grid results

	Correct result	No result	CMF
Simplex Run 1	828	172	-
Simplex Run 2	858	142	-
Simplex Run 3	822	178	-
Simplex average	**836**	**164**	**-**
Traditional NVD Run 1	928	9	63
Traditional NVD Run 2	921	14	65
Traditional NVD Run 3	921	7	72
Traditional NVD Average	**923.33**	**10**	**66.66**
FT-Grid Run 1	996	4	0
FT-Grid Run 2	990	10	0
FT-Grid Run 3	996	4	0
FT-Grid Average	**994**	**6**	**0**

Multiple runs of experiments performed through the use of FT-Grid running an *n*-version design fault-tolerance configuration on Grid services perturbed using CROWN-FIT (discussed in Section 5) have shown it to obtain a greater percentage of correct results than either a single (*simplex*) service, or a traditional *n*-version design fault-tolerance scheme. These results are shown in Table 2.

5. CROWN-FIT

Fault Injection Technology (FIT) is the University of Leeds network level fault injector framework designed to work with middleware systems in order to address the dependability assessment issues discussed in Section 2. FIT contains a *Fault Injection Engine* (FIE) that is implemented in such a way that different middleware message formats can be handled, including both text and binary. *CROWN-FIT* is a specific tailoring of the FIT framework to work with the CROWN-C and Globus Toolkit middlewares. CROWN-FIT is implemented as a plug-in for Eclipse, which is a platform independent framework for developing applications (see Figure 5), and works alongside a Grid service development plug-in known as *CROWN Designer*.


Java – Eclipse SDK – /Volumes/Documents/Documents/workspace
Resource
Java Browsing
J2EE
CVS Reposit...
Plug-in Deve...
Debug
Java
Package Explorer
Hierarchy
Fault Injection Engine [server.nik.looker.n
Fault Injection Engine Updates
>FIT-Support [server.nik.looker.name]
Grid-FIT
Servers
TestContainer
0-fie-sup.jar
src
name.looker.nik.www.TestService
Test.java
TestService_PortType.java
TestService_Service.java
TestService_ServiceLocator.j
TestServiceSOAPImpl.java
TestServiceSOAPSkeleton.java
TestServiceSOAPStub.java
TestServiceSOAPImpl.java.bal
TestService.wsdl
JRE System Library [JVM 1.5.0 (MacOS
Tomcat v5.5 runtime
Web App Libraries
build
WebContent
Xerces-J2.6.2
Grid-FIT
System Model
Fault Model
Injection Campaign
Method – fooNMTOKENS
Method – fooNonNegativeInteger
Message – fooNonNegativeIntegerRequest
Parameter – fooNonNegativeIntegerRequest
Message – fooNonNegativeIntegerResponse
Method – fooNonPositiveInteger
Message – fooNonPositiveIntegerRequest
Parameter – fooNonPositiveIntegerRequest
Message – fooNonPositiveIntegerResponse
Problems
Javadoc
Declaration
Console
Progress
Properties
Servers
Grid-FIT Log
name.looker.nik.www.TestService.Test.java – TestContainer/src
Starting server – Tomc...r @ localhost

Name	Value
monitor	0.0
name	fooNonNegativeIntegerRequest
tns	http://www.nik.looker.name/TestService/
trigger	0.0
type	xsd:nonNegativeInteger
efm	inject positive integer

```
<?xml version="1.0" encoding="utf-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w
<fit>HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=utf-8
Date: Sat, 09 Dec 2006 12:02:09 GMT
Connection: close

<?xml version="1.0" encoding="utf-8"?><soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:soapenv="http://schem
<fit dir="send" date="1165665730910"><?xml version="1.0" encoding="utf-8"?><soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
<fit><?xml version="1.0" encoding="utf-8"?><soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:soapenv="http://
<fit dir="send" date="1165665730980"><?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/sc
<fit><?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:soapenv="http://
<fit dir="receive" date="1165665731012">HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
```

Figure 5. The Eclipse-based *CROWN-FIT* tool.

