Middleware for semantic-based security and safety management of open services

Stefan Poslad*

Department of Electronic Engineering
Queen Mary, University of London
Mile End Road, London E1 4NS, UK
E-mail: stefan.poslad@elec.qmul.ac.uk
*Corresponding author

Juan Jim Tan

^{e}Biz. Intelligence Group
Suite 1703, City Plaza, No. 21 Jalan Tebrau
80300 Johor Bahru, Malaysia
E-mail: jim@ebigsys.com

Xuan Huang and Landong Zuo

Department of Electronic Engineering
Queen Mary, University of London
Mile End Road, London E1 4NS, UK
E-mail: xuan.huang@elec.qmul.ac.uk
E-mail: landong.zuo@elec.qmul.ac.uk

Abstract: The trend towards ubiquitous public services is driving the deployment of large-scale, heterogeneous, distributed information services. In order to support automated information access and processing, this information is marked up using semantic metadata models represented using ontology languages such as OWL. The use of such a semantic metadata model is twofold: to enable content-based access and to provide services that enable users and applications to select, employ, compose and monitor web-based resources automatically. Deployment of a semantic model consists of the semantic metadata model itself and the semantic services to leverage the semantic metadata within application services. The main objective of this paper is to show how an existing semantic-based security management of open services framework has been extended to improve the safety of services using a semantic error management model. It has been applied to an application that queries multiple distributed environmental database resources.

Keywords: safety management; open services; semantic services; security; Web Ontology Language (OWL); management.

Reference to this paper should be made as follows: Poslad, S., Tan, J.J., Huang, X. and Zuo, L. (2005) 'Middleware for semantic-based security and safety management of open services', *Int. J. Web and Grid Services*, Vol. 1, Nos. 3/4, pp.305–327.

Biographical notes: Dr. Stefan Poslad is Lecturer at Queen Mary, University of London. He chaired the FIPA agent standards forum security technical committee and was on its Board of Directors before its recent transformation to become an IEEE standards committee in May 2005. He is an active member and coorganiser of agent, trust and ubiquitous system thematic networks and workshops. He has worked on several distributed system project applications for security including mobile user tourism services, environmental data integration and leisure services composition. His interests include: security, trust and privacy models for open distributed services; open distributed services models based upon the semantic web and intelligent agents, and ubiquitous computing.

Juan Jim Tan received both his MSc degree with distinction in e-Commerce Engineering and his PhD degree from the Department of Electronic Engineering in Queen Mary, University of London (QMUL). Part of his PhD has contributed to the EU Agentcities.RTD project where he has worked in the area of security and interoperability in open service environments. Currently, he is Academic Visitor at QMUL and Consultant at ^{e}Biz.Intelligence Group (Malaysia). His research interests include: the semantic web, adaptive management, and security interoperability for distributed systems.

Xuan Huang is currently a research student undertaking a PhD degree at the Department of Electronic Engineering in Queen Mary, University of London and part of her PhD has contributed to the EU EDEN-IW (IST-2000-29317) project where she has worked in the area of semantic safety management. Her research interests include: the semantic web, intelligent agents and multi-agent systems, and service and application integration. She received her MSc degree with distinction in Internet Computing from Queen Mary, University of London.

Landong Zuo is a PhD student at Queen Mary, University of London. He is working on the ontology engineering area in the intelligent communication lab of the Electronic Engineering Department. His work has contributed to the EU project EDEN-IW (IST-2000-29317) where he developed a resource agent service to integrate the heterogeneous legacy databases. His research interests mainly focus on semantic web, database integration, semantic management and interoperability for distributed information system. He had worked for Motorola China Ltd. as Senior Software Engineer on GSM and 3G CDMA network management system before he joined Queen Mary, University of London.

1 Introduction: background

An open service environment such as the internet provides users with increasing flexibility to select services. It lowers the barrier of entry for new providers to offer services when publication specifications are available, supporting a more dynamic and adaptive business environment for different stakeholders such as providers, users and service mediators. Given that such an environment is naturally open and dynamic, providers and users need to discover and understand each others' capabilities and preferences and users need to be able to invoke provider services on-the-fly. The operational management of open services is aided by using an explicit model of service actions and service processes. This supports the orchestration of multiple services and enables user service preferences to be matched against the advertised services.

1.1 Safety and security of open services

There are two major management concerns of open services: safety and security. Safety refers to the probability that a system does not fail in a manner that causes catastrophic damage (Nicol *et al.*, 2004). Security commonly refers to the ability to protect services against disruption and against unauthorised access. In addition to interoperability being needed at the service layer, interoperability is also needed at a management level. This is because applications make requests to services embedded in heterogeneous computational infrastructures accessible across heterogeneous networks. Application requests can fail, generating errors because of faults in the applications that request services; the services themselves and the processing and network infrastructure in which applications and services are executed. Application requests can also fail because of a lack of understanding between the application and the service infrastructure layers. Conventionally, failures are often handled either by masking them, by switching to a replicated copy if fault-tolerance is supported or by simply reporting infrastructure errors to the process that triggered the error. However, application-level processes often cannot understand such errors or why a service stops behaving as expected. Security and safety concerns are heightened because access is often across a public network and because open services can be more difficult to operate and to control access to.

Interoperability amongst the various open system actors can be greatly aided by sharing semantic type metadata as expounded by the semantic web in order to support a common understanding between them. Such semantic metadata is often represented using an ontology model that models domain knowledge in terms of taxonomic relations between object classes and instances, descriptions of attributes of elements of classes and constraints on class attributes.

However, the current generation of semantic web applications are still in their infancy and are limited by several engineering problems, such as the complexity in resolving numerous computation and linguistic problems arising from different contextual meanings and the use of fragile semantic message processing; this latter processing may cause fatal exceptions when incorrect or nonresolvable semantic terms are set. In addition, there is the inability to deal with multiple fragmented semantic views and the impedance mismatch of semantic actions triggering and possibly returning nonsemantic system infrastructure events that cannot be resolved, *e.g.*, semantic data exchange can cause network errors when a firewall prevents service access.

1.2 Aims

Although a semantic approach can be used to support service interoperability across different heterogeneous data instances and applications within a domain, the use of a semantic model in a distributed service environment needs some associated semantic services. These (semantic) application services need to be interlinked to management services to protect the security and safety of the application services. The main objective and contribution of this paper is to show how an existing semantic approach to support the security management of open services has been extended to improve the safety of services through the use of a semantic error management model. This is also an example of a general approach to interlink multidomain semantic metadata models and processes. In order to ground this approach, it is applied to an environmental information management problem that concerns integrating information from multiple heterogeneous databases containing inland water quality data; this was the subject of the Environmental Data Exchange Network for Inland Water or EDEN-IW project (Stjernholm *et al.*, 2004). As it is usually not an option to restructure the databases themselves so that all their database schemas are homogenised, middleware is layered over existing heterogeneous database resources for the following reasons: to support transparent queries across heterogeneous database resources; to support heterogeneous processing and to support heterogeneous users. This middleware needs to support safety and security management services to reduce service disruptions and failures.

