Security in WebCom: Addressing Naming Issues for a Web Services Architecture

Thomas B. Quillinan
t.quillinan@cs.ucc.ie

Simon N. Foley
s.foley@cs.ucc.ie

Department of Computer Science
University College Cork
Cork, Ireland

ABSTRACT

Supporting security in distributed systems is becoming more important with the ongoing work in grids, distributed middlewares and web services. Decentralised security architectures allow the stakeholders in these distributed computations, the providers of both compute resources and the applications executing on them, to have a say in how a computation progresses. One of the most important issues in creating authorisation policies is how the components of these distributed applications are *named*. Providing a consistent and flexible naming architecture allows more fine-grained and usable security policies to be created and enforced.

This paper introduces the naming architecture for the WebCom system. This architecture supports the addressing of all required information, with as much precision as is needed to create sophisticated authorisation policies.

Keywords

Naming; Trust Management; Distributed Systems; Web Services Security

1. INTRODUCTION

Providing a secure environment in which to execute distributed computations is an important area of ongoing research, especially when considering the emerging technologies such as the Grid [11, 12], cluster computing and middlewares such as COM/.NET [17], EJB [24] and CORBA [13], and especially within the realm of web services. In particular, managing authorisation decisions in these distributed environments is an important aspect of the overall security of these systems. Security architectures developed for distributed environments must consider the nature of the computations executing on available resources. Concerns potentially include the required authorisation guarantees needed for the data being operated on; the origin of the computation; it's future destinations, and so forth.

In practice, distributed systems require enforcement of security on a variety of architectures. Ideally, we need a security architecture that deals transparently with the distributed nature of the computation, instead of hindering it. Decentralised security architectures, such as Trust Management [2, 3, 7], provide an approach towards addressing this concern. Security policies are maintained by the stakeholders in the computations, the users who launch the computations and the computational resources that execute the jobs. Each of the stakeholders define what type of computations they permit.

Traditionally, security authorisation based on Trust Management in distributed systems is achieved through the embedding of relevant security calls within the distributed application [2]. A Trust Management interpretation for distributed computation might operate as follows: when a critical piece of code is reached, a check is first made by the resource scheduling the computation to ensure the computation is being scheduled to a suitable environment. The resource that is to execute the computations should also check to ensure that the job conforms to the security requirements of it's local policy. For example, a resource might require that no access is permitted by the computation to the local file system. However, making such checks requires that the security code be closely coupled with the functional code of the application. Ideally, such security code should be separated from the application code. This promotes the separation of security and functional concerns within the application.

Previous work has examined the use of Trust Management in providing a security architecture within the WebCom metacomputing environment [8]. WebCom provides support for heterogeneous applications, potentially made up of such web service/middleware technologies as EJB, COM (.NET) and Grid middlewares, such as Globus. WebCom uses the Condensed Graphs [18] computational model to control the sequencing of applications. In WebCom, security checks are separated from functional code. This provides the ability to separate the functional and security design of distributed computations. However, the preliminary implementation in [9] provided only a shallow embedding of trust management within the distributed computation.

In this paper we propose an architecture that allows greater control over the application components within the WebCom architecture. The security architecture proposed is independent of the underlying application code, providing the separation of security and functional concerns. We argue that to have more complete control over a computational component, you must be able to name all aspects of it. To address this, we propose a naming architecture for the WebCom system. This provides a means to capture the needed detail to address fine-grained security policies. These WebCom names facilitate control over the individual fragments of distributed applications. Depending on the security policies defined by

ACM Workshop on Secure Web Services, October 29, 2004, Fairfax VA, USA.

the compute resources, the computation may be restricted based on, for example, the application, the functional type of computational fragments, the input data, the result data and the historical path of execution. The naming architecture allows a systematic way to name, and therefore control, authorisation for COM, CORBA and EJB middlewares. This in turn provides the basis to create comprehensive security policies for web service-based applications.

In [15], the authors identify the problems with different infrastructures, requirements and semantics of different web services technologies. We believe that these problems can be reduced to a naming problem: How does each user of the web service refer to the attributes of that particular service? Having this ability to name each service available helps the user to create more useful policies about, for example, what service to use depending on their requirements at that time.

The contribution of this paper is a naming architecture that can be used to specify complex policies to control the execution of applications. The architecture has been used to create policies for other components of the WebCom system, such as the fault tolerance and load balancing managers. The names can be used to hold as much, or as little contextual information as necessary to make policy decisions.

The remainder of this paper is organised as follows: Section 2 provides some background on the Condensed Graph model of computation and Trust Management systems. These graphs provide the framework for specifying distributed applications that can form the basis of web services. Extending these graphs to provide a naming architecture for distributed computations is examined in Section 3. We then describe, in Section 4, how these names are used within the WebCom system. Finally, we discuss the results and draw some conclusions in Section 5.