The major innovation of CROWN-FIT is a novel fault injection mechanism that allows network-level fault injection to be used to simulate Code Insertion fault injection, whilst circumventing the need for modifications to service source code [16]. This is accomplished by intercepting middleware messages within the protocol stack, decoding the middleware message in real-time, and injecting appropriate faults. This is shown in Figure 6.

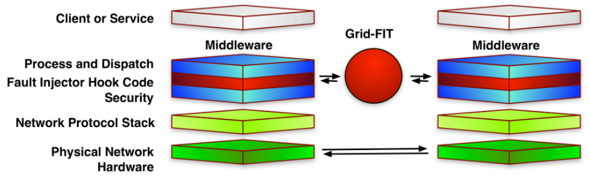


Figure 6. CROWN-FIT injection points

Since middleware messages (Grid middlewares typically use the *SOAP* [17] message protocol) are intercepted as complete entities, it is possible to corrupt, reorder and drop complete messages, rather than just part of a network packet that may be discarded before it reaches the middleware layer. Messages can thus be modified and then passed on to the rest of the protocol stack; in this way, faults can be injected but not filtered out by the protocol stack. By decoding the middleware message and allowing this level of targeted fault injection, it is possible to perform parameter perturbation similar to that achieved by Code Insertion at the API level. This can also be used to perturb SOAP element attributes in order to assess middleware protocols.

As an example of the effectiveness of CROWN-FIT within the CROWN-C Grid middleware, we present a case study that expands upon the results obtained in [18], which uses FIT to demonstrate potential flaws in an underlying Grid Middleware. This case study develops one aspect of that work; namely the lack of validation of SOAP messages within Globus-based Grid middlewares. This has potentially serious implications in terms of the integrity of a system, especially in terms of malicious attacks, since SOAP messages can be constructed and sent to a service which have the potential to cause attacks such as Denial of Service (DoS) and buffer overflow.

Figure 7 shows a typical SOAP message; each element is composed of a tag and attributes containing schema information. This case study uses CROWN-FIT to invalidate the schema information contained within a SOAP message whilst keeping its XML syntactically correct.

Table 3. Results of the CROWN-FIT case study.

SOAP element	Element attribute	Original Value	Injected Value	Exception Thrown
soapenv:Envelope	xmlns:soapenv	http://schemas.xmlsoap.org/soap/envelope/	"invalid schema"	no
	xmlns:xsd	http://www.w3.org/2001/XMLSchema	"invalid schema"	no
	xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance	"invalid schema"	no
	extra attribute		soapenv:Envelope	no
soapenv:Body	extra attribute		soapenv:Body	no
ns1:fooResponse	soapenv:encodingStyle	http://schemas.xmlsoap.org/soap/encoding/	"invalid schema"	no
	xmlns:ns1	http://www.nik.looker.name/TestService/	"invalid schema"	no
	extra attribute		ns1:fooResponse	no
fooResponse	extra attribute		fooResponse	no

CROWN-FIT fault models are written which substitutes new text for a specific schema address. These fault models are then applied to various SOAP messages in sequence, in order to see if any adverse affects can be observed. A further fault model is constructed that adds an extra attribute to an element; this is in violation of the SOAP schemas. Table 3 gives the results obtained from the fault injection campaign. From this we can see that no exceptions are thrown for any of the schema invalidations, thus demonstrating that no XML validation is preformed on SOAP messages by the Axis 1.3 java package used in this case study.

