Infrastructure Security Modelling for Utility Computing

Brian Monahan (brian.monahan@hp.com)
Trusted Systems Laboratory
HP Laboratories
Filton Road, Bristol, BS34 8QZ, UK

24th December 2004

Abstract

In this paper we consider how *logic-based object modelling* techniques may be used to help Utility Providers and their customers obtain insight concerning the security characteristics of utility infrastructure and networked systems. We briefly describe two modelling tool prototypes that were built and the underlying technology they used.

Starting from an asset-management view of utility computing infrastructure, we realised that the more interesting questions lie not entirely with the infrastructure itself, but in the way that it is used and deployed within organisations as part of a service-oriented delivery framework. As a result, it is clearly necessary to represent the business process structure and correlate this with the underlying utility infrastructure and the people that interact with these services and systems.

1. Introduction

Logic-based object modelling can be used to help people visualize and explore the security properties of an IT infrastructure configuration. The idea is that the model co-exists in real-time with the infrastructure so that the designers, architects, and utility administrators can, based on the model, explore and see the consequences of their actions. The goal of this work is to provide tools to support the paradigm of *model based security management and assurance*.

We show that these broad, logic-based object modelling techniques (i.e. avoiding representation of detailed behavioural characteristics) can be used to provide a tractable, usable and effective abstraction that models the management of security properties in a utility computing context.

Finally, we discuss and motivate the need for approaches to conducting broad risk assessments of utility computing environments that can help bridge between organisational structures on the one hand and utility systems and IT services on the other.

2. Adaptive Utility Computing

Business is constantly seeking ways and means to increase their return on investment in IT systems. A promising way exploits the Utility Computing model, in which business will contract third-party Utility Providers to provide IT services, typically within a networked data centre environment. To do this economically, Utility Providers will need to provide a computing environment with a high degree of automated support for their IT services and processing.

However, there is a complication. The utility resources that are dynamically allocated by a Utility Provider to their customers will typically need to access and compute over highly-valued data and other IP assets owned by those customers. This potentially represents a considerable risk of exposure and compromise to the significant IP assets of any customer that tries to exploit utility computing in an effective way. Accordingly, customers will need continual assurance that their data and other IP assets are being adequately looked after and protected on their behalf. At the same time, utility providers need to have the means to offer this assurance in a practical and effective manner that could entice, attract and retain customers.

Adaptive utility computing aims to provide computing resources as services on the basis of contractual outsourcing and rental. Such a capability enhances business agility since it means that IT resources can be made dynamically available on a commercial basis to corporate users, thus allowing IT resources to be rapidly and dynamically reallocated as demand varies (i.e. "flexing"). Furthermore, standard commoditised IT infrastructure (i.e. networking interfaces, server systems, and standard OS systems) will be used so that the customer's software configuration can be readily replicated over as many different machines as required, subject to availability. Valued information assets and services can be located at various points in these systems, with a variety of different access paths and dependency links.

For our purposes here, Utility Computing is about creating a flexible infrastructure that is shared between distrusting customers, whilst allowing customers to increase or decrease the amount of resources they are using as their demand varies. We assume a utility provider whose job it is to provide a secure, highly instrumented and trustworthy environment for their customers. Customers will be segmented into virtual infrastructures (farms), and there will be utility management machines responsible for allocating and provisioning resources (i.e. CPU and storage) into and out of these farms in a secure manner.

The basic security property required is that customers should not be able to see each other's data, or even be aware of their presence in terms of networking and service provision (e.g. network isolation). Customers should assume that several defensive measures will be used in the architecture to provide defence-in-depth for the utility itself. In particular, it should be very hard for customers to access or affect the back-end Utility Management servers.

There are a number of techniques that can be used to isolate farms, varying from strong physical separation (air-gapping), use of VLANs and encryption, through to configuration of traditional infrastructure such as firewalls, identity management and access control mechanisms. Customers should assume that the infrastructure will already have been instrumented to the extent that that the provider will be able to gather

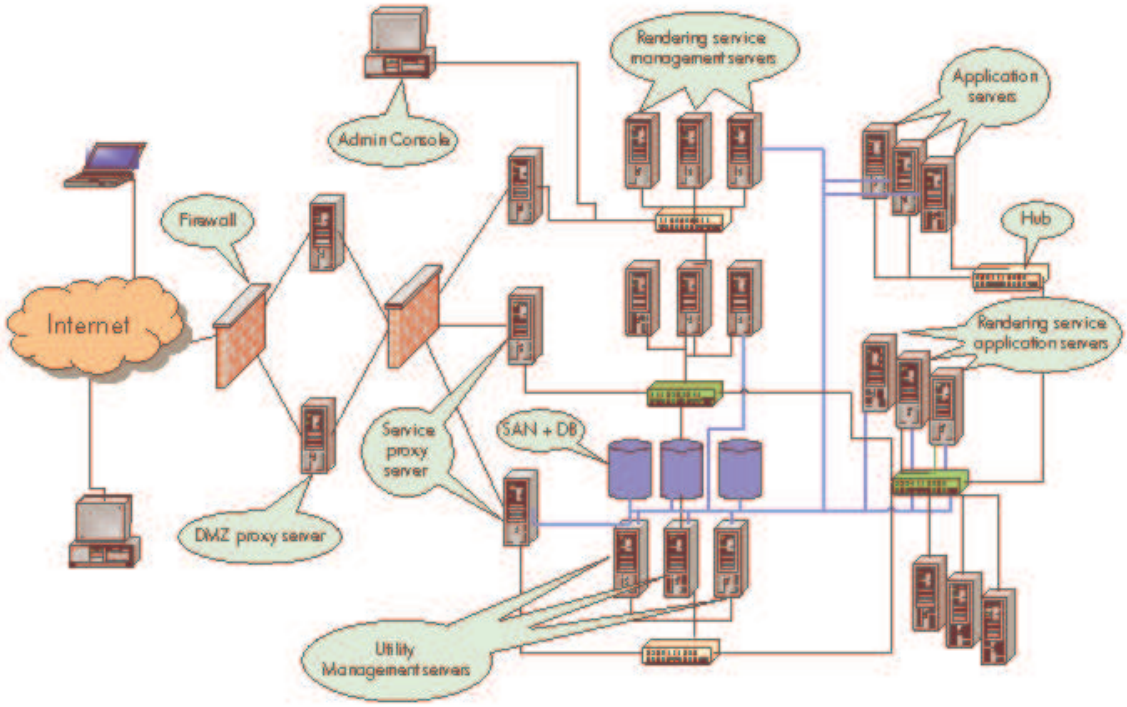

Fig 1. A utility infrastructure consisting of network components running a rendering service.

standard statistics about resource usage, but lacking the ability to eavesdrop in detail upon the customer's activities.