2. BACKGROUND

In this section we introduce the Condensed Graphs model of computation. Section 4 will describe an architecture that uses this model. We will also provide some background information about Trust Management and how it relates to distributed computation.

2.1 Condensed Graphs

The heart of the WebCom system is the Condensed Graphs computational model that it employs [18]. Applications are coded as hierarchical graphs that provide a simple notation in which lazy, eager and imperative computation can be naturally expressed [18]. An advantage of developing distributed applications using Condensed Graphs is that their implementation (graph) can be coded independently of the underlying system and/or network architecture.

There are two types of distributable operation: nodes that represent atomic tasks and condensed nodes that represent subtasks encapsulated as subgraphs. Atomic operations are value-transforming actions and can be defined at any level of granularity, ranging from low-level machine instructions to mobile-code programs such as applets, middleware/web service components (such as COM/.NET, EJB, Globus), or COTS components. Atomic operations need not address synchronisation or concurrency concerns: such details are implicitly specified in the Condensed Graph and managed by WebCom.

Condensed Graphs can be used to solve a range of different problem domains, from highly parallel computational tasks [19] to distributed workflow applications [8]. Specifying an application as a Condensed graph allows the implicit parallelism of that application to be exploited transparently. Results generated by the execution of each node flow along the arcs to the next node in the graph.

EXAMPLE 1. *The Condensed graph shown in Figure 1 defines a very simple purchase ordering application. This application is specified as a Workflow; each node is implemented as a functional component.*

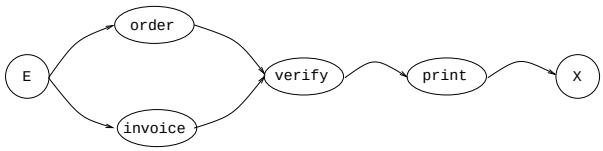

Figure 1: A simple purchase processing application, specified as a Condensed Graph.

The application operates as follows: When a product is required, it is ordered by an authorised staff member, using the `order` *node. When the invoice for payment arrives, it is inputted by the finance department, through the execution of the* `invoice` *component. These results (the order and invoice documents) flow along the graph to the* `verify` *node. The order and invoice are verified by a manager, who authorises the payment for the product. A cheque is then printed through the execution of the* `print` *node.*

Any node in this graph could itself be a Condensed Graph. Furthermore this graph, called `PurchaseOrder`*, could itself be part of another Condensed Graph. Recursive graphs are also possible. The PurchaseOrder graph could include an instance of itself with the graph definition.*

It should be noted that this is an extremely simple example of a Condensed Graph application. More complex examples can be found in [20] and [16]. △

In the original Secure WebCom [8], authorisation decisions were made based only on the operational function of a condensed node, such as `order` or `verify`. This design provides a shallow embedding of Trust Management into WebCom. The policies that can be defined are limited by the amount of information available to the security infrastructure, in effect the trust decisions that can be made are coarse grained. For example, the policy could dictate that the `verify` node may be executed by a specific manager; however, it is not possible to define a policy based on the size of that order. It may be important to have the ability to define a policy stating that, for example, the `verify` operation could be executed by a specific manager when the order value is over a defined limit. Providing the ability to specify these security conditions requires a new security architecture within Condensed Graphs.

2.2 Trust Management

Unlike identity based authorisation systems, such as those using X.509 [5] certificates, where authorisation is based on linking a detailed identity to a public key, Trust Management addresses the need to associate abilities to public keys.

Trust Management systems have a number of advantages compared to the traditional systems created using X.509. Policies and certificates are created and maintained separately from the application in a very natural way. The attributes used within the policies/certificates are application defined, and they are represented in a free-form fashion, allowing the application designer to decide what characteristics are required. Changing the format of the attributes does not require changes to the trust management system used. By removing the traditional lookup of an identity's authority, and instead representing that authority within the certificate, applications no longer need to consider the security of where this authority is stored. An additional benefit of utilising a trust management

system within applications is that designers and implementers of these applications are required to consider trust management applications explicitly. This in itself encourages good practices when considering the overall security of such applications. Trust management policies are easy to distribute across networks, helping to avoid reliance on centralised configuration of distributed applications.

Both KeyNote [1, 2] and SPKI/SDSI [7] are expressive and flexible trust management schemes that provide simple credential notations for expressing both security policies and delegation. A standard Application Programming interface (API) is used by application to make queries about whether security critical requests (to the application) have authorisation or not. The formulation and management of security policies and credentials are separate from the application, making it straight forward to support trust management policies across different applications.