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
   <ns1:fooResponse
     soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     xmlns:ns1="http://www.nik.looker.name/TestService/">
     <fooResponse xsi:type="xsd:unsignedInt">0</fooResponse>
   </ns1:fooResponse>
 </soapenv:Body>
</soapenv:Envelope>
```

Figure 7. Example SOAP message.

Since XML parsing and validation are relatively time costly activities, we can postulate that a decision was made during development to minimise some of this overhead by turning XML validation off. In order for the approach to work, the assumption must be made that a valid, non-malicious middleware is being used to communicate with Axis 1.3; for this to be a valid assumption, there must be an implicit trust of all the components within a system. Whilst this assumption may hold for well known and trusted Grid environments, it is more of a risk when composing systems from third party services which may have been located by service discovery, since there may be no trust guarantee. Whilst some of the risk may be mitigated via the use of a broker, a middleware that doesn't have the potential vulnerabilities as those described above is desirable.

6. Grid-MPA

As mentioned in section 2, dynamic authentication between organisations can be highly complex and time-consuming if some intermediate authentication paths have to be created and credentials have to be converted. Therefore, when there is no existing direct authentication relationship in place between two security realms, it is practically difficult for a system to enable any secure collaboration between services from the two security realms in a just-in-time fashion. In order to address *heterogeneous cross-realm authentication* (HCRA) issues in service-based business sessions, we have developed a new authentication system entitled *Grid-MPA*. Our Grid-MPA method refers to a business process execution as a *business session*, and the principals working within the business session as *session partners*. In a Grid context, session partners are Grid service instances.

In Grid-MPA, every session partner is associated with a distinct identifier, and a secret key is generated and distributed for every pair of collaborating session partners. Session authorities (SAs) are employed to manage the membership of session partners in business sessions and provide reliable real-time information for authentication between session partners.

Before a principal (i.e. service instance) joins a business session, it must attempt to register with the associated SA. The SA will decide whether to accept it based on certain policies; for example, the SA may require that the applicant must be recommended by a member (or multiple members) of the corresponding business session. In this way, Grid-MPA generates a security boundary for business sessions that allows only trusted principals to join a business session; it also allows the identity of session partners to be verified through cryptographic methods. Therefore, collaborating instances within a business session can authenticate with each other by simply using their session memberships, as illustrated in Figure 8; thus, a reasonable level of trust relationships between members of a business session can be generated.

Because in Grid-MPA authentication of a Grid service instance within a business session is much simpler than in conventional HCRA, we refer to this as *simplified cross realm authentication* (SCRA). In a multi-party session with n security realms, up to $(n - 1) \times (n - 2)/2$ authentication processes can be simplified as SCRA, based on session memberships. Therefore, Grid-MPA largely avoids the establishment of authentication paths between collaborative session partners and credential conversion.