1.3 Structure of this paper

The remainder of this paper is structured as follows: Section 2 surveys the main approaches to security and safety management. Section 3 describes the analysis and design of our method, which consists of a semantic service interoperability framework and an integrated system management framework to enhance security and safety management. In Section 4, the semantic management framework for safety and security is applied to the problem of environmental information integration. Section 5 discusses the application and presents conclusions.

2 Related work

2.1 Managing the security of open services

Security for open systems faces many new challenges when operating either in closed homogeneous and heterogeneous domains. Whilst static security configurations can be specified and agreed in advance (for example, web clients that need to use HTTPS to support confidential information exchange), this approach lacks flexibility in a dynamic heterogeneous domain that can have multiple security requirements. Syntactical specifications of security from the W3C, OASIS, WS-Security and Grid consortia support the representation of security information using XML but these have no semantic metadata models of security. The current plethora of different active security standards indicates that no single security framework is suitable for use with heterogeneous distributed systems. This has led to the use of mediating models that are able to bridge between disparate security standards (Denker *et al.*, 2003; Tan *et al.*, 2005).

Management domains provide the means for partitioning management responsibility by *grouping* objects in order to specify policies including access control policies. *Access control policies* specify the operations that a group of *subject* objects is permitted to perform on a group of *target* objects. Research has focused on policy specification languages such as Ponder (Sloman, 1994), TPL (Herzberg *et al.*, 2000), and PDL (Lobo *et al.*, 1999) that define rules for policy management. Policy-based management approaches such as KAoS (Bradshaw *et al.*, 1997) also include algorithms to handle conflict resolution. As the scope of these policies is specified in terms of management domains, access control decisions are mainly based on the authenticated domain membership of the subject and can be difficult to apply towards open service environments where membership is often more unpredictable.

In Table 1, a comparison is made between different approaches such as the WS-security specifications,[1] Grid Security (Foster *et al.*, 1998), and semantic and policy-based distributed systems such as KAoS (Bradshaw *et al.*, 1997), Rei (Kagal *et al.*, 2003), and Ponder (Sloman, 1994). It is evident that approaches differ in how they describe and offer security services and in their use of an explicit process model for interlinking security with application processes. Coupling semantic service profiles with process models has the potential to allow services to be dynamically orchestrated and to handle preferences or requests that may otherwise fail.

Table 1 Comparison of syntactical, policy-based and semantic approaches for security specification and management

	KaoS	*Rei*	*Ponder*	*WS-security*	*Grid security*
Semantic model	Yes	Yes	No	No	No
Policy model	Access control based Representation: Web Ontology Language (OWL)	Access control based Representation: Prolog-like syntax + RDF-S	Access control based Representation: own language	Constraint-based Representation: XML	Authorisations Representation: Grid policy/ WS-security
Process model	Not defined	Not defined	Not defined	BPEL4WS	Not defined
Explicit configuration model for interoperability	Uses mediating and proxy agents to link domains	Model for security service descriptions not clear	Model for security service descriptions not clear	Use of a profile model and standard service description language (WSDL)	Uses WS-security approach?
Reasoning model	Java theorem prover	Prolog engine	Event calculus representation	Not defined	Not defined

Security is more than a set of protection mechanisms and a set of rules and processes for activating them; it is also a set of processes that need to be interlinked and synchronised to the service processes that trigger them. For example, additional safeguards such as encryption are used to protect information confidentiality during particular parts of information exchange and processing. The use of encryption will lengthen processing and this may require message time-outs used to detect the nondelivery of messages to be adjusted. Therefore, there is a need to specify and manage service processes to support security management. Few of the methods surveyed have an explicit process model and configuration model to support interoperability between heterogeneous security management and application services processes (see Table 1). In addition, a reasoning model can also be useful in order to select the precedence of security policies when several apply and to select how best to handle a threat when a one-to-many mapping between a threat and safeguards exists.

2.2 *Managing the safety of open services*

A system failure can be defined as a deviation from the normal process. In open distributed systems, there is a greater chance of and a greater variety of failures because of the larger number and possible dynamic compositions of service components.

In addition, a more complex operating environment is present leading to transient and intermittent failures and because status messages and error messages may trigger failures that are not understood by the application processes. A further complication in an open distributed system is that Byzantine-type faults may be more likely. Byzantine faults occur when processes continue to operate issuing incorrect results or possibly work together with other faulty processes to give the impressions that they are working normally.

From a theoretical point of view, several different concepts of fault, failure and error could be differentiated: a fault is considered as the root cause of a failure, the failure represents some deviation from the normal behaviour and an error represents an invalid system state, *i.e.*, one that is not allowed by the system behaviour specification. This allows the system for example, to model an inactive fault that causes an error but that does not yet result in a system failure (Denker *et al.*, 2003). However, the semantics of these concepts vary between researchers.

The key challenges in managing such failures or errors are firstly, to detect the error or failure, then to identify the likely cause of the failure chosen out of several possible causes and finally to trigger a suitable error-handling process. System analysis, testing, quality assurance and operational monitoring can provide the data to design systems to specify the operational conditions in which faults are activated.

There are several approaches to failure management. In flow-based failure control, the flow of operations can be tested at set points to check the status of the operation, *e.g.*, did a particular operation return expected results, return no results or are they still pending? Based upon the process status when the error occurred, we can define and invoke a particular sub-branch of the process to allow the application process to continue. There is often an implicit application context associated with a fault, *i.e.*, the point in the process where the failure occurred. Hence the failure handling depends on the application processing conditions at the point of failure and the type of failure. Event-based failure management is based on the Event-Condition-Action or ECA pattern. The ECA model itself does not explicitly model the process conditions in which the failure occurs; this can make it difficult to select suitable failure handing when several possible ones may apply.

Another form of safety management is fault tolerance or the system's ability to supply regular service operations in the presence of hardware and software faults. In principle, the use of replication together with some management support services allows a failed resource to be replaced by its replicated copy. Avizienis *et al.* (2004) provide a basic taxonomy and concepts for dependable and secure computing elements. Lamport *et al.* (1982) use cryptographic methods to maintain a consistent state in distributed systems. DELTA-4 (Desmedt, 1994) uses replicated and security services to provide distributed intrusion-tolerant services for data storage, authentication and authorisation. The e-Vault prototype for secure distributed storage (Garay *et al.*, 1997) addresses a subset of the applications, namely storing and retrieving immutable data. Castro and Liskov (1999) present a practical algorithm for distributed service replication that is very fast if no failures occur. But the protocol is deterministic and can be blocked by Byzantine faults. The fleet architecture of Malkhi and Reiter (2000) supports a different coordination in large-scale distributed systems in which the domain does not attempt to remove malicious services. Instead it helps users to utilise (Byzantine) agreements between normally functioning processes in the face of Byzantine faults to maintain security in distributed environments.

In practice, web service-type specifications such as WSDL (Web Service Description Language)[2] and BPEL4WS[3] do not explicitly distinguish between fault, failure and error. Instead, a single concept of a failure or exception is modelled that specifies fault and remedial actions as its properties. These are expressed in XML. Many fault management methods do not explicitly model faults in a transparent manner that is extensible or easy to interlink into application processes. For example, WSDL models a fault as an event that disrupts the normal flow of messages during message exchange. An Interface Fault component describes a fault that may occur during invocation of an operation of the interface. When and how the fault message flows is indicated by the Interface Operation component. The sequencing and cardinality of the messages involved in a particular interaction is governed by the message exchange pattern used by the operation. A message exchange pattern defines placeholders for messages, the participants in the pattern and the cardinality and sequencing of messages exchanged by the participants. However, the WSDL core specification does not define machine-understandable message exchange patterns or processes, nor does it define any specific patterns.