3. NAMING IN CONDENSED GRAPHS

We argue that the key to acquiring fine-grained control of condensed graph applications is how the elements of these applications are *named*. If you have the ability to name an object, then you have the ability to write authorisation statements regarding it. Naming distributed objects is not a new topic. Work such as [21] provide solutions towards naming of distributed objects, however we are more interested in naming evolving computations rather than static objects. Achieving this fine-grained control requires a complete and consistent naming architecture for Condensed Graphs. A complete name for a Condensed Graph must take into consideration the aspects of each node in the graph. Nodes have functions, inputs and outputs. Inputs can either be values or other nodes. Outputs can be values, when the node performs some computation. For certain nodes the destination of the node's result is important. Each node is defined in a graph, and so this attribute must also be represented. Finally the location of the node, where it is, or will be, executed, also forms part of the characteristics of the computation.

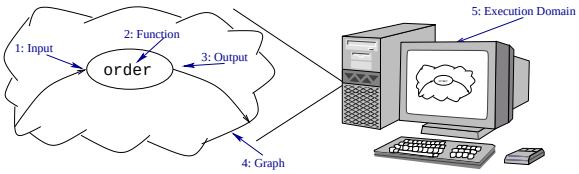

Figure 2: Components of a Condensed Graph name.

SDSI [22] introduced of linked local Namespaces. In SDSI all principals (keys) are equal. Each key has it's own name-space, as in PGP [25]. When one principal refers to an object in their own name space, they define that object's name themselves. For example, if Alice has a computer, she calls it "Computer". Bob may also have a computer, and he too may refer to it as "Computer". As Alice and Bob are separate principals this is perfectly acceptable. However, how does Bob refer to Alice's Computer? Suppose Bob knows Alice simply as "Alice"; Local naming provides the ability to use names from other namespaces. Bob, therefore refers to Alice's Computer as "Alice's Computer". More precisely: the object Alice refers to as "Computer".

Applying this principle to Condensed Graphs, we can define, for example, the input to a node as *"Node's Input"*, from the perspective the graph. Abstracting this name further we get *"Graph's Node's Input"*. This name can be abstracted to hold as much contextual information to enable the definition of a unique name for each portion of a graph. For Example: *"Country's University's Faculty's Alice's Computer's Graph's Node's Input"* coming from *"Country's University's Faculty's Bob's Computer's Graph's Node's Output"*. Using this approach, we can name every portion of the graph in as much detail as is required to uniquely identify it.

The components of a name are the input, function, output, parent graph and the execution domain. Combining all these basic elements of a node into a complete name, we create a definitive means to refer to *that* instance of a node in a graph currently executing on a particular resource. Thus, from Alice's perspective, a version of the `order` node from Figure 1 executing on her computer can be specified as shown in Figure 3.

{*Input, Function, Output, Graph, Domain*}:
{(Computer's PurchaseOrder's Order's `Input`), (Computer's PurchaseOrder's `Order`), (Computer's PurchaseOrder's Order's `Output`), (Computer's `PurchaseOrder`), (`Computer`)}

Figure 3: The Name of the `order` node from Alice's perspective

From Bob's perspective, the components of the name must specify the principal Alice, in whose namespace these name-components exist. Local naming provides the ability to store the required detail to uniquely identify each portion of the nodes in as much detail as is necessary.

We represent WebCom names as S-Expressions [23]. S-Expressions are lisp-like data structures for representing complex data. They are either byte-strings (octet-strings) or lists of other S-Expressions. They were designed to be a compact, human-readable efficient and transportable mechanism for storing data. Using S-Expressions to represent WebCom names, we can create authorisation statements to control computations in as fine-grained a manner as is required.

Figure 4 provides the format of a WebCom name. All parts of the name are optional, a name can be represented by a combination of any of these fields or by a simple S-Expression. There can be one or more `input` and/or `output` fields when the `inputs` and/or `outputs` fields, respectively, are present.

webcomname ::=

```
(WebComName
  [(domain webcomname)]
  [(graph webcomname)]
  [(function webcomname)]
  [(inputs
     (input webcomname)
     {(input webcomname)}
   )]
  [(outputs
     (output webcomname)
     {(output webcomname)}
  )]
)
```

webcomname ::=

```
(WebComName S-Expression)
```

Figure 4: The format of a generic WebCom Name.

EXAMPLE 2. *Figure 5 shows an example of a WebCom name for the node `order` from the Condensed Graph shown in Figure 1. In this example all possible fields are present. The node referred to is the instance present in University's Faculty's Alice's Computer.*

```
(WebComName
  (domain (ref: University (ref: Faculty (ref: Alice Computer))))
  (graph (ref: University (ref: Faculty (ref: Alice
              (ref: Computer PurchaseOrder)))))
  (function (ref: University (ref: Faculty (ref: Alice
                 (ref: Computer (ref: PurchaseOrder Order))))))
  (inputs
    (input (ref: University (ref: Faculty (ref: Alice
                (ref: Computer (ref: PurchaseOrder  E))))))
  )
  (outputs
    (output (ref: University (ref: Faculty (ref: Alice
                 (ref: Computer (ref: PurchaseOrder Verify))))))
  )
)
```

Figure 5: An S-Expression version of the name for the `order` node.