Such flexibility of the IT infrastructure is likely to be attractive to Utility Providers, Service Providers and End Customers alike, because:

- Utility Providers can make their infrastructure available on a dynamic basis to different customers. The main advantage is that Utility Computing helps cut down the costs of provisioning a customer's configuration. This means that it becomes possible to provide service to a wider range of customers.
- Service Providers and End Customers can obtain, under contract, out IT resources from Utility Providers upon demand. They don't need to concern themselves about systems availability or the cost of running and maintaining all of these systems; this is the responsibility of the Utility Provider.

There are several ways in which customers may choose to interact with the resources put at their disposal. Here are two ways:

1. Customers have direct access to the computational resources they have rented and utilise them directly on tasks of their own choosing. The software deployed and the data resources used may be owned and provided by the customer.
2. Customers require a standard commodity service using standard infrastructure and configurations. The customer therefore expects this environment to be rolled out for them by the Utility Provider. The customer's sole IP is likely to reside entirely in the data that is used and generated by running the service provided by the utility.

Typically, there is a specific mechanism provided for the customer to communicate with the utility resources running on his behalf. In each case, the utility resources are deployed according to some structured description.

2.1. Why modelling utility infrastructure helps

There are several concerns for utility providers and their business customers alike. Utility Providers are concerned that their systems are being as fully utilised as possible On the other hand, Service Providers (i.e. Brokers who purely market and sell Utility Services) and End-customers are concerned that they are getting the services that they are being charged for according to contract, that their IP is being kept confidential and that the appropriate computational services are well-managed.

- **Provider asks: What happens to my utility systems if this worm attacks us?**

  Consider the following scenario: a Utility Provider is operating a large set of networked systems in a data centre with resources fully allocated to a number of their business customers. The Utility Provider learns that there are various kinds of worm attacks (e.g. Sasser) are underway. Although patches will shortly be available, there will be some time during which customers could be exposed:

  Some questions are:

  - What is the likely effect/impact of an outbreak within the data centre?
  - In what order should my servers be patched to reduce the impact of these attacks for my business customers?
  - Given best-effort defence, we should accept that some systems will still be vulnerable – at least until the official patches can be applied. In that case, on what basis can I produce a reasonable estimate of the legitimate computing and network activity that I should charge my unpatched customers for? What is the trade off to be made here?
  - Does this attack compromise customer data separation? If so, what could be done about it?

- **Customer asks: How is my confidential data protected?**

  Consider the following scenario: a corporate business customer outsource's an important part of their IT operations to a Utility Provider, subject to an appropriate Service-Level Agreement and contract. However, to run the service effectively, the customer will need to provide direct access to significant IP such as confidential commercial data. Such information could certainly be useful to a competitor.

  Some questions are:

  - What is the risk of exposure of my valued IP to undesirable/unauthorised access?
  - Can I organise my resources and their defences better to mitigate my risks of data exposure, whilst still continuing to operate effectively?
  - Why can't others in the Utility see my data? How does the Utility's configuration prevent unauthorised access? Can I access anyone else's VLAN? If I could see them, perhaps they could see me (e.g. network isolation)?
  - How "well" are my services performing under this Utility Provider? Is performance meeting my Service-Level Agreements and Service-Level Objectives?

The answer to several of these questions involves constructing some kind of model of the utility security system that is accessible to customer and provider alike. The

remainder of this report is concerned with suggesting some reasonable and effective ways in which this might be done in practice.

3. Building models of utility infrastructure for security

As we have seen above, the basic problem is to represent the security aspects of a deployed utility, in a form permitting exploration of interesting and relevant "what-if" consequences.

An important part of the value proposition for Utility Computing is that the utility systems architectures can be built up from standardised, commodity third-party components for the networking, the server hardware and the software stack. This means that the overall system offers a uniform, standardised computing environment to each of its customers that is not dependent in detail upon which particular resources are allocated to particular customers. This has the benefit from the Utility Providers point of view that hardware and software systems can be more readily replaced and swapped around in the event of component or systems failure.

This has a further implication for the kind of security modelling that can be effectively used in practice. Because third-party components are used, this effectively restricts the type of information, properties and characteristics that the model has available about any particular component system or device. Broadly, the security model has to be based as much as possible upon the infrastructure's configuration information.

3.1. Modelling utility security requires effective abstractions

Modelling the utility in an effective manner could be attempted at many different levels. For example, each of the networking devices, the compute servers and even the software itself can be thought of in terms of detailed systems activities and processes. However, as explained above, the utility is built out of standardised, third-party components for which it is unreasonable to expect there to be sufficiently detailed, readily available descriptions of behaviour. Accordingly, we have to instead make good use of whatever information about these components that is available, such as the systems configuration information, for instance.

Even if we did have sufficiently detailed behavioural descriptions of all the hardware and software components, the upshot of the current research into systems verification ([HR04]) implies it is an intractable problem to systematically derive and extract useful consequences from such detailed, complex information. Thus, some form of abstraction would need to be applied in any case, if one wants to be able to gain any kind of effective prediction concerning the security of utility configurations.

There are some other factors working in our favour as well. The properties that we wish to deal with are security characteristics that, in any case, one may reasonably expect to be extractable from systems configuration information (e.g. network connectivity, access control permissions, and firewall rules). Fortunately, the kind of models we are interested in here involve viewing the utility architecture as a kind of graph structure which can be extracted from such information. This structure also conveniently permits us to perform various reachability path queries, allowing us to examine the security consequences of modelled utility configurations (e.g. impact analysis).