BPEL4WS follows an ECA-type approach when dealing with errors. There are two fault properties. The fault name refers to the type of fault and the fault variable is the specific fault instance. Activities then are defined to handle specific faults. No machine-readable semantics are defined in the WSDL or BPEL4WS service models. OWL-S is a semantic service model but it does not yet define any exception handling, although the OWL-S model can be grounded using WSDL.

Hence, safety management will likely entail a set of multiple heterogeneous integrated fault management approaches. Safety management at an enterprise level can be seen as part of a more generic model of distributed system and network management that not only encompasses security management, safety management but also other management functions, *e.g.*, the ISO FCAPS (Fault management, Configuration management, Accounting, Performance management and Security management) model.[3] However, this conceptual model of management was developed at the time when computing was centralised and is difficult to apply in a heterogeneous distributed environment.

3 Method

The use of a semantic model enables content-based access to information and enables users and applications to select, employ, compose and monitor web-based resources automatically. The semantic model consists of two parts: the semantic metadata model and the semantic services to leverage the semantic metadata within the application services.

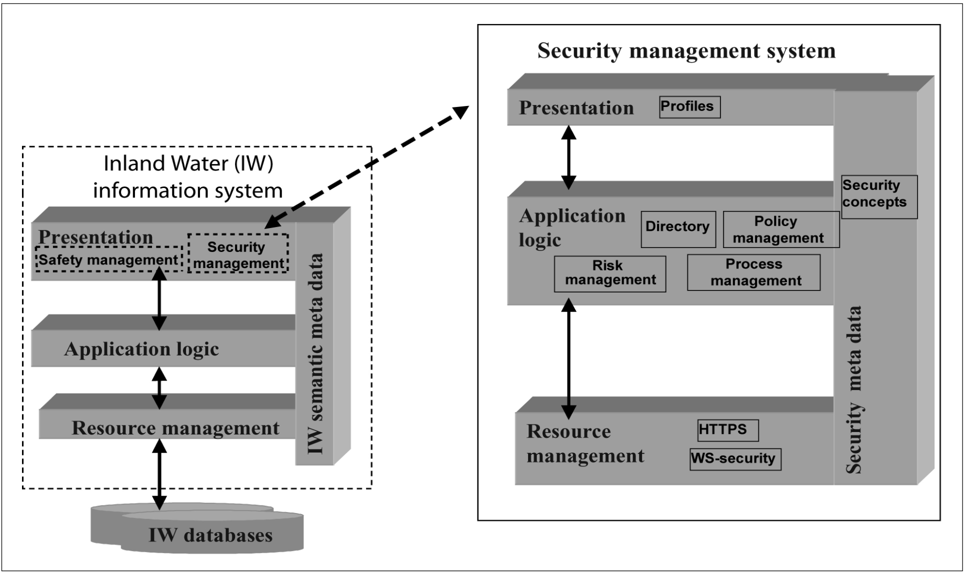
In this section, the design of the semantic management models for security and safety is described. The application and evaluation of these management services to the inland water domain are left to Section 4. In order to reuse semantic data models across multiple services in multiple application domains, different semantic concerns are partitioned to aid service integration. To support this, the design of the semantic metadata model is organised in a similar way to abstract information system models (see Section 3.1).

The Semantic service model to support security and safety management is designed in a similar way to the OWL-S Semantic Markup for Web Services model.[4] This defines a service model that tells a client how to use the service by detailing the semantic content of requests, the conditions under which particular outcomes will occur and describes the associated service processes. It defines service profiles to specify service configurations that can be offered, published and requested. It supports service groundings that specify the details of how an agent can access a service. There are several sets of groundings described for the semantic model here. The application groundings for the inland water domain specify how to access heterogeneous inland water resources. The security management service specifies groundings to use published security specifications to protect assets. Lastly, our system also uses a general grounding in order to use FIPA agent service descriptions to access and integrate services (Poslad and Charlton, 2001).

3.1 Semantic model for security management

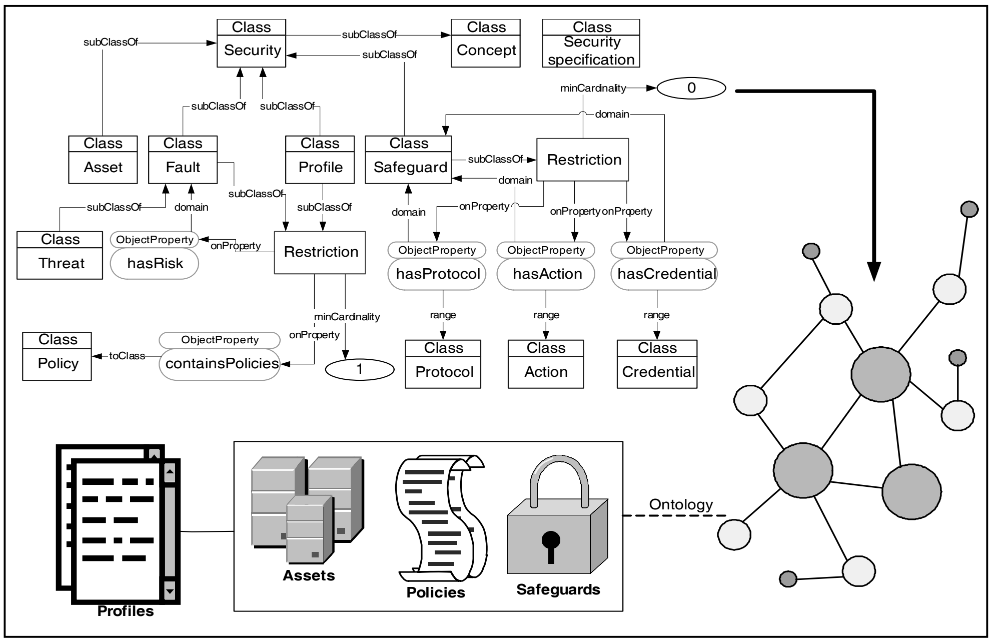
Security management can be as viewed as an application service for the Inland Water application (Section 4). The security management ontological model is however itself a semantic model that can be shared amongst heterogeneous application domains. It is partitioned in a similar way to the inland water application model (Section 4.1), in that the (V-SAT) conceptual model is separated from the different security management services that use those concepts such as the service advertiser and policy-based management applications (see Figure 1).

Figure 1 A three-tier information model management can be used to integrate services across multiple management services and application domains



The V-SAT model defines viewpoints of safeguards that protect assets (the items of value in a system, *e.g.*, the Inland Water data resources) against threats. It is based upon a semantic metadata model of security concepts and their relationships originally expressed in DAML+OIL (Tan *et al.*, 2005) (see Figure 2). Particular constraints on these relationships are defined as policies. For example, a login screen (asset) uses confidentiality actions such as password access and encryption to protect against disclosure to other users. Policies define the constraints for the encryption safeguard action that is used. Another example is that senders may be required to sign messages they send, mediators are required to verify signatures on behalf of receivers and receivers are required use access control to receive messages. The partitioning of the semantic model enables the security conceptual model to be independent of the application requirements and the use of specific security mechanisms. A more complete and formal description of the framework and ontology is given in (Tan *et al.*, 2005).