Alternatively, a representation of this node could include less information, or could be a simple S-Expression, for example:

```
(WebComName (ref: Alice Order))
```

This represents a node that Alice refers to as "Order". In Bob's namespace, this would be referred to as "Alice's Order" [1]. △

While these names provide the contextual detail required to enable authorisation policies to be created before the computation takes place, it is immediately obvious that the size of these names will cause them to become unusable in computations of a non-trivial nature. A consistent system is required to provide a canonical form for these names, yet still containing enough detail to allow informed authorisation decisions to be made. We address this concern using "reduction rules".

3.1 Reduction Rules

The contextual detail provided by WebCom names come at the cost of possible redundant information being stored in the name. For example, in Figure 5, the graph name (`PurchaseOrder`) is mentioned four times. This cost can be lessened through the use of *reduction rules*. A reduction rule is a heuristic by which an abstract name can be translated into a more compact, but equivalent form. These rules must ensure consistent names are produced. A basic formalisation of reduction rules into a simple notation provides the representation:

$$Reduce(OriginalWebComName) \rightarrow (ReducedWebComName)$$

EXAMPLE 3. *The Function Reduce(Name) in Equation 3 converts a complete WebCom name into one relying on the Function to represent the node.*

$$Reduce((Domain)(Graph)(Function)(Inputs)(Outputs)) \rightarrow (Function)$$

This reduction rule converts a full WebCom name into one that provides the detail used to refer to nodes in the original Secure WebCom system. Applications can define their own reduction rules, based on the amount of detail that is required for the security policy. For example, in our Purchase Ordering system, we would like to keep the information regarding the inputs and outputs to allow policies to be defined such as: If the input to `order` *is greater than $100, then the* `verify` *node must be scheduled to the CFO.* △

[1]Note, the `ref:` keyword is used in S-Expressions as the formalisation of the "'s" relationship.

WebCom names can be used to express precise details about nodes in Condensed Graphs. However, this detail is often not required when making authorisation decisions. Reduction rules compress the information in a consistent way. For example, the common information in each part of the `order` node's name from Figure 5 could be removed. Figure 6 the name from Figure 5 after such a transformation has been applied.

```
(webcomname
  (domain (ref: University
            (ref: Faculty
              (ref: Alice Computer))))
  (graph  PurchaseOrder)
  (function Order)
  (inputs (input  E))
  (outputs (output Verify))
)
```

Figure 6: The reduced name of the `order` node, after removal of redundant information.

The scheme outlined above provides a basis for considering the component parts of the node names. Outputs from one node form the inputs to another node. We can replace the input references with the name of the node that provides the input. Similarly outputs can be replaced with the names of the destination nodes of the outputs. This allows more precision within the naming architecture. However this also creates a loop, each node has the name of every other node in the graph as part of it's own name.

Again, reduction rules are used address this issue. We define a rule to limit the information stored in each nodes name. For example, such a rule could take the inputs and outputs of every node and replace them with a representation of the relevant node name containing just the *Function* of that node, as shown in Figure 6.

Applications can use part or all of a fixed set of reduction rules to provide the details required for proper authorisation decisions to be made within those applications. Different rules may be used for different applications, depending on the computational detail required for specific policies. In this paper, we consider only simple reduction rules. Developing more complex rules is a topic of ongoing research.

These policies need not be limited to the security arena. Work is ongoing towards providing support, using the naming architecture, for the load balancing[20], fault-tolerance[16], messaging and logging architectures in WebCom. For example, in a debugging situation, it may well be desirable to have as much detail as possible concerning the results returned by each node, on what resource the

nodes executed in order to track where, and under what circumstances, a computation has failed.

3.2 Policy Examples

We have integrated both the KeyNote [2] and SPKI/SDSI [7] Trust Management systems into WebCom. This provides authorisation support for Condensed Graphs. While local names have been implemented in SPKI / SDSI [6], adding support for the naming architecture of Condensed Graphs into KeyNote is simply a matter of specifying them as part of the conditions field. However, for the sake of clarity, in this paper we use the SPKI versions of the credentials[2].

EXAMPLE 4. *A SPKI credential that captures the required portions of the WebCom name to authorise the* `verify` *node is shown in Figure 7.*

```
(cert
 (issuer (ref: University
                (ref: Faculty Alice)))
 (subject (ref: University
                (ref: Faculty Bob)))
 (propagate)
 ((tag
   (webcomname
    (domain (ref: Alice Computer))
    (graph PurchaseOrder)
    (function verify)
    (inputs
     (input order)
     (input invoice)
    )
    (outputs
     (output print)
    )
 ))
 (not-before "2004-06-01_00:00:00")
 (not-after "2004-08-15_23:59:59")
)
```

Figure 7: A SPKI credential, written by University's Faculty's Alice, delegating the authority to execute the `verify` node to University's Faculty's Bob.