3.2. Semantic network models

The approach we shall take here exploits a number of ideas familiar from semantic networks, object-modelling (see [UML, CIM]) and more generally, knowledge

representation studies in philosophy and AI (see [Sowa1, Sowa2]). The security modelling approach starts from the idea of being able to rigorously *survey* hardware/software infrastructure and their configuration attributes. This *asset-management* philosophy has already been explored with some success (see [HG02, GHR03, DISCEX01, KYBS99]), and is thus potentially applicable to utility computing.

We naturally represent particular entities such as hardware servers and software applications by *objects* having a certain *attribute* structure that is specified by a *class* structure. For example:

```
def_class(server, [device, computer]
   [ location / string,
     role     / string,
     model    / string,
     os       / OS ])
```

specifies a class called `server` that is a sub-class of both `device` and `computer` with several simple attributes such as `location` (of type `string`) and operating system (`os` of type `OS`).

The systems entities that we are attempting to capture and describe are naturally multi-faceted and so we provide a class system that also supports multiple inheritance. Note that supporting multiple inheritance of classes means that the ancestor classes of some class must have attributes that are mutually consistent in terms of their types.

Values are defined in terms of the particular classes they instantiate and the attributes that they are given. For example:

```
defn (neptune,
      server(role                / "Management server",
             os                  / rh_linux,
             remote_admin_access / false,
             tty                 / p27,
             location            / "main m/c room"
             )
)
```

defines a particular instance of the class `server`, called `neptune`. Note how this instance doesn't possess the particular attribute "`model`" mentioned in the class definition for class `server`. Additionally, the instance also included a couple of extra attributes (i.e. `remote_admin_access` and `tty`).

All of this is fine, as we may add, delete or modify attribute information at some later stage to reflect our current state of knowledge. In modelling "live" systems, we are inherently dealing with incomplete and imperfect information that are continually subject to change and revision. Nothing about the configuration of the utility is assumed known with complete finality.

We have therefore found it useful to be *tolerant* of partial and incomplete information. In particular, we do not require that attributes are always defined for every instance of a given class. However, once the attribute value is defined, then we expect it to match the associated type constraint.

In fact, we may define instances and classes in any order; class definitions can follow after instance definitions if necessary. This implies that instances may need to be (re)validated upon class (re)definition.

In principle, classes may also have *logical invariants* associated with them. However, these are only applied and checked upon update of the relevant attributes for each

instance. This is because invariants are only meaningful and checkable if all the relevant attributes are defined. This gives a more permissive regime accommodating our understanding that knowledge about the utility configuration is typically incomplete.

3.2.1. Associations

We need more than pure objects to express all the characteristics that we are interested in. In particular, we are interested in various graph-theoretical concepts of linkage and connection that naturally arise when modelling systems (e.g. network connectivity between devices, module and library use relationships). To this end, we introduce a structured form of *binary association* (or *link*). These are structured entities that explicitly join or connect two objects (the *source* and *target*). We allow associations to be either directed or undirected.

Associations are structured in the same ways that objects are in the sense that they have a class structure (called *link-classes*) and also may have attributes of their own. Thus, we distinguish between attributes and associations – which are often treated in the same way in other modelling systems. This means we can easily formulate properties qualifying not only objects but also the associations themselves.

The advantage of using link-classes to qualify associations is that we can constrain the kinds of object that can be used as sources and targets. For example, we make use of this to ensure that associations representing network connectivity can only be attached to computer systems and not other kinds of entity, such as some kind of software component. Furthermore, by using attributes on the links themselves, we can assert that an association represents a communications path between two systems using particular protocols e.g. `https`, `tcp-ip`. Another application of using attributes on associations is in modelling VLAN links.

The use of attributes on both objects and associations is illustrated in Fig 2 below.

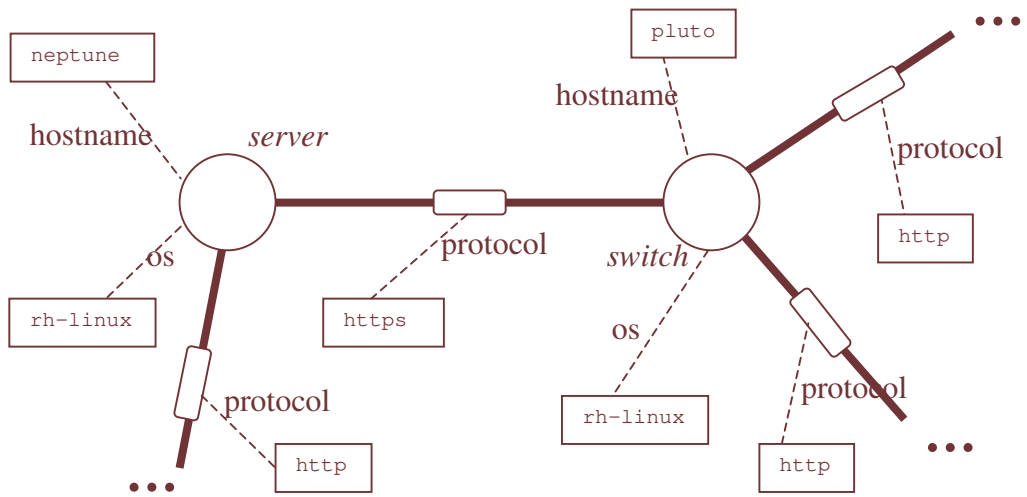

Fig 2: Attributed objects and attributed associations (thick lines)

In Fig 2, the dotted lines indicate an attribute for an object or association, and named by an identifier. The square box attached to the dotted line represents an atomic value (i.e string or number). The thick line indicates an association or dependency of some kind between objects; clearly, there may be more than one association between two objects.

3.3. Semantics of routing & path making

As we have described, the utility architecture is modelled in terms of attributed objects linked together by structured, attributed associations. This means that the kinds of connection between objects are not just simple links but can be quite complex in their own right.