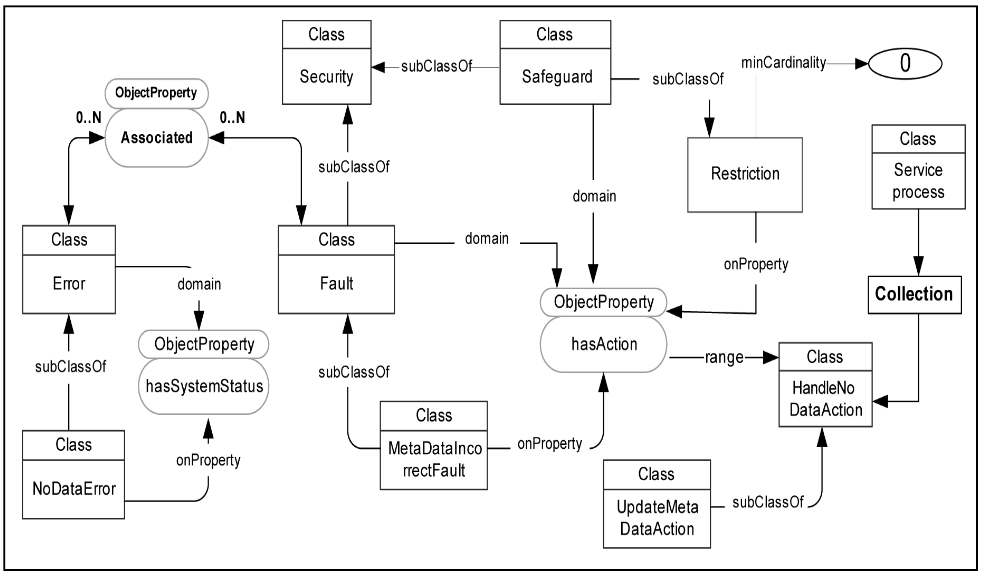
Figure 2 The security management ontology model



3.2 *Semantic model for safety management*

A fault object extension has been added to the security management ontological model to deal with the safety management of faults (see Figure 3). The safety management extension can extend the same policy and process models used for security management to model the constraints for safety and to interleave the safety management processes into the normal service operational processes. In order to interlink security with safety management, the security threat concept that can disrupt the operation of the application, *e.g.*, a denial of service threat to the Inland Water information asset, is related to the safety concept of a fault. As there is a lack of clear semantics and definitions to differentiate failure from error, we use these terms interchangeably. We take the term fault to mean the cause of an error.

Figure 3 The security management core conceptual ontological model includes a fault class to support safety management



The fault class in Figure 2 links directly to the security concept, while the error concept in Figure 3 is used to represent errors that have semantics with respect to an application context. This can entail wrapping lower level system exceptions to have higher level application semantics. Figure 3 also shows how errors can be interlinked with concepts defined in other semantic models, such as V-SAT and application service process models. The *hasSystemStatus* property links to the particular service process in which the error occurs. The *hasAction* property indicates the relevant actions that could be taken to handle the error. For example, a *No Data* error that occurs during a database query process could be linked to an error-handling action to refresh the resource descriptions in the directory service.

3.3 Profile model

The service profile describes how the service operates such that a service user or a service mediator can determine whether the service meets its needs or not. The service profile includes a description of what is accomplished by the service, any limitations on service applicability and quality of service, and any conditions that the service requester must satisfy to use the service successfully. The profile model conventionally acts in the presentation tier of the conventional information system but uses a directory service from the process layer (see Figure 1). A profile generally represents particular user or application configurations of service operations. These are constrained by service management policies defined in the profile. Specific application processes define patterns of service operations that occur as specified in the profiles. A key challenge is the need to interlink multiple processes. For example, successful access to an information resource by an inland water domain service may trigger a security authorisation process, whereas a failure to access the resource because it is unavailable may trigger an error-handling process to access a replicated copy. Security profiles are specific kinds of profiles that define relationships between security safeguards, application domain assets and threats that disrupt the use of those application assets.

Some services such as security span multiple domains. Composite profiles are defined as a collection of one or more safeguard, asset, and threat entities belonging to an open service. The composite entities can be seen as a single-managed object belonging to a service or domain (Tan and Poslad, 2004).

3.4 Policy model

A profile may also contain various policy rules, defining the security instantiations and any constraints. Management services, such as security, require constraints described as policy rules that enforce management decisions and that can initiate the relevant service control actions to manage the behaviour of the system. In multidomain services, the number of assets could be large, rendering it impractical to specify policies in terms of individual assets; it is easier to group assets into domains where the policy applies (Tan and Poslad, 2004). However, autonomous services component interaction is complex and may need specific policy management support. To optimise the number of policy objects into a manageable scope, an explicit ontological model provides the relationship mappings of security concepts and their specifications as profiles to permit the clustering of similar safeguards and policies into discoverable entries (Tan *et al.*, 2005).

According to Tan and Poslad (2004), an arbitrary predicate based on the value of an object attribute can be used to select sets of subject or target objects to which a policy applies. A search over all reachable objects within a distributed system to determine these sets is clearly impractical. The selection of objects is thus limited to be within a domain whose membership is known. However, the use of appropriate security ontologies, along with policies that specify the necessary preconditions for membership and domain information can enhance the management of distributed profiles in multiple domains.

Interacting services and agents may result in conflicting profiles because their policy constraints do not match. In many cases, it is not practical to prevent conflicts but it is practical to detect and resolve them. The idea of supporting policy negotiation for conflicting policies is potentially nonscalable and can introduce high overheads and complexity. Therefore, to enable alternative solutions for policy conflict resolution, the introduction of alternative policies (*n-arity*) may be defined, *e.g.*, in an optional ‘KeyLength’ can be supported.

Figure 4 Example of an *n-arity* policy

```
<profile:KeyInfo rdf:ID="keyInformation">
 <policy rdf:range="xsd:String">
      ( exists (?a (KeyLength ?a)) (or (= ?a 1024) (= ?a 512) )
 </policy>
</profile:KeyInfo>
```

The above policy defines the key bit size length of a particular credential where a disjunction operator supports either 1024 bits *or* 512 bits key lengths. The order characterises the precedence of the system and can also be used for resolving policy conflicts. Consequently, security profiles can be executed by the security application layer in which policy, privacy and cryptographic computation management can be enforced.

A security configuration policy specifies the conditions for safeguards and security requirements that external entities should abide by when utilising services offered by the system. Profiles occasionally require assets to adhere to certain conditions or rules. Hence preconditions are needed to offer a more precise formulation of expressions. However, precondition descriptions for specifying these input rules are vague or cannot be directly expressed in some semantic service description languages such as DAML-S. As a result, other constructs are used to express these conditions. These constructs are based upon introducing a string-based property, *e.g.*, (exists (?a (KeyLength ?a)) (or (= ?a 1024) (= ?a 512)) in Figure 4. Using a string-based property has its advantages; it provides the freedom for specifying the required condition of the process, and its proprietary representation. However, the use of standard semantic representation methods is encouraged to provide a more normative way of representing policies. This allows policies to be specified using a multitude of languages. Having highly configurable systems can easily lead to bad configurations, thus the support for heterogeneity would benefit from defining policies based upon an agreed security ontology.