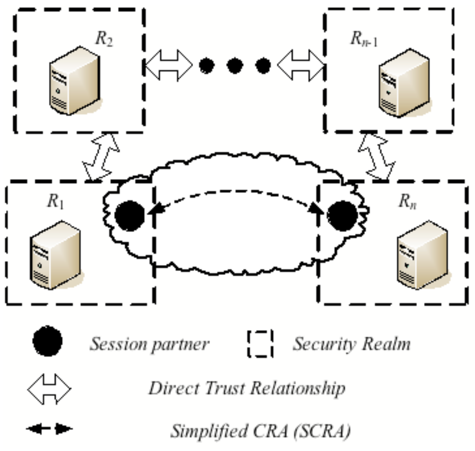


Figure 8. HCRA and SCRA

Grid-MPA has been tested with two Grid middleware systems - CROWN-C and Globus Toolkit 4 - and an experimental system has been created to evaluate the performance of Grid-MPA in a realistic environment. Empirical evaluation results show that Grid-MPA (*ES1* in Figure 9) has very promising scalability trends. In the experiment illustrated in Figure 9, the time consumption of Grid-MPA when introducing new session partners is proportional to the number of session partners introduced, and Grid-MPA can execute in a stable state until more than 260,000 session partners are generated and introduced into a business session (after this amount, all available memory had been consumed in the test system). Additionally, these results indicate that the overhead imposed by Grid-MPA is comparable with the overheads introduced by the standard security mechanisms (*ES2* in Figure 9) used in both the standard CROWN and GT4 middleware systems.

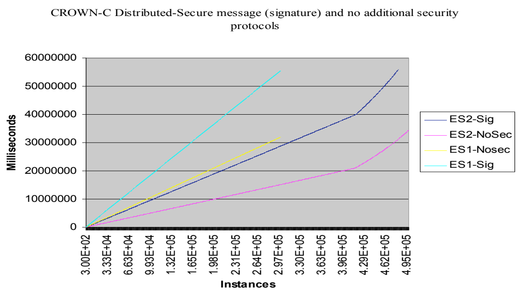


Figure 9. Empirical evaluation

Besides empirical evaluation, the correctness of the Grid-MPA protocols has been formally analysed with BAN logic; the details of this analysis are presented in [19].

7. ATNService

As discussed in Section 2, dynamically building mutual trust relationships between service requesters and service providers from different security realms is a fundamental problem, especially when considering the need to preserve their privacy.

To address these issues, we have developed an *ATNService* [20] that supports the *automated trust negotiation* [21] approach to enhance the Grid security infrastructure, thereby allowing flexible negotiation between strangers from different security realms. This service has been successfully implemented into CROWN-C, and complements existing Grid security infrastructures when trust is needed between unknown parties. During the process of trust negotiation, the ATNService invokes an *ATN Engine*; this is an independent library that parses credentials and policies, and manages the states of different trust negotiation sessions. It features three key components:

- **Negotiation Strategy**: This component decides whether the negotiation process should continue or not. If it continues, it determines whether credentials can satisfy the specified access control policy, then discloses the corresponding satisfied credentials, or new access control policies.
- **Trust Chain Construction**: This component is used to collect necessary credentials and construct a *trust chain* (also known as a *credential chain*) from trusted issuers to the requester, based on delegation credentials, role mapping policies, etc.
- **Trust Ticket Manager**: This component can issue or verify short-term tickets to avoid re-negotiation when multiple service access requests occur in a short time interval.

The relationship between the ATN Engine and ATNService is illustrated in Figure 10. When a client wishes to access a target service protected by ATNService, a series of procedures are invoked, including building a secure tunnel based on the Secure Conversation specification, trust negotiation, and target service authorization. When the client sends a service request message to a target service located in another domain, the requesting SOAP message processing chain contains a local RedirectHandler that will initialize an ATN Engine. This ATN Engine will then invoke the ATNSevice for the target service.

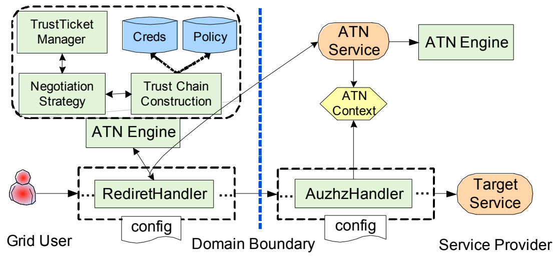


Figure 10. CROWN-C ATNService and ATN Engine