There are two kinds of queries that we are going to need to use:

- *Node queries* that select particular sets of nodes.
- *Path queries* that show that two sets of nodes are linked together by paths satisfying certain constraints. This kind of query naturally involves reachability over the graph of associations. As such, these queries have some resemblance to temporal model-checking (see [HR04, ModChk]).

As a result of this expressiveness of linkage, we can impose *semantic constraints* on the routing connectivity between different classes of nodes, for example. This allows particular classes of node, such as firewalls and switches, to have some specific connectivity properties that can be dependant upon:

- Attribute information associated within the particular node.
- Attributes within the incident associations themselves.
- Other specific path information (e.g. overall source and destination).

These special connectivity properties are defined by *connection predicates* for particular classes and link-classes.

For example, each router instance will typically have a "rules" attribute whose value could define the permitted VLAN connections. The linkages permitted via the router instance then depend upon these rules and the attributes of the respective associations and their link-classes. This dependency will be determined by a connection predicate defined for the class of routers. We illustrate this in Fig 3 below.

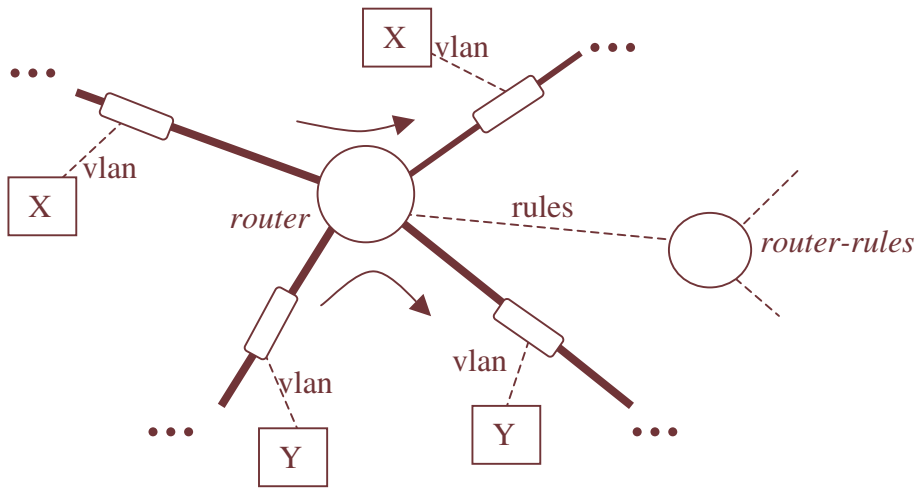

Fig 3: Making a path by connecting dependant associations at nodes.

Routing and path formation can in general depend upon more than the local attributes of the links associated to the node. For example, routing through a firewall will typically depend upon the source and destination IP addresses of a path.

3.4. Using reachability path queries over graph models

As utility designers, providers and operators, we are typically interested in knowing:

- Is it possible for the configuration of this part of the utility to have a certain kind of impact on this other part of the utility?
- Given that some part of the utility has a given property, what is the likely impact this has elsewhere on the utility?
- Given that a certain particular situation has arisen, what utility configurations could have allowed this to happen?

The kind of *reasoning* about the utility needed to answer all of the above critically depends upon being able to explore the model and find paths having certain characteristics that link certain sets of nodes (i.e. mapping the *dependencies*). Paths are represented as (non-repeating) sequences of links, where the nodes and links satisfy certain properties. In simple cases, such path-finding typically involves computing reachability in terms of transitive closure of the graph. However, because of the potentially complex nature of the associations used, we need instead to adopt a lazy computation strategy that tries to minimise the number of unnecessary paths or linkages computed.

A query evaluation framework was developed for the purposes of conducting demo's and small-scale experimentation. For the time being, we only informally illustrate the kind of queries that are supported in our demo prototypes (see below) via some examples given below. Note that the query framework provides only simple means to name and index the results of queries denoting sets of nodes, edges or even paths.

- `ask( servers )`

  This query determines the current set of all servers.

- `ask( servers and [ os/ rh_linux, version / 9.7 ] )`

  This query determines the set of servers with attribute `os` set to "`rh_linux`" and attribute `version` set to 9.7.

- ```
  ask( server and
       reaches(file_server, network and [protocol/https]) )
  ```

  This query determines those servers that can reach/access those file_servers via edges of link-class `network` having attribute `protocol` set to `https`.

- `reach( n1 , n2 )`

  This query determines a lazy enumeration of the set of paths from a node labelled `n1` to node labelled `n2` (where there is an additional semantic constraint built-in). Typically, there may be several paths satisfying the semantic constraint but usually only the first is of interest as a witness of existence.

- ```
  ask_multi_path( customer_sys,
        [ [ svc_portal,                      network_http]
        , [ server and contains(render_app), network_http]
        , [ vuln_utility_servers,            network_http] ))
  ```

  This query determines a lazy enumeration of *composite* paths starting from nodes belonging to `customer_sys` and which use links belonging to `network_http` to reach several intermediate node sets (e.g. `svc_portal`) and which finally reaches the set `vuln_utility_servers`.

From the above examples, it is clear that our queries will critically depend upon attribute values of both standard objects and associations between objects. Future extensions include defining and implementing a query and data description language based upon the framework developed so far.

3.4.1. Relationship to conventional database technology

Traditional database oriented knowledge representation, based upon non-recursive relational algebra (as typified by SQL) doesn't adequately cope with the richer path-type queries, such as reachability and transitive closure (see [Ullman88], page 145). Thus, our query language has to strictly extend the range of queries that are typically supported by a conventional relational database.

By adding a form of *recursive* query, we provide a strictly more expressive query language than provided by any variant of SQL, the Standard Query Language. This result has been well-known since the 1980's, but not much exploited except perhaps in AI-style reasoning applications (see [HR04, ModChk]). In practice, such queries would have to be executed using ad-hoc "stored routines/procedures" that are external to the database system itself.