The credential in Figure 7 specifies the required details necessary in this case to identify an instance of the `verify` *node from the* `PurchaseOrder` *graph, with inputs of* `order` *and* `invoice`*, with the output going to the node* `print`*, running in the domain "Alice's Computer". Further detail, such as the particular WebCom environment, or the particular inputs to the graph could be codified.*

If a graph contains multiple `verify` *nodes, a more generic credential, created using the reduction rule from Equation 3, and shown in Figure 8, would suffice. This limited form of encoding provides an equivalent form to how the original Secure WebCom authorisation architecture operated.*

The credential in Figure 8 allows University's Faculty's Charles to execute any `verify` *node in any graph. University's Faculty's Charles could further delegate parts of this authorisation. For example, he could choose to limit the delegatee to executing a specific node (using the input and output keywords) or specific graphs. This allows the creation of sophisticated yet consistent security policies, giving the principals the power to define what parts of their authorisation that they wish to delegate.*

[2]Both SPKI and KeyNote plan to use XML-based credential formats in the future.

```
(cert
 (issuer (ref: University
                (ref: Faculty Alice)))
 (subject (ref: University
                (ref: Faculty Charles)))
 (propogate)
 (tag (webcomname (function verify)))
 (not-before "2004-06-01_00:00:00")
 (not-after "2004-08-15_23:59:59")
)
```

Figure 8: A SPKI credential, written by University's Faculty's Alice, delegating the right to execute all `verify` nodes in the PurchaseOrders graph to University's Faculty's Charles.

```
(cert
 (issuer (ref: University
                (ref: Faculty Charles)))
 (subject (ref: University
                (ref: Faculty Dave)))
 (tag
   (webcomname
    (function verify)
    (graph PurchaseOrder)
   ))
 (not-before "2004-06-01_00:00:00")
 (not-after "2004-08-15_23:59:59")
)
```

Figure 9: A SPKI credential, written by University's Faculty's Charles, delegating the authority to execute the `verify` node to University's Faculty's Dave, adding additional restrictions.

Figure 9 shows such a credential, written by University's Faculty's Charles for University's Faculty's Dave, specifying that the `verify` *node must be in the* `PurchaseOrder` *graph, and also preventing Dave delegating this authorisation further.* △

Other issues must also be considered when developing a complete naming infrastructure for Condensed Graphs. As computations evolve and partial results are integrated into the graph, these results have to be represented in the names. In our architecture, reduction rules are applied to integrate these results into the names. This has the advantage that a security check can be made based on the path that the computation has taken to this point. This helps to provide support for the creation of policies such as Chinese Wall policies [4].

EXAMPLE 5. *Condensed Graph applications can also contain middleware components. We can refer to these components using the same naming architecture. Casting these middleware components as WebCom names ensures that naming the computation remains consistent throughout, regardless of the underlying functional implementation of the nodes in the graphs. Figure 10 shows a DCOM object represented as a WebCom name.*

△

It is apparent that more powerful policies can be created using different reduction rules to increase or decrease the granularity of the name as required. It is important, however, that these reductions be consistent with the security policies of the resources involved in the computation. The development of these policies is a topic of ongoing research.

```
(webcomname
  (domain
   (ref: http://www.cuc.ucc.ie/
   (ref: DCOM
   (ref: Word.Application)))
  )
  (function (ref: OBJREF
   {000209FF-0000-0000-C000-000000000046}))

)
```

Figure 10: A DCOM object represented as a WebCom name.

4. SUPPORTING SECURITY IN WEBCOM

The architecture of WebCom [19], is shown in Figure 11. WebCom is made up of several modular components: Execution Engine Modules, Connection Manger Modules, Fault Tolerance Modules[16] Load Balancing Modules[20] and Security Manager Modules.

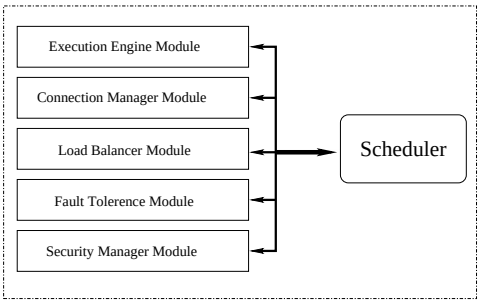

Figure 11: WebCom Architecture Diagram

The Scheduler initialises the required modules and handles communication between them. Execution Engine Modules take functional components and executes them; Connection Manager Modules handle communication between WebCom environments; Load Balancing and Fault Tolerance Modules handle faults and balances the load over the clients of the server; Security Manager Modules check each executable component and ensure adherence to the local system security policy. Each WebCom environment can have as many of each type of module as required; each is consulted where appropriate. For example a WebCom environment might have two Execution Engine Modules, one handling Condensed Graph Applications and the other acting as a gateway to a Globus grid. When a Globus job is uncovered, it is targeted to the Globus Engine Module. In a distributed system, such as the Grid or a metacomputer, there are many WebCom environments interoperating.