Upon receiving negotiation requests from the client's ATN Engine, the ATNService of the service provider will also initialize an ATN Engine, and store the state of negotiation into its *ATNContext*. The two participants may disclose their credentials and access control policies for sensitive information for several rounds, until a final decision ('success' or 'failure') is reached. If the negotiation succeeds, the ATNService will return a success message, and the context stored in ATNContext will be updated accordingly.

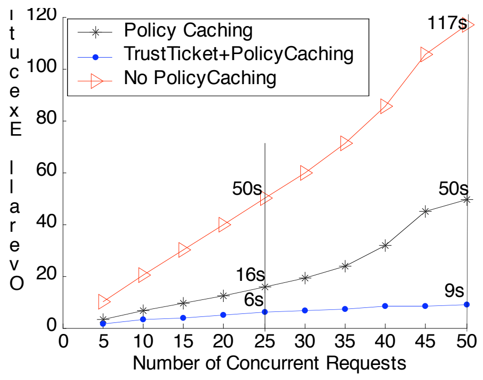


Figure 11. Number of concurrent requests vs. execution time

Finally, the client can insert the session id into a SOAP message header and sign it before sending the message to the target service; the target service can verify the authenticity of the session id through its AuthzHandler, and authorize service access if verification succeeds. Additionally, in a service container, a special target service may be accessed frequently by many requesters; therefore, the caching of credentials and access control policies, with a dedicated queue for such services, is supported. This mechanism is intended to avoid frequent initialization of the negotiation engine.

In CROWN-C, we have evaluated ATNService through comprehensive experimentation, with encouraging results. Figure 11 plots overall execution time against the number of concurrent requests. As can be seen, the overall negotiation execution time of ATNService increases linearly, and the effectiveness of trust ticket and policy caching is obvious. For instance, when 25 concurrent requests are received, service execution time is 6s, 16s and 50s respectively. When 50 concurrent requests are received, service execution time is 9s, 50s and 117s respectively. These results highlight the strong reduction in overall execution time when using trust ticket and with-caching methods. In CROWN-C, we believe that most service invocations benefit from the trust ticket and policy caching mechanisms.

8. Conclusions and future work

CROWN-C is a Grid middleware system developed by the Universities of Leeds, UK, and Beihang, China, as part of the EPSRC-funded COLAB project. It extends the CROWN Grid middleware to feature specific enhancements designed to support the development and assessment of high-assurance service-oriented Grid systems and applications.

This paper highlights some of the new dependability and security challenges introduced by the service-oriented paradigm, and relates these challenges to the dependability and security enhancements featured in the CROWN-C Grid middleware.

These enhancements are: 1) The *FT-Grid* fault-tolerance tool, designed to facilitate the development of topology-aware, dynamic fault-tolerant service-based systems; 2) The *CROWN-FIT* fault-injection tool to perform middleware-level dependability assessment; 3) the *Grid-MPA* tool, which seeks to enable instance-level multi-party authentication between interacting services; and 4) the *ATNService* tool, which seeks to establish a capability for automated trust negotiation between virtual organisation participants from different security realms. For each enhancement, the latest experimental results and evaluations are presented.

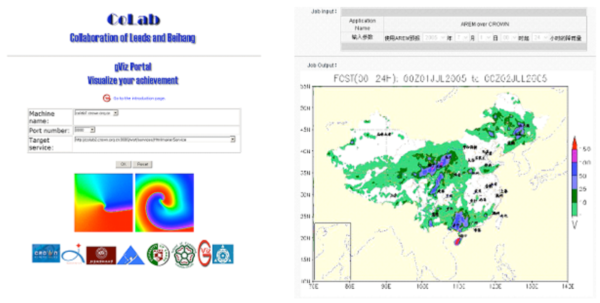


Figure 12. Screenshots of CROWN-C running the g-Viz and AREM services

Work on the CROWN-C system is ongoing as part of the COLAB project; future work includes more rigorous testing of the CROWN-C system, using a wide variety of Grid applications, such as the *g-Viz* visualization service [22] and the Advanced Regional Eta-coordinate Model (*AREM*) weather prediction tool [23]. Screenshots of both of these applications running on CROWN-C are shown in Figure 12.

Acknowledgements

CROWN-C is developed in part through the COLAB project (EPSRC Grant EP/D077249/1). PreServ was developed at the University of Southampton.

References