3.5. The need for standardised systems descriptions

One of the common criticisms about model driven approaches to systems architecture is that high-level models can very quickly lose touch with the actual system after implementation and deployment. Typically, models aren't kept up-to-date and do not provide an accurate reflection of the system dynamically, thus quickly losing any authority it may have once had to speak about system properties.

In some sense, our work above also suffers from this as it solely discusses representations and techniques for reasoning about models of systems infrastructure, and merely assumes that there is some accurate correspondence to the current configuration.

However, the situation is by no means lost. There already exist mature, well-developed tools and standards for reporting systems configurations (e.g. HP OpenView and SNMP). More recently, some promising standards and technologies (e.g. CIM and SmartFrog) are emerging that could help provide the semantically rich device and infrastructure descriptions that are required. Broadly, this means that we can define a collection of plug-ins that allow systems infrastructure descriptions to be supplied in a variety of formats and then used to build models for subsequent processing and analysis (see Fig 4).

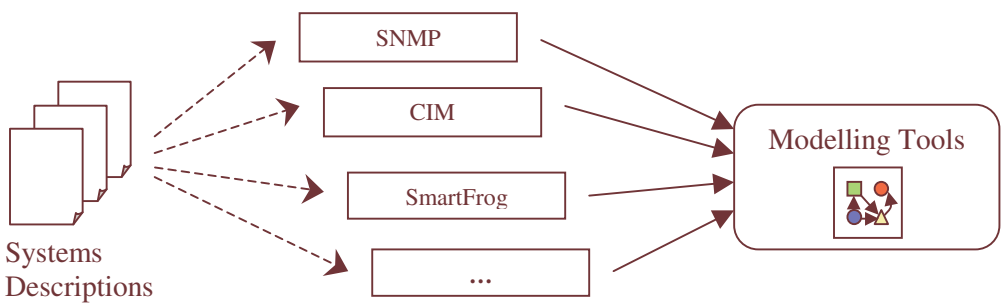

Fig 4: Using Systems Infrastructure Descriptions to provide accurate data for modelling utility systems.

In the next two sections, we describe a couple of particular means by which accurate systems infrastructure configuration descriptions may be extracted.

3.5.1. Common Information Model

One example of such a standard is the systems management technology known as the Common Information Model, as standardised by the Distributed Management Task Force (see [DMTF, CIM]). CIM provides an object-oriented data model of an implementation-neutral schema for describing overall management information in a network/enterprise environment, whilst allowing for vendor extensions. This is typically used to describe individual device configurations and related management information. This enables vendors to define in an implementation-neutral manner the semantically rich management information that can be exchanged between network-connected systems.

The CIM standard is composed of a Specification and a Schema. The Schema provides structure for the model descriptions, while the Specification defines the underlying details and interpretation. The CIM standard includes an XML representation for data exchange and provides a mapping of CIM Messages onto HTTP allowing implementations of CIM to interoperate in an open, standardized manner. CIM uses a set of concepts similar to those described earlier. In particular, CIM makes use of associations to represent the structured links we use between instances of objects. This commonality makes CIM particularly suitable as a systems descriptive framework that is compatible with our query framework involving path properties.

The Schema provides a wide range of classes – from storage, to providing support for modelling the Java™ 2 Enterprise Edition (J2EE) environment. The Schema includes the concept of management profiles, support for managing security principals and describing their authentication policy and privileges, manages IPsec policy and resulting security associations, and features modelling of management infrastructure for discovery.

The DMTF is currently working on extensions to all of its management standards, including CIM, to take greater account of Utility Computing and related industry initiatives.

3.5.2. SmartFrog

Another important class of examples are automated distributed deployment technologies, such as SmartFrog (see [SF, SF-RefMan]). This is a technology for describing distributed systems as networks of cooperating software components, for the purpose of initiating them and subsequently managing their activity.

Systems[1] deployed using SmartFrog typically have multiple software components running across a network of computing resources, where the components must work together to deliver the functionality of the system as a whole. It's critical that the right components are running on the correct computers, that the components are correctly configured, and that they are correctly combined together into the complete system. This requirement recurs across many services and applications that run on all kinds of computing infrastructure, naturally including utility computing systems.

A concrete example might be a three-tier web application, which will often consist of a database server, application logic middleware, web server software, firewalls and load-

[1] The material in this section is based upon the SmartFrog FAQ.

balancers. All of these can be thought of as components that need to work together to deliver the complete web-service. Each component must be installed on an appropriate resource and correctly configured. Components must be started in a certain sequence, and linked together into the complete system.

With SmartFrog, system configuration details are captured in formal system descriptions documents written in the SmartFrog description language. These documents are interpreted by the SmartFrog distributed runtime engine in order to install the required software components, to configure them, and to start the complete software system according to the details of the description.

4. Prototypes

We built two prototype artefacts to illustrate and demonstrate in an effective way our modelling approach in the context of utility computing. All prototypes were constructed using non-proprietary, open-source, multi-platform technologies.

4.1. Enterprise Security Modelling Tool

The first prototype, the Enterprise Security Modelling Tool, developed the object-oriented deductive database approach, in which:

- Models of utility computing infrastructure were constructed (by hand) in the manner described above. As implied earlier, in neither of our prototypes did we consider the issue of where these descriptions came from. We did not spend any effort here in developing automated mechanisms for capturing this information.
- Certain kinds of graph reachability query was ran against the model and the results obtained were shown to combine together to help investigate high level accessibility questions, as motivated by the two scenarios mentioned earlier in Section 3.

This prototype was assembled and implemented fairly rapidly in Prolog (see [SWI, AofP]) and demonstrated as a part of an international press event held at HP Labs in late April 2004. It was designed mostly to show that some kind of effective model could be constructed and then queried in a manner useful to utility customers and providers – this much was achieved.