Condensed Graphs are stored and transported as XML documents. The WebCom names are embedded within these graph definitions. The graph definition of the Purchase Order graph from Figure 1 is shown in Figure 12.

The original implementation of the Secure WebCom system [8] provided support for authorisation using the KeyNote Trust Management system. The structure of the authorisation architecture provides a means to make trust mediations based on the function of the nodes in Condensed Graphs. Clients provided credentials to their WebCom parent. These credentials were used by the WebCom server to decide where to schedule each node in the graph.

This architecture provides a means to control the computation at the node level. This involved a shallow embedding of the Trust Management system into Secure WebCom, authorisation checks could only be made based on a limited sense of how the computation was proceeding. This architecture did not allow trust decisions based on a more fine-grained concept of these nodes.

Nodes can potentially take input data, execute some function on this data and return results. Providing a means to capture more specific information about the computation would allow more fine-grained security policies to be developed. These policies could allow, for example, a policy specifying that certain machines can execute a node depending on the value of the inputs. Support for such policies is implemented within the Security Manager Module in WebCom.

4.1 Security Manager Module

We will now examine how the Security Manager Module functions in more detail.

When the scheduler uncovers a node for remote execution, it requests a suitable client from its pool of connected clients. In order that such a client is found, the security manager module(s) and load balancer module(s) must first negotiate to locate a client that is both authorised to execute the particular node and is sufficiently unloaded to perform the execution. This negotiation attempts to discover the least loaded client that is authorised to execute the work, according to both the load-balancing and security policies. If no suitable client is found, the node is placed back in the scheduling queue until such a client is available.

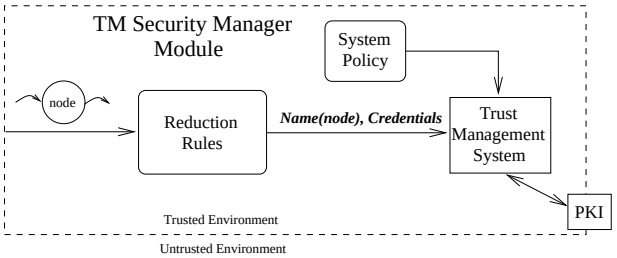

Figure 13: Trust Management based Security Manager Module

Integrating the naming architecture into the WebCom architecture is achieved through the creation of a security manager module to support it. Such a module is shown in Figure 13. This security manager takes a node, extracts it's *WebCom name* and reduces it according to the reduction rules of the particular application. This name forms part of the query to a trust management system, along with the system policy and the appropriate client's credentials. If the trust management system finds a trusted path from the system policy to the client's key, it notifies the security manager. The security manager in turn notifies the scheduler that a suitable client has been found. When a result is returned from a client, before it is integrated into the execution, a security check is performed to ensure the system policy has been upheld.

EXAMPLE 6. *In the purchase ordering application from Example 1, the system policy could dictate that an ordinary employee can only order goods to the value of $100. If the* `order` *node is executed and the result indicates that this policy has been breached, the system will reject the result. In WebCom, the node would be rescheduled for execution, just as if it had failed for functional reasons.* △

Through the provision of these security mediations, both before and after the execution of nodes in the graph, we gain the ability to create subtle security policies that constrain the flow of the

computation to ensure it complies with the local security policies of the resources. WebCom supports messaging between modules, both inside the same environment and with other WebComs across the network. Each module defines messages that they support. Using this mechanism, security managers can communicate. They can use this to provide speculative authorisation checks. For example, this permits checking beforehand whether a web service would allow itself to be used in the current computation, without first scheduling that component to that resource. The messaging system also provides a rudimentary method of exchanging credentials between WebCom environments. If a security manager doesn't have all of the credentials that are necessary to complete an authorisation check, then it can request credentials from its peers.

5. DISCUSSION AND CONCLUSIONS

We have developed several different applications using the WebCom system. For example, providing integration with middlewares such as COM, CORBA and EJB has allowed creation of web service applications in Condensed Graphs using components from these systems. The WebCom architecture is used to provide integration between web services, providing the "glue" connecting one service to the next. This integration extends to the security architectures of these systems [10].