3.5 Process and reasoning model

Service providers such as database resources must provide information to allow service users and middleware to invoke them. Service descriptions about the available service operations are needed as are the service groundings to access the service operations. But in addition, clients need to choreograph sets of external service invocations and internal processes into application specific workflows. These workflows involve activity-based orchestration of services and service selection. Different branches of the workflow may need to be dynamically selected in response to variations in operating conditions such as failures and in response to client preferences.

It is difficult for service providers to define the choreography as these may be client-, application- and environment-specific. Hence, service composition middleware is needed to aid clients to reason about the sequencing and the constraints of service actions in their application, security and safety management processes. Middleware can utilise formal logical reasoning implemented using Java Theorem Prover (JTP)[5] to choreograph service descriptions represented in ontologies such as DAML+OIL. Alternatively, reasoning could be represented using rule-based systems such as JESS (Java Expert System Shell) (Friedman-Hill, 2003) and integrated to work with ontology instances.

4 Application of the semantic security and safety management method to Inland Water

The Inland Water information retrieval from databases provides the application domain for illustrating the application of the semantic safety and security management framework. A semantic conceptual (ontological) model for the Inland Water domain has been derived from an analysis of use-cases and from consultation with domain experts during the EDEN-IW project (Stjernholm *et al.*, 2004). Various middleware services that interlink with different information instances and applications can be interlinked to work with the conceptual model. A security management model (Section 3.1) is used to protect inland water domain assets. A safety management service is used to handle failures in accessing Inland Water resources (Section 3.2). The profile and service process models in the security and safety management framework (Section 3.3) can be used to allow an Inland Water application to interlink with safety and security management pocesses.

4.1 Inland Water semantic model for integrating heterogeneous databases

Inland Water information held in databases is heterogeneous because of differences in: the data instances that are collected and stored (*e.g.*, data can be collected at different time intervals); the sets of queries that the database can answer; how the application information is modelled (information model design) and how these information (design) models are mapped to particular databases (database schema).

There are a variety of use-cases for these Inland Water resources such as *Find the concentration of determinant X in River Y at time T?* This query does not specify the name of the database that stores the information so this must in turn be queried in a metadata directory that records summary profiles of the data stored in the individual databases. River basins may cross multiple regions and can be stored in multiple databases so queries may need to harmonise data from multiple databases. There are various management applications that seek information to monitor the physical IW resources for changes across time and space and to check if policies to manage the physical IW resources are effective. There are also scientific applications that seek to check if the queried data can support the variability across different measurements of the same parameter.

The EDEN-IW Inland Water semantic model was originally represented using DAML+OIL and has been converted in part to OWL, the W3C Web Ontology Language.[6] Most complex ontology models are partitioned and typically consist of a conceptual model that can be shared across domains that is separated from the application services that use it (Spyns *et al.*, 2002). The motivation for this is to ease the maintenance of the ontology model so that new application commitments to use the ontology do not necessarily require changes to the core conceptual model. The EDEN-IW project uses an ontological model to support multiple processing applications, such as those that query single databases versus those that query and integrate the results from multiple heterogeneous inland water databases, and to support multiple data presentation viewpoints. The EDEN-IW ontology model is aligned to a general information systems model and partitioned into a data resource management layer, an application processing layer and a data presentation layer (Figure 1). Each of these layers contains specialised semantic information processes and data that can be interlinked using the semantic conceptual model (Stjernholm *et al.*, 2004). For example, the resource partition contains Local Database Views (LDV) for different inland water databases. The presentation partition defines an international-standard thesaurus to provide provenance for the terms and multilingual synonyms to present data in different languages.

4.2 EDEN-IW service grounding

A *Service Grounding* specifies the details of how services can be accessed. Typically, such a grounding specifies the communication protocols, message formats, and other service-specific details such as port numbers used in contacting the service. In addition, the grounding must specify for each type of input or output specified in the service model, an unambiguous way of exchanging data elements of that type with the service, that is, the serialisation techniques employed. The EDEN-IW services use the FIPA Agent model and agent management services to access the Inland Water services. Agents can distribute messages, process and reason about the semantic message exchange. Individual software agents interact using a generic multilevel application protocol based on speech acts called an Agent Communication Language (ACL) (Poslad and Charlton, 2001).

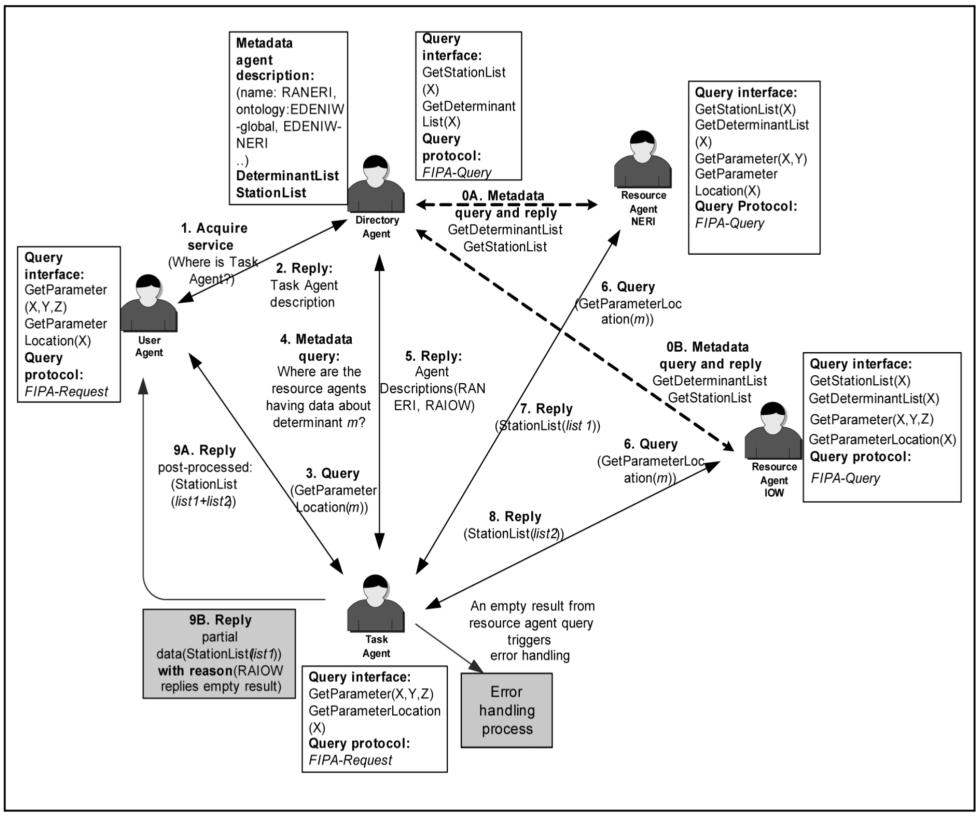
The system organisation is based upon the assigned roles of agents to support application services. There are four principal agent roles within the system: User Agent (UA), Directory Agent (DA), Task Agent (TA) and Resource Agent (RA). Non-agent services include ontology services, web services, and database resources. *The User Agent* supports the main presentation functions of query input and multilingual, multiformat results display. *The Task Agent* manages particular workflows that are triggered by particular Inland Water queries (businesses processes). It does this by business process modelling and coordinating any associated individual agent processing. Workflow management also entails handing errors that may arise. *The Directory Agent* maintains a repository of the active agent services and the connection information to reach those services. It also maintains Inland Water metadata, such as the stations and determinants, that summarise the content available in each particular database. *Resource Agents* provide access to particular relational databases to retrieve their IW data.