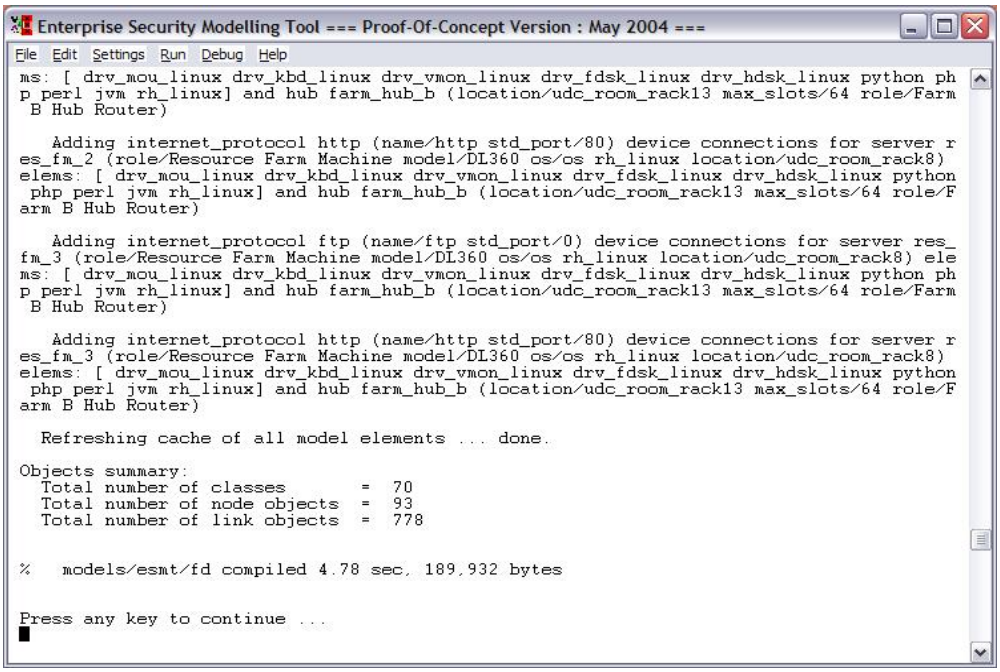

Fig 5: Screenshot of the Enterprise Security Modelling Tool

What we learnt from building this prototype was that further thought is needed about mechanisms for capturing accurate and relevant utility configuration information in a live manner. Perhaps as telling, it also became abundantly clear that relevant security information is not easily extractable from basic configuration information on its own – the organisational context is necessary to help understand what the interesting queries are that utility providers and their customers might care about.

4.2. Labyrinth

Our second prototype, called Labyrinth, was developed to explicitly show that it was possible to provide an accessible graphical user interface to simplify the interaction with the modelling tools. A further driver for this was to be able to present this work as part of our overall research programme at an influential external event, the EU's IST 2004 conference held in The Hague, NL, 15th-17th November, 2004. The conference programme largely addressed policy issues related to research and the long-term perspectives and trends affecting the EU IST research agenda.

The strategy we took was to focus on how graphical information could be extracted for presentation from systems models and how to illustrate the results of queries.