In this paper we have proposed a consistent framework for naming computations with as much precision as is required to make informed authorisation policies. Naming computational components in a consistent and, when required, unique fashion allows a deep embedding of authorisation control into the distributed computation. These names provide a basis for developing fine-grained security policies, with the ability to control how and where computations are executed.

The issues of naming in the WebCom architecture applies equally to web services. Developers who wish to integrate many services into an application must use the name defined by the providers of that service. Identifying particular services in security polices becomes complex and tedious. Using a consistent naming infrastructure allows more understandable policies to be generated.

Using these names to integrate web services from different providers allows the creation of sophisticated security policies. However, these policies need not be security related, WebCom provides the architectural support for policies related to fault tolerance, load balancing, etc. For example, depending on the results from one web service, another service could be checked to verify that result.

The advantages of the naming architecture in WebCom can be applied more generally. These names could be used in more conventional authorisation architectures. However, we believe that the nature of the Condensed Graph model provide unique advantages to WebCom. Application components, specified as nodes in a Condensed Graph do not require security code to be embedded with the functional code. These security checks are implemented externally, providing a natural separation of security and functional concerns. This also has the advantage that security policies can be modified to suit the local environment, without requiring functional code to be recompiled. Research is ongoing towards using the naming architecture to support more complex policies in the future. This work primarily involves the development of more advanced reduction rules for use within WebCom.

Acknowledgements

The support of the Informatics Research Initiative of Enterprise Ireland is gratefully acknowledged. The authors wish to thank the members of the Centre for Unified Computing in UCC, especially Philip Healy and John Morrison, without whose support this work would not have been possible. Finally, we are grateful to Eoin Huggard for his work developing JKeyNote [14].

6. REFERENCES

[1] M. Blaze. Using the KeyNote trust management system. `http://www.crypto.com/trustmgt`, December 1999.

[2] M. Blaze et al. The keynote trust-management system version 2. Sept. 1999. Internet Request For Comments 2704.

[3] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proceedings of the Symposium on Security and Privacy*. IEEE Computer Society Press, 1996.

[4] D. Brewer and M. Nash. The Chinese Wall security policy. In *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, pages 206–214. IEEE Computer Society Press, May 1989.

[5] CCITT Draft Recomendation. *The Directory Authentication Framework, Version 7*, Nov. 1987.

[6] C. Ellison. Spki requirements. Referral for Comment (RFC) 2692, Internet Engineering Task Force, September 1999.

[7] C. Ellison et al. SPKI certificate theory. Sept. 1999. Internet Request for Comments: 2693.

[8] S. N. Foley, T. B. Quillinan, and J. P. Morrison. Secure component distribution using webcom. In *Proceeding of the 17th International Conference on Information Security (IFIP/SEC 2002)*, Cairo, Egypt, May 2002.

[9] S. N. Foley, T. B. Quillinan, J. P. Morrison, D. A. Power, and J. J. Kennedy. Exploiting keynote in webcom: Architecture neutral glue for trust management. In *Proceedings of the Nordic Workshop on Secure IT Systems Encouraging Co-operation*, Reykjavik University, Reykjavik, Iceland, Oct. 2000.

[10] S. N. Foley, T. B. Quillinan, M. O'Connor, B. P. Mulcahy, and J. P. Morrison. A framework for heterogeneous middleware security. In *Proceedings of the 13th International Heterogeneous Computing Workshop*, Santa Fe, New Mexico, USA., April 2004. IPDPS.

[11] I. Foster et al. A security architecture for computational grids. In *5th ACM Conference on Computer and Communications Security*, 1998.

[12] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[13] T. O. M. Group. Common object request broker architecture (corba/iiop). Technical report, The Object Management Group, December 2002. http://www.omg.org/technology/documents/formal/corba_iiop.htm.

[14] E. Huggard. JKeyNote. Fourth year project, University College Cork, Ireland, April 2003.

[15] L. Kagal, T. Finin, and A. Joshi. A policy based approach to security for the semantic web. In *Proceedings of the 2nd International Semantic Web Conference (ISWC2003)*, Sundial Resort, Sanibel Island, Florida, USA, 2003. Springer-Verlag.

[16] J. J. Kennedy. *Design and Implementation N-Tier Metacomputer with Decentralised Fault Tolerence*. PhD thesis, University College Cork, Ireland, 2004.

[17] Microsoft Corporation. *Microsoft Platform SDK. The Component Object Model. Microsoft Developer Network.*, 0.9 edition, October 1995. http://www.msdn.microsoft.com.

[18] J. Morrison. *Condensed Graphs: Unifying Availability-Driven, Coercion-Driven and Control-Driven Computing*. PhD thesis, Eindhoven, 1996.

[19] J. P. Morrison, B. Clayton, D. A. Power, and A. Patil. Webcom-g: Grid enabled metacomputing. *Neural, Scientific and Parallel Computations Journal.*, 2004. To Appear.