4.3 Inland water service use scenario

To make these ideas more comprehensible and appraisable, an application scenario is given in Figure 5. Heterogeneous Inland Water data resources must publish their service profiles in a directory service. The service profiles define the application processes supported. These can be interlinked to security processes and safety processes. In the Inland Water query composition scenario, the Task Agent or TA acts as a broker between User Agent, Directory Agent and heterogeneous Resource Agents. It matches the different service descriptions with the incoming query requirements and plans the interactions between agents. The Task Agent discovers services through a directory service. Through the profile description and policies, the Task Agent is able to reason about the profiles to understand the concepts and processes needed to support interoperability between disparate services.

User queries generate dynamic workflows and these require some reasoning during the execution of the application. For example, depending on the values entered in a particular user query, different databases need to be accessed and different kinds of postprocessing may be needed to convert measurements into a common set of units. When the User Agent queries the average of a determinant concentration in River X and River Y, the TA processes the request by initially understanding the core conceptual ontology and its relation to the local database ontology views. For example, the system determines the Determinant Concentration from the concepts of Determinant ID and Rivers, then derives the individual concentration values and forms the concatenation of an average concentration value from the different rivers.

Figure 5 Agent interaction for a user query. Numbered arrows in solid lines reflect the interaction of processing a user query in general, and the dotted lines refer to the metadata subquery. The shaded or coloured textboxes show one simple example of an exception such as no results being returned during an interaction between a TA (Task Agent) and a RA (Resource Agent). The *no results* triggers an error-handling process in the TA; as a result, the TA notifies the UA (User Agent) with an explanation.


Metadata agent description: (name: RANERI, ontology:EDENIW-global, EDENIW-NERI ..) DeterminantList StationList
Query interface: GetStationList (X) GetDeterminant List(X) Query protocol: FIPA-Query
Query interface: GetStationList(X) GetDeterminantList (X) GetParameter(X,Y) GetParameter Location(X) Query Protocol: FIPA-Query
Directory Agent
Resource Agent NERI
0A. Metadata query and reply GetDeterminantList GetStationList
1. Acquire service (Where is Task Agent?)
2. Reply: Task Agent description
Query interface: GetParameter (X,Y,Z) GetParameter Location(X) Query protocol: FIPA-Request
User Agent
6. Query (GetParameterLocation(m))
0B. Metadata query and reply GetDeterminantList GetStationList
4. Metadata query: Where are the resource agents having data about determinant m?
5. Reply: Agent Descriptions(RANERI, RAIOW)
7. Reply (StationList(list 1))
Query interface: GetStationList(X) GetDeterminantList(X) GetParameter(X,Y,Z) GetParameterLocation(X) Query protocol: FIPA-Query
Resource Agent IOW
6. Query (GetParameterLocation(m))
9A. Reply post-processed: (StationList (list1+list2))
3. Query (GetParameter Location(m))
8. Reply (StationList(list2))
An empty result from resource agent query triggers error handling
9B. Reply partial data(StationList(list1)) with reason(RAIOW replies empty result)
Task Agent
Query interface: GetParameter(X,Y,Z) GetParameterLocation (X) Query protocol: FIPA-Request
Error handling process

Sequences of agent interaction are depicted in Figure 5. More specifically, processing a typical user query in the EDEN-IW system consists of the following interaction:

1 UA queries DA for contact information of a TA to assist it with processing the query.

2 DA responds with the agent descriptions of any available TAs.

3 UA contacts a TA with a user query, *e.g.*, GetParameterLocation(m); this means *give me all the stations that have information about parameter m.*

4 After receiving the query, the TA analyses and extracts key word parameters, *e.g.*, parameter *m*, and uses these to issue a metadata query to the DA to locate the relevant data resources.

5 DA replies to the metadata query with resource agent descriptions, *e.g.*, RANERI (Resource Agent NERI) and RAIOW (Resource Agent IOW) are both registered as having data about parameter *m.*

6 TA sends the GetParameterLocation(m) to RANERI and RAIOW.

7 RANERI replies with StationList(list1).

8 RAIOW replies with StationList(list2).

9 TA combines the two results and replies to UA with a list of stations having data from two databases, *i.e.,* StationList(list1+list2).

4.4 Safety model

There are several errors that can occur during the query process given in the scenario in Section 4.3. A query may return no results, even when directory metadata exists that indicates results are available for that resource. Too many results can be returned causing a type of denial of service or incorrect results can be returned. Failures or errors in the computation processes and the network links that comprise the infrastructure can occur. For each error, there are multiple possible faults and workflows that lead to it. For example, a query returning no results (even when the metadata indicates that results should be available) can be caused by multiple application-level computation problems. These include: a database access problem; an SQL query to retrieve specific data cannot be generated because of a mapping problem between the core conceptual model and the local database ontology; the ontology files cannot be accessed or the advertised metadata descriptions of resources are incorrect. The classification and analysis of failures enable applications to understand failures and to better manage failures.

In order to illustrate and to evaluate our proposed semantic error-handling model, we focus on a particular error that can arise with respect to the scenario given in Section 4.3, *i.e.*, a No-Data error response to a query that we expect should return some results. Two possible faults that cause the No-Data error are considered: one of them being an incorrect generation of SQL queries, the other one being a mismatch between the resource description data advertised in the directory service agent and the data available in the corresponding data resource. The challenges of such an error handling model are firstly, to effectively and efficiently identify the actual fault that caused the error and secondly, to consider its effect on any currently executing application processes in order to handle the error in an appropriate way.

In a conventional error-handling system, the error triggers a specific error handler that is directly linked to that particular fault. There are two main difficulties with this simple approach. Firstly, there may be a one-to-many mapping between the error and fault. In order to handle the cause of the error, the particular fault that caused the error must be ascertained through initiating and assessing some test processes. Secondly, if the cause is not in this local code, the error must be propagated, possibly to a different application context, to other non-local error handling processes. As errors are often expressed using low-level programming system concepts, they do not have any meaning outside the local context of the currently executing code, making the error more difficult to be handled by other application processes.

Thus, in order to handle the effect the error has on the executing application processes, we have several requirements: to detect error events; to reason about the faults when one type of error can be caused by many faults; to issue test processes to help ascertain faults; to transform low-level error events into higher-level semantic events that have meaning to application-level rather than to just the system-level processes, and to consider how best to complete or abort an application process in which an error occurs.