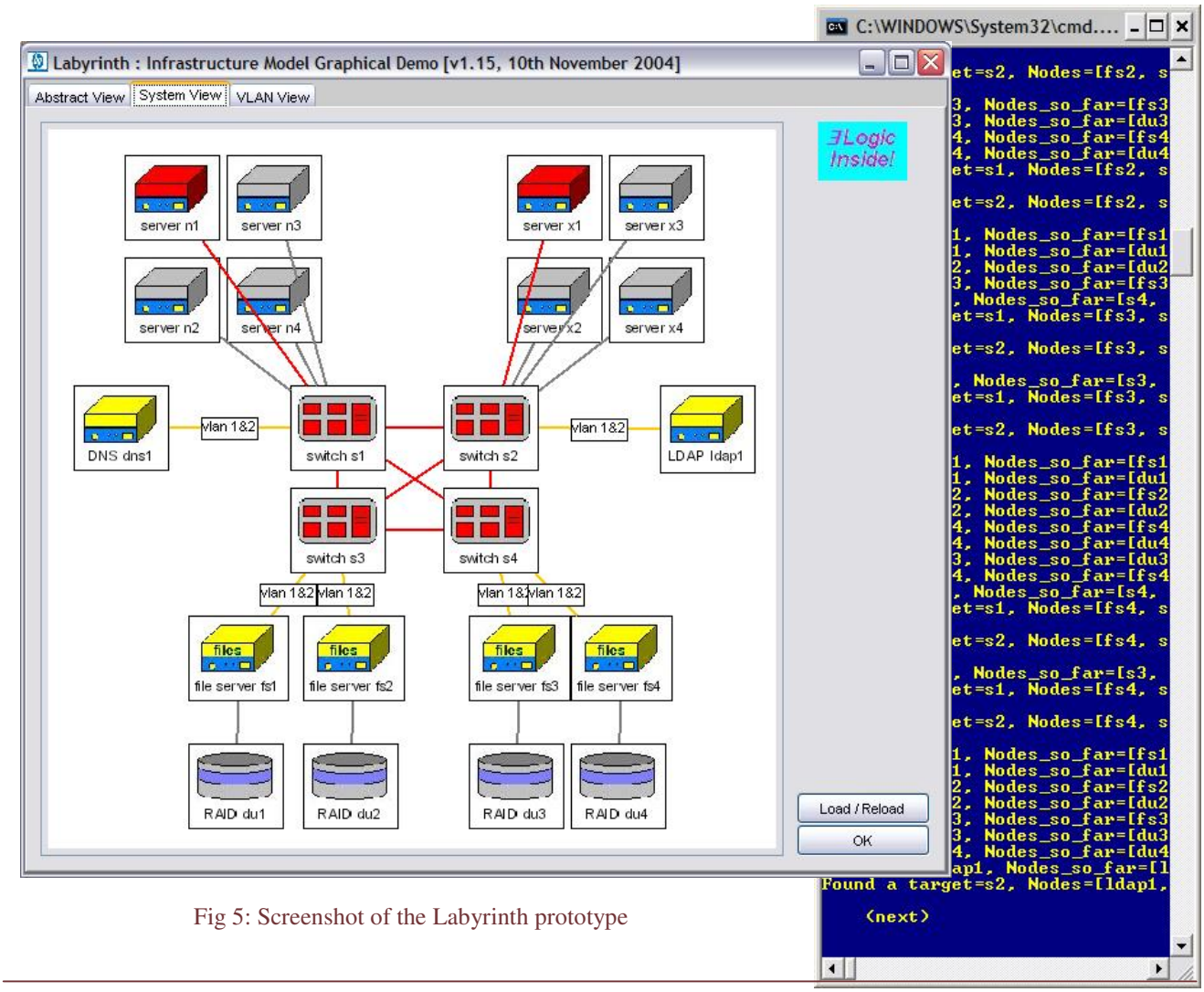

Fig 5: Screenshot of the Labyrinth prototype

Given these objectives, it was also important to develop architecture for this system that could be *compatible* with our earlier prototype ESMT. In particular, there had to be a practical way to include existing and future work on the reasoning engine without being *forced* to re-implement everything again from scratch. As it happens, the Labyrinth reasoning engine was simpler than that used for ESMT, as we needed to focus on producing graphical output for display. However, we don't anticipate major difficulties in upgrading the Labyrinth reasoning engine to incorporate the capabilities developed in ESMT. The architecture we adopted is diagrammed in Fig 6:

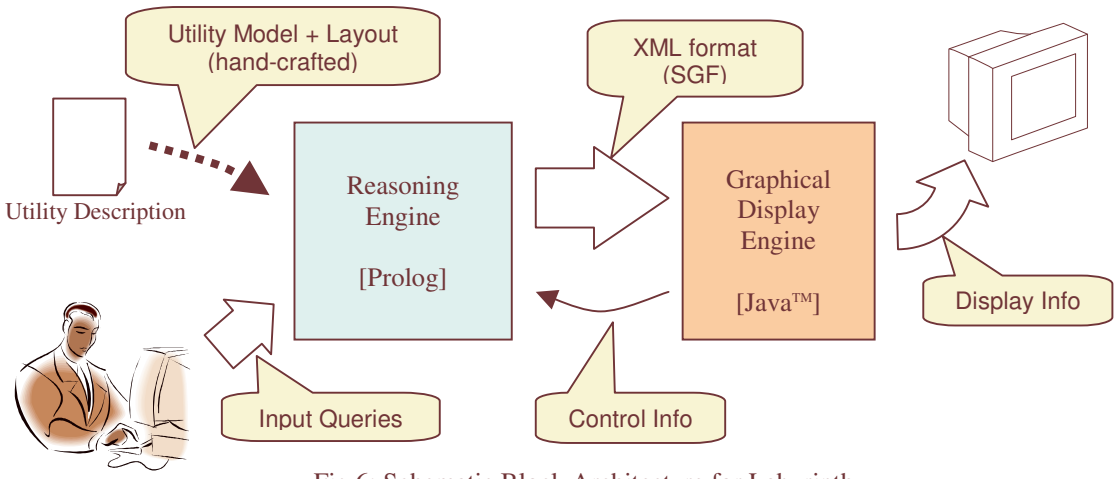

Fig 6: Schematic Block Architecture for Labyrinth

The bulk of the effort was spent in developing the Graphical Display Engine in Java™ – this was substantially based upon the open-source JGraph graphical visualisation library (see [JGraph]). We made extensive use of the popular Eclipse development environment.

What we learnt from developing Labyrinth was firstly that Prolog can be very effectively coupled with other systems as a subordinate process. There is of course a need to coordinate activities between all the systems involved – but this turned out to be straightforward to implement.

Secondly, as one would probably have anticipated, there are no "silver bullet" solutions to the task of laying out graphical information to highlight important aspects of a particular situation. In some sense, this should be expected, as complex sets of information will typically have more than one aspect needing to be highlighted – there is often no clear priority. Graph drawing is well known to be a hard problem to solve and JGraph itself implements a number of well-known algorithms (e.g. Simulated Annealing, Moen's algorithm, Sugyjama, etc.) that have been developed to do this. However, the high bandwidth provided by graphical display's for showing complex information is something of a mixed blessing. It is hard to avoid, on the one hand, confusion due to showing too much and over-simplification by not showing enough, on the other.

In the end, we manually laid out the diagrams and avoided, for the time being, any "closed" automation of layout. This reduces the frustration of trying to indirectly persuade systems to layout data according to one's aesthetic desires, but at the expense of having to do so by hand. It would be interesting to combine automated "layout policy" with some degree of manual placement and layout editing.

5. Risk Analysis – or asking the right questions

As already discussed, Utility Computing provides technology for customers to outsource and "rent" computing resources from Utility Providers. These Utility Providers offer a managed computing environment into which customers can deploy their own services and software environments, built upon standard, already instrumented, components and environments. Infrastructure models have been developed and investigated here that can help both utility customers and utility providers explore and query the security consequences of configurations. In general, utility security will require collaborative cooperation between utility providers and their customers.

However, as implied earlier, simply being able to ask all sorts of low level queries about infrastructure and its security configuration doesn't help with knowing what the important, business related security questions are that *need* to be asked (i.e. security policy). There is also the important question of translating high-level concerns about security protection and policy into lower-level queries about infrastructure.

In short, supplying effective security involves knowing what needs to be defended, whilst at the same time enabling business utility customers to serve their end-customers, to continuously optimise operations, and maintain their competitive edge.

This broader picture involves not only understanding the configuration of the infrastructure systems but additionally understanding the needs that they are designed to serve. This involves to some extent understanding and mapping out the organizational context and the business processes involved. Such knowledge helps both the business customer and their utility providers to see better what the risks are and thus making informed decisions concerning how best to defend their assets with the resources available.

Given this analysis, we are broadening our research goals to encompass more of this "risk management" point of view which takes more of the organisational structure into account. Traditionally, risk assessment has meant calculating "impact × probability" in some meaningful way. Such a calculation is difficult to do meaningfully unless the impacts of compromise and loss of service functionality have been understood in business terms.

A key part of this risk assessment process involves the business determining its "risk appetite". This is a risk profile that identifies classes of risk and at what level risk is deemed acceptable and, consequently, what level it is deemed unacceptable. Once risk has been identified and assessed, appropriate controls and process mechanism can then be put in place to mitigate the overall risk by reducing the probability of incidents and even their impact. Of course, these controls themselves will have some management overhead and a need for appropriate configuration etc.