[20] D. A. Power. *A Framework for: Heterogeneous Metacomputing, Load Balancing and Programming in WebCom*. PhD thesis, University College Cork, Ireland, 2004.

[21] S. Radia. Naming policies in the spring system. In *Proceedings of the 1st International Workshop on Services in Distributed and Networked Environments.* Sun Microsystems, Inc., IEEE, 1994.

[22] R. Rivest and B. Lampson. SDSI - a simple distributed security infrastructure. In *DIMACS Workshop on Trust Management in Networks*, 1996.

[23] R. L. Rivest. S-expressions. Technical report, Network Working Group, May 1997. Internet Draft: http://theory.lcs.mit.edu/ rivest/sexp.txt.

[24] Sun Microsystems. *Enterprise JavaBeans(tm) Specification, Version 2.1*, June 2003. http://java.sun.com/products/ejb/docs.html.

[25] P. Zimmermann. *The Official PGP Users Guide*. MIT Press, 1995.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cg:graphdefs SYSTEM "http://cuc.ucc.ie/xml/cg.dtd">

<cg:graphdefs xmlns:cg="http://cuc.ucc.ie/xml/cg">
  <cg:node name="E">
   <cg:operandport strictness="strict"/>
   <cg:operatorport operator="cg.engine.EnterOperator"/>
   <cg:destinationport>
     <cg:destination nodename="Order" portnumber="0"/>
     <cg:destination nodename="Invoice" portnumber="0"/>
   </cg:destinationport>
 </cg:node>
 <cg:graphdef name="PurchaseOrder">
    <cg:node name="Order">
      <secname:securename xmlns:secname="http://cuc.ucc.ie/xml/secname">
        <secname:domain name="(domain (University (Faculty (Alice Computer))))"/>
        <secname:graph name="(graph cg.nl.po.PurchaseOrder)"/>
        <secname:destination name="(output cg.nl.po.VerifyOp)"/>
        <secname:function name="(function cg.nl.po.OrderOp)"/>
      </secname:securename>
      <cg:operandport strictness="strict"/>
      <cg:operatorport operator="cg.nl.po.Order"/>
      <cg:destinationport>
        <cg:destination nodename="Verify" portnumber="0"/>
      </cg:destinationport>
    </cg:node>
    <cg:node name="Invoice">
      <secname:securename xmlns:secname="http://cuc.ucc.ie/xml/secname">
        <secname:domain name="(domain (University (Faculty (Alice Computer))))"/>
        <secname:graph name="(graph cg.nl.po.PurchaseOrder)"/>
        <secname:destination name="(output cg.nl.po.VerifyOp)"/>
        <secname:function name="(function cg.nl.po.InvoiceOp)"/>
      </secname:securename>
      <cg:operandport strictness="strict"/>
      <cg:operatorport operator="cg.nl.po.InvoiceOp"/>
      <cg:destinationport>
        <cg:destination nodename="Verify" portnumber="0"/>
      </cg:destinationport>
    </cg:node>
    <cg:node name="Verify">
      <secname:securename xmlns:secname="http://cuc.ucc.ie/xml/secname">
        <secname:domain name="(domain (University (Faculty (Alice Computer))))"/>
        <secname:graph name="(graph cg.nl.po.PurchaseOrder)"/>
        <secname:input name="(input cg.nl.po.OrderOp)"/>
        <secname:input name="(input cg.nl.po.InvoiceOp)"/>
        <secname:destination name="(output cg.nl.po.PrintOp)"/>
        <secname:function name="(function cg.nl.po.VerifyOp)"/>
      </secname:securename>
      <cg:operandport strictness="strict"/>
      <cg:operatorport operator="cg.nl.po.VerifyOp"/>
      <cg:destinationport>
        <cg:destination nodename="Print" portnumber="0"/>
      </cg:destinationport>
    </cg:node>
    <cg:node name="Print">
      <secname:securename xmlns:secname="http://cuc.ucc.ie/xml/secname">
        <secname:domain name="(domain (University (Faculty (Alice Computer))))"/>
        <secname:graph name="(graph cg.nl.po.PurchaseOrder)"/>
        <secname:input name="(input cg.nl.po.VerifyOp)"/>
        <secname:function name="(function cg.nl.po.PrintOp)"/>
      </secname:securename>
      <cg:operandport strictness="strict"/>
      <cg:operatorport operator="cg.nl.po.PrintOp"/>
      <cg:destinationport/>
    </cg:node>
    <cg:node name="X">
      <cg:operandport strictness="strict"/>
      <cg:operatorport operator="cg.engine.ExitOperator"/>
      <cg:destinationport/>
    </cg:node>
  </cg:graphdef>
</cg:graphdefs>
```

Figure 12: XML definition of the PurchaseOrderGraph