[1] OASIS Reference Model for Service Oriented Architectures, Working Draft 11, http://www.oasis-open.org/committees/download.php/15966/wd-soa-rm-11.pdf, 15th Dec 2005.
[2] K. Channabasavaiah, K. Holley, E. Tuggle, “Migrating to a Service-Oriented Architecture”, IBM White Paper, http://www-128.ibm.com/developerworks/webservices /library/ws-migratesoa/, 2004.
[3] K. Bennett, P. Layzell, D. Budgen, P. Brereton, L. Macaulay, M. Munro, "Service-based software: the future for flexible software," in Proc of the 7th APSEC, pp. 214, 2000.
[4] P. Townend, J. Xu, “Dependability in Grids”, in IEEE Distributed Systems Online, Vol. 6, No. 12, Dec 2005.
[5] P. Townend, P. Groth, J. Xu, "A Provenance-Aware Weighted Fault Tolerance Scheme for Service-Based Applications", in Proc of the 8th IEEE ISORC, Seattle, 2005.
[6] B. Randell et al., “Dependability – Its Attributes – Impairments and Means”, in Predictably Dependable Computing Systems, Springer-Verlag, 1995.
[7] A. Avizienis, J-C. Laprie, B. Randell, C. Landwehr, “Basic Concepts and Taxonomy of Dependable and Secure Computing,” in IEEE Transactions on Dependable and Secure Computing, Vol. 1, No. 1, January-March 2004.
[8] I. Jang-uk, P. Avery, R. Cavanaugh, L. Chitnis, M. Kulkarni, S. Ranka, “SPHINX: A Fault-Tolerant System for Scheduling in Dynamic Grid Environments”, in Proc of the 19th IEEE IPDPS, 2005.
[9] M. Elliot, S. Pickles, K. Purdam, D. Smith, “Disclosure Risk and Grid Computing”, in Proceedings of 4th U.K. e-Science All-Hands Meeting, 2005.
[10] S. Naqvi, M. Riguidel, “A Threat Model for Grid Security Services”, in Proc of the EGC 2005, Amsterdam, The Netherlands, Lecture Notes in Computer Science 3470, 2005.
[11] Foster I, “What is the Grid? A three point checklist,” http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf
[12] M. Szomszor, L. Moreau, “Recording and reasoning over data provenance in web and Grid services”, in Proc of the ODBASE 2003, Vol. 2888 of Lecture Notes in Computer Science, pp. 603-620, Catania, 2003.
[13] J. Huai, C. Hu, J. Li, H. Sun, T. Wo, “CROWN: A service Grid middleware with trust management mechanism”, in Science in China Series F: Information Sciences, Springer-Verlag GmbH, Vol. 49, No. 6, 2006.
[14] A. Avizienis, “The *N*-version Approach to Fault-Tolerant Software”, in IEEE Transactions on Software Engineering, vol. 11, 1985.
[15] F. Di Giandomenico, L. Strigini, “Adjudicators for Diverse Redundant Components”, in Proc of the 9th IEEE SRDS, pp. 114-123, Alabama, 1990.
[16] N. Looker, M. Munro, and J. Xu, "A Comparison of Network Level Fault Injection with Code Insertion," in the Proc of 29th IEEE COMPSAC, Edinburgh, 2005.
[17] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer, "Simple Object Access Protocol (SOAP)," ed: 1.3, http://www.w3.org/TR/soap/
[18] N. Looker and J. Xu, "Dependability Assessment of Grid Middleware," in Proc of the 37th IEEE/IFIP DSN, Edinburgh, vol. 1, pp 125-130, 2007.
[19] D. Zhang, J. Xu, and X. Li, “Dynamic Cross-Realm Authentication for Multi-Party Service Interactions,” in Proc of the 37th IEEE/IFIP DSN, Edinburgh, vol. 1, pp. 440-449, 2007.
[20] Jianxin Li, Jinpeng Huai, Jie Xu, Yanmin Zhu, Wei Xue, “TOWER: Practical Trust Negotiation Framework for Grids”, in Proceedings of the IEEE International Conference on e-Science and Grid Computing, Amsterdam, 2006.
[21] W. H. Winsborough, K. E. Seamons, and V. E. Jones, "Automated Trust Negotiation," in Proc of DARPA DISCEX, 2000.
[22] H. Wang, K. Brodlie, J. Handley, J. Wood, “A Service-oriented approach to collaborative visualization”, in Proceedings of the UK e-Science All Hands Meeting, pp. 241-248, 2006.
[23] Y. Li, W. Huang, J. Zhao, “Roles of mesoscale terrain and latent heat release in typhoon precipitation: A numerical case study”, in Advances in Atmospheric Sciences, Vol. 24, No. 1, pp.35-43, 2007.