From a business point of view, security issues are also increasingly linked with corporate IT governance. Legislation such as the HIPAA and Sarbanes-Oxley acts in the US now make corporate management directly accountable for their organisational practices, including financial integrity and security. The need for regulatory compliance is now forcing companies on a global scale to develop and adopt explicit security policies and mechanisms. Also, at a systems level, there is increasingly a parallel to be seen between policy and management for security in business-critical systems and policy and management for safety-critical systems. Approaches for actively managing

risk associated with safety concerns may therefore be relevant in the context of security (see [Leveson, PRA]).

The UK MoD and QinetiQ have jointly developed their Domain-Based Security (DBSy) risk methodology to map out and assess information security requirements at a business relevant level (see [DBSy1, DBSy2, DBSy3]). A Business Communications Model is first mapped out and developed which establishes the business need for communications between business entities. A second Infrastructure Architecture Model is then developed that captures at a high-level the infrastructure and networking requirements in alignment with the business need established earlier. The risk analysis process then makes use of *Compromise Path Analysis* to establish the degree of risk due to potential sources of compromise and information leakage. This prioritisation means that defences can then be more strategically placed to mitigate the most pressing sources of compromise.

Techniques such as DBSy offer an interesting, promising data point but will need further research to take it into a commercial context. For example, the various DBSy models do not easily address the *stewardship* issues that naturally arise in the context of Utility Computing, where customers place their IT capital in the hands of one or more trusted Utility Providers. Today, commercial organisations view the Internet as primarily a business tool via which business transactions are routinely performed. The Internet is also a source of potential threats, which therefore has to be balanced against modern business needs. The risk analysis models need to more explicitly incorporate strong identity concepts (e.g. Authorisation-Authentication-Audit, Virtual Private Networks) to appropriately assign responsibility and also capability. Finally, there is a clear need to explicitly identify systems management roles and associated controls as a part of the infrastructure mapping – and this certainly lies at the interface between systems and business organisation.

Conclusions

Our initial objectives were to investigate the building of simplified semantic models that could examine security consequences of managed infrastructures. We developed two prototype systems demonstrating that such models could be effectively implemented and deployed. There is a clear need for more automation to capture and report on systems infrastructure configuration details. To a large extent, such automation is already in progress as a part of the development of existing and emerging technologies and standards, such as CIM and SmartFrog.

By far the most important realisation is that it is ***necessary*** (not merely 'interesting') to have knowledge of the business objectives and their risks before being able to assess if security and IT governance objectives have been adequately met by a particular systems configuration. Inevitably, there is a mismatch in translating high-level security policy requirements into queries against lower-level systems configurations and in translating the responses back again. It is an interesting open question to see how far this gap can be bridged. As a result, it is clearly necessary to represent the business process structure and correlate this with the underlying utility infrastructure and the people that interact with these services and systems.

Acknowledgements

I thank Simon Shiu, Adrian Baldwin, Yolanta Beres, Chris I Dalton, Nicholas Murison, David Plaquin, Rich Smith and Frederic Gittler for their assistance and many helpful comments during this research.

References

[AofP] L. Sterling, and E. Shapiro, *The Art of Prolog*, 2nd Ed, MIT Press, 1994

[CIM] http://www.dmtf.org/standards/cim/

[DBSy1] Domain Based Security White Paper, QinetiQ 2004
http://www.qinetiq.com/home/core_skills/knowledge_information_and_systems/trusted_information_management/white_paper_index.Par.0004.File.pdf

[DBSy2] C.L. Robinson and K.J.Hughes, *Managing Infosec Risk in Complex Projects*, 4th Annual Systems Engineering for Defence Conference, RMCS Shrivenham, 15-16th February 2001

[DBSy3] C. L. Robinson, *Security Requirements Models to Support the Accreditation Process*, 2nd Annual Sunningdale Accreditor's Conference, 10th – 11th September 2001

[DISCEX01] J.Burns, A.Cheng, P.Gurung, S.Rajagopalan, P.Rao, D.Rosenbluth, A.V.Surendran, D.M.Martin Jr. *Automatic Management of Network Security Policy*, DISCEX II'01 -- DARPA Information Survivability Conference and Exposition, IEEE Press, June 2001.

[DMTF] http://www.dmtf.org

[GHR03] J.D.Guttman, A.Herzog, J.D.Ramsdell, *Information Flow in Operating Systems: Eager Formal Methods*, Workshop on Issues in the Theory of Security (WITS'03), April, 2003.

[HG02] A.Herzog, J.D.Guttman, *Eager Formal Methods for Security Management*, Proc. of VERIFY'02, 2002

[HR04] M.Huth, and M.Ryan, *Logic in Computer Science*, 2nd Ed, CUP, 2004

[JGraph] http://www.jgraph.com/

[KYBS99] A.V.Konstantinou, Y.Yemini, S.Bhatt, S.Rajagopalan, *Managing Security in Dynamic Networks*, USENIX -- 13th Systems Administration Conference -- LISA '99, November 1999.

[Leveson] Nancy G. Leveson, *Safeware: System Safety and Computers*, Addison Wesley, 1995.

[ModChk] E.M.Clarke, O.Grumberg, and D.A. Peled, *Model Checking*, MIT Press, 1999.

[PRA] H.Kumamoto, E.J.Henley, Probabilistic Risk Assessment and Management for Engineers and Scientists, 2nd Ed, IEEE Press, 1996

[SF] http://www.smartfrog.org/

[SF-RefMan] http://www.smartfrog.org/papers/sfReference.pdf

[Sowa1] J.F.Sowa, *Conceptual Structures*, Addison Wesley, 1984

[Sowa2] J.F.Sowa, *Knowledge Representation*, Brooks/Cole, 2000

[SWI] http://www.swi-prolog.org/

[Ullman88] J.D.Ullman, *Principles of Database and Knowledge-Base Systems*, Vol.1, Computer Science Press, 1988

[UML] http://www.uml.org/