Figure 6 shows the error-handling process; it is a flow-based error control in which the status of actions in a process flow is tested for errors. In our case, because much of the workflows and decision-making occur within the Task Agent, the error-handling subprocess is usually triggered in the Task Agent. In order to handle errors at the application level, the safety management ontology must be loaded, *e.g.*, an OWL representation of this ontology is imported and parsed. This ontology defines a set of known application error events and it relates infrastructure or system-level events to application-level events. It also relates each application event to a set of one or more faults or causes. An example of a fragment of the ontology is given in Figure 7. This ontology is consulted in order to identify the set of faults that relates to this error. To determine the specific fault that caused the error, reasoning is needed. This entails using some parameters derived from the process state when the error occurred to select further test actions to be performed and then reasoning about the results of these actions.

Figure 6 The semantic error-handling process flow

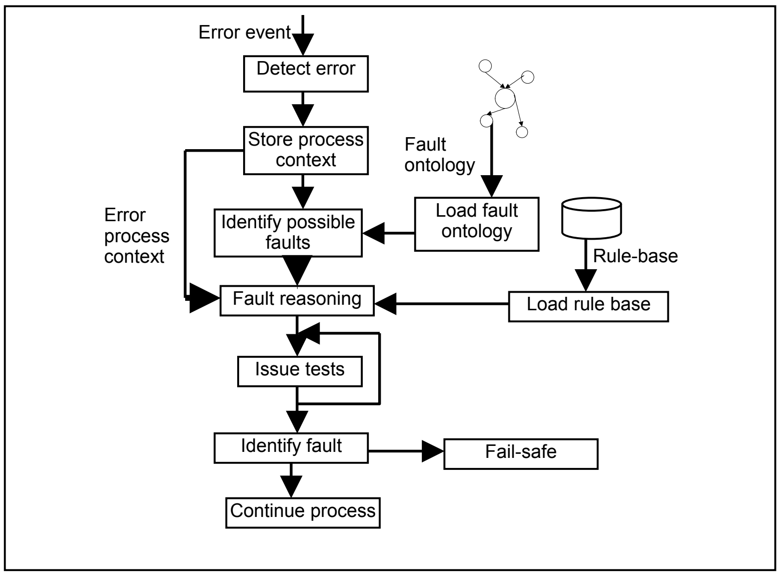


Figure 7 A fragment of the safety management ontology showing the relationship between error and fault

```
<owl:Class rdf:ID="error">
  <rdfs:subClassOf rdf:resource="event"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#associated"/>
      <owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
</owl:Class>

<owl:Class rdf:ID="fault">
  <rdfs:subClassOf rdf:resource="event"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#associated"/>
      <owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
</owl:Class>

<owl:ObjectProperty rdf:ID="CausedBy">
  <owl:inverseOf rdf:resource="#ResultIn" />
  <rdfs:domain rdf:resource="#fault" />
  <rdfs:range rdf:resource="#error" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="ResultIn">
  <owl:inverseOf rdf:resource="#CausedBy" />
  <rdfs:domain rdf:resource="#error" />
  <rdfs:range rdf:resource="#fault" />
</owl:ObjectProperty>
```

Figure 7 A fragment of the safety management ontology showing the relationship between error and fault (continued)

```
<error rdf:ID="ResourceAgentConfigurationWrong">
   <ResultIn rdf:resource="#ResourceAgentEmptyResult" />
</error>

<error rdf:ID="ResourceAgentEmptyResult">
   <CausedBy rdf:resource="#ResourceAgentConfigurationWrong" />
   <CausedBy rdf:resource="#DirectoryAgentMetadataMismatch" />
</error>
```

Whilst it seems at first attractive to represent such reasoning using some purely logic foundation, perhaps based upon some subset of first-order logic that may be inherently supported in some ontology models, such as OWL, it is more common practice to use a rule-based system as this often supports such features as conflict handling, default reasoning, rule prioritisation, rule naming and procedural attachments that a purely logic approach often does not support. Hence a rule-based system, *e.g.*, JESS (Friedman-Hill, 2003) is used to represent the rules and facts that define faults and application contexts that cause the error and a rule-based engine will be used to reason about these rules and facts. Essentially, the reasoning concerns performing tests on some conditions and invoking new functions to generate further outputs to test. This may be performed prior to the rule execution phase or triggered during rule execution. Currently, there is no prioritisation of the order in which possible faults are tested. Once the rule-based reasoning to identify the fault is correct, the current process is triggered to either continue or to failsafe returning the reason for terminating the current process.

Figure 8 An example JESS rule

```
(defrule ra-result-empty-ontology-incorrect-setting
  (error (name RAResultEmpty) (type ?errorType) (origin ?agentName) (OBJECT
?errorObject))
  (faultList ?$possibleFaults)
  (fault (?$possibleFaults ResourceAgentConfigurationWrong) (origin
?agentName))
  (ResourceAgent ?agentName)
  (not (outOfService ?agentName))
  =>
  (call (?agentName RAOntologyDiagonsis))
   (assert (InDiagonsis ?agentName))
  (printout RA-result-empty "check the ontology setting in " ?name crlf))
```

An example rule that is used to identify a fault for the No-Data error is given in Figure 8. This rule defines that if a *RAResultEmpty* error is detected and one of the possible faults for this error is *ResourceAgentConfigurationWrong* and if the error originates in a resource agent and if that resource agent is not in the out of service state, then an action to diagnose the ontology setting for that resource agent is called. The list of possible faults is generated from the fault management ontology. For each of the possible faults in the list, there exists one or more diagnosis actions that help to identify the actual fault.

5 Discussion and conclusion

5.1 Comparison of our system to surveyed systems

Our system adopts a semantic- and rule-based approach to integrate security management and safety management processes into application processes. Generally, existing systems either focus on security or safety aspects such as fault-tolerance but not both, and they do not always present a clear model of how these are integrated into application processes. In terms of security management, our (V-SAT) system is similar to several approaches whose characteristics are summarised in Table 1. V-SAT, however, unlike the systems in Table 1, uses all of the following: a semantic-based model that is also to exchange information (profiles) about externally supported security mechanisms; a policy model to dynamically manage security; a process model and a reasoning model to process constraints and rules. In addition, unlike the systems supported in Table 1, V-SAT also supports a model for safety management. Safety management adopts a flow-based failure control method in contrast to systems that focus on fault-tolerance via service and resource replication. Unlike conventional error handling, V-SAT adopts a semantic approach coupled with rule-based reasoning to identify errors that may stem from multiple possible faults and to allow errors to be semantically tagged to allow their effect to be assessed at the application process level rather than at the infrastructure level.

5.2 Discussion

In Section 3, a basic method that uses a semantic-based approach to manage the security and safety of distributed information processes has been described. In order to evaluate the approach, it has been applied to the domain of Inland Water environment information management in Section 4. This approach demonstrates that reasoning about multiple causes of detected errors that can be propagated and transformed when necessary into semantically tagged error events, can help to prevent the application from failing when these errors occur. We need to go beyond handling errors locally because in some cases, the propagation of the error can cause additional disruption to the application, causing further non-local errors, *e.g.*, interrupting the normal workflow and possibly causing time-outs. The benefits of this approach are hence, safer operation of complex distributed services in the face of disruptions. The average service downtime can be effectively reduced by execution of appropriate actions.

There are several important design issues to be taken into account in developing such a dynamic and open services model that can manage different kinds of application and management processes. For example: by incorporating and integrating an ontological concept model, policy model, an explicit application process model, a profile-based service description model and a reasoning model. Whilst it seems that ontology languages such as OWL have the expressivity in theory to represent and manipulate all of these different kinds of models, this adds to the complexity. There are a lack of specifications and tools to do this in practice and to integrate these with existing legacy processing and information systems. There also seems to be other more mature models for some of the specific parts of the system such as rule-based systems to support policy-based management and dynamic error handling. Whilst the motivation for using an ontology model is often quoted to promote a machine-readable, automated infrastructure, middleware services are vital to support these.

5.3 Conclusion

Safety and security in open-service environments poses some of the most challenging problems for the semantic web. The substantial increase in the number of open services presents problems of interoperability as these often use different metadata models and processes for security and safety management. The plethora of heterogeneous service specifications published by different interest groups in a web environment where anyone can publish can make it challenging to maintain service operation and access across multiple domains. A lack of adaptive management models supported in management processes can become a major impediment to ubiquitous service access and decentralised self-policing environments. The difficulty in propagating fault information in one part of the system to another part makes management decisions about cause and effect difficult. These challenges have steered the approach to managing security in traditional closed systems away from centralised and rigid methods towards more open and flexible approaches facilitated using semantic metadata exchange.

The main contribution of this paper is to show how an existing semantic approach to support the security management of open services can be extended to improve the safety of services using a semantic error management model. A semantic approach is particularly useful in providing extensibility and flexibility in future on-demand semantic web and web services, and in representing a promising way forward towards flexible safety and security in open environments. The model is based on a semantic-driven profile model that supports the interlinking of security and safety mechanisms and service processes by explicitly publishing the interaction requirements and the capabilities of service providers such as those that offer data resource access. The semantic model can also promote a reasoning model to support adaptive fault and security policy-based management and to support the choice of appropriate error handling.

References

Avizienis, A., Laprie, J., Randell, B. and Landwehr, C. (2004) 'Basic concepts and taxonomy of dependable and secure computing', *IEEE Transactions on Dependable and Secure Computing*, Vol. 1, No. 1, pp.11–33.

Bradshaw, J., Dutfield, S., Benoit, P. and Woolley, J.D. (1997) 'KAoS: toward an industrial -strength generic agent architecture', in J.M. Bradshaw (Ed.) *Software Agents*, Cambridge, USA: AAAI/MIT Press, pp.375–418.

Castro, M. and Liskov, B. (1999) 'Practical byzantine fault tolerance', *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI).*

Denker, G., Kagal, L., Finin, T., Paolucci, M. and Sycara, K. (2003) 'Security for DAML web services: annotation and matchmaking', *ISWC-2003*, Sanibel Island, Florida, USA, 20–23 October.

Desmedt, Y. (1994) 'Threshold cryptography', *European Transactions on Telecommunications*, Vol. 5, No. 4, pp.449–457.

Foster, I., Kesselman, C., Tsudik, G. and Tuecke, S. (1998) 'A security architecture for computational grids', *Proceedings of the* 5th *ACM Conference on Computer and Communications Security Conference*, pp.83–92.

Friedman-Hill, E. (2003) *Jess in Action: Java Rule-Based Systems*, in Action Series, Manning Publications, ISBN 1930110898.

Garay, J.A., Gennaro, R., Jutla, C. and Rabin, T. (1997) 'Secure distributed storage and retrieval', *Proceedings of the 11th International Workshop on Distributed Algorithms (WDAG).*

Herzberg, A., Mass, Y., Michaeli, J. and Ravid, Y. (2000) 'Access control meets public key infrastructure, or: assigning roles to strangers', *IEEE Symposium on Security and Privacy*, California, USA, May, pp.2–14.

Kagal, L., Finin, T. and Joshi, A. (2003) 'A policy based approach to security for the semantic web', in D. Fensel, K. Sycara and J. Mylopoulos (Eds.) *The Semantic Web – ISWC 2003*, Berlin: Springer, LNCS 2870, pp.402–418.

Lamport, L., Shostak, R. and Pease, M. (1982) 'The byzantine generals problem', *ACM Transactions on Programming Languages and Systems*, Vol. 4, No. 3, pp.382–401.

Lobo, J., Bhatia, R. and Naqvi, S. (1999) 'A policy description language', *Proceedings of AAAI*, Orlando, USA, pp.291–298.

Malkhi, D. and Reiter, M. (2000) 'An architecture for survivable coordination in large distributed systems', *IEEE Transactions on Knowledge and Data Engineering*, Vol. 12, No. 2, pp.187–202.

Nicol, D., Sanders, W. and Trivedi, K. (2004) 'Model-based evaluation: from dependability to security', *IEEE Transactions on Dependable and Secure Computing*, Vol. 1, No. 1, pp.48–65.

Poslad, S. and Charlton, P. (2001) 'Standardizing agent interoperability: the FIPA approach', in M. Luck, V. Marík, O. Stepánková and R. Trappl (Eds.) *Multi-Agent Systems and Applications, 9th ECCAI Advanced Course, ACAI 2001 and Agent Link's 3rd European Agent Systems Summer School (EASSS 2001), Selected Tutorial Papers, Lecture Notes in Computer Science 2086*, Prague, Czech Republic, 2–13 July, Springer, ISBN 3–540–42312–5, pp.98–117.

Sloman, M. (1994) 'Policy driven management for distributed systems', *Journal of Network and Systems Management*, Plenum Press, Vol. 2, No. 4, pp.333–360.

Spyns, P., Meersman, R. and Jarrar, M. (2002) 'Data modelling', *Proceedings of the Database and Information Systems Research for Semantic Web and Enterprises Workshop*, Amicalola Falls and State Park, Georgia, 3–5 April.

Stjernholm, M., Poslad, S., Zuo, L., Sortkjær, O. and Huang, X. (2004) 'The EDEN-IW ontology model for sharing knowledge and water quality data between heterogeneous databases', *18th Conference on EnviroInfo 2004 of German Informatics Society (GI)*, Geneva, Switzerland, 21–23 October.

Tan, J.J. and Poslad, S. (2004) 'Dynamic security reconfiguration in semantic open services environment', *Journal on Engineering Applications of Artificial Intelligence*, Vol. 17, No. 7, pp.783–797.

Tan, J.J., Poslad, S. and Titkov, L. (2005) 'A semantic approach to harmonising security models for open services', *Journal of Applied Artificial Intelligence*.

Notes

1 Web Service Security, http://www-106.ibm.com/developerworks/webservices/library/ws-secure/

2 *Web Services Definition Language*, http://www.w3.org/TR/wsdl

3 *Business Process Execution Language for Web Services*, Version 1.1, http://www-128.ibm.com/developerworks/library/ws-bpel/

4 *OWL-S Semantic Markup for Web Services Model*.

5 *Java Theorem Prover (JTP)*, http://www.ksl.stanford.edu/software/JTP/

6 *OWL Web Ontology Language Overview*, http://www.w3.org/TR/2004/REC-owl-features-20040210/