UNIVERSITY OF
CAMBRIDGE
Computer Laboratory

Second Year Report and Dissertation Schedule

Evangelos Kotsovinos
Trinity Hall
ek247@cl.cam.ac.uk

July 2003

Contents

1 Introduction

In this document I describe the work that I have undertaken during my second year, and provide a work plan and dissertation schedule for my final year towards my PhD in the Systems Research Group, Computer Laboratory, University of Cambridge.

As the price of raw computing resources keeps falling, as a result of Moore's Law, it becomes apparent that it is the *location* of those resources that is becoming increasingly significant. Users that connect to remote servers can experience *communication latencies* of more than two seconds, while having to pay substantial amounts for long-haul network traffic.

Moreover, there is an increasing need for infrastructures that can support the deployment of large-scale or global-scale services, such as computational grids [13], scientific simulations, mobile agent systems [11] or PlanetLab [23]. Using existing architectures, it is extremely expensive to deploy such services, as the cost of entry is very high. Leasing dedicated machines around the world – the machines need to be dedicated if the service is untrusted – is very costly. Setting them up properly, and moving and installing all necessary applications there, is prohibitively complicated.

To address those needs, we proposed the *XenoServer Open Platform*, a global-scale public computing infrastructure, aiming to support the safe deployment of utrusted code in exchange for money.

My work in the second year towards my PhD was aimed at refining the design of the XenoServer Open Platform, while proceeding with the implementation. I have also worked on two new research subjects, namely Distributed Trust Management (XenoTrust) and Role-Based Resource Management. As most of my work on the XenoServer Open Platform has been published in papers during the year, I will devote the biggest part of this report to Role-Based Resource Management, which has not been published yet.

2 The XenoServer Open Platform

A description of the initial version of the XenoServer Open Platform, similar to the version suggested in my first year report and thesis proposal [15, 16], is provided in [17]. There have been some design changes, mainly in decoupling registration, authentication and charging from resource discovery, with the latter being moved out of XenoCorp's responsibility.

As I have written a substantial amount of text during the year in papers

describing the design of the XenoServer Open Platform and XenoTrust, I will only include a brief summary in this document. More information can be found in the papers cited.

In [10], we provide a concrete version of the design of the XenoServer Open Platform, and describe the mechanisms that can be used to perform efficient resource discovery in this global-scale setting. The Xen hypervisor [1] is now used to perform isolation between the execution environments, instead of VServers and User-Mode Linux.

The components of our platform are the following. XenoServers host and safely execute untrusted client code. XenoCorp is the authoritative entity and handles registration, authentication and charging. XeST is a unified globally-accessible file system, designed to allow clients access to their files from any XenoServer. The XenoServer Information Service (XIS) is being used by XenoServers to advertise themelves to clients, and by clients to lookup for XenoServers. Finally, XenoSearch is used to carry out more complex searches, such as "look for a XenoServer that is close to a given set of clients".

An analogy to the real world could be that of VISA; clients buy resources from XenoServers, who are the resource merchants. XenoCorp is playing the role of VISA, mediating between merchants and clients. XIS is similar to the yellow pages, where clients can look for XenoServers, and XenoSearch is a more advanced searching tool, like on-line search engines.

This version of the platform has been implemented using Java RMI. A visually appealing description of the code deployment path is given in [18].

2.1 Resource Discovery

To name entities in the XenoServer Open Platform, we use XOPIDs. The format of those is:

```
<XenoCorp:Kind:ID>
```

XOPIDs are hierarchical, and consist of three parts; the *XenoCorp* under whose authority the entity exists or operates, the *kind* of the entity – such as "resource", "client", "XenoServer" etc. – and an *identifier* for the entity – for instance, a sequence number. This hierarchical approach allows each XenoCorp to use a separate name space.

Each XenoServer reads its available resources from a configuration file at

start-up time, and maintains a list of those. When an execution environment is launched on the XenoServer, it is associated with a list of the resources that are allocated to it. Thus, the XenoServer is always aware of total resource availability, resource usage and exactly how resources are apportioned between the environments.

```
          Owner  defender-0.xeno.cl.cam.ac.uk:1:1
  Administrator  defender-0.xeno.cl.cam.ac.uk:1:1
     IP Address  128.232.35.170
       Hostname  atikatak-0.xeno.cl.cam.ac.uk
           City  Cambridge
           Area  CambridgeShire
        Country  UK
      Resources  CPU, defender.cl.cam.ac.uk:6:0, defender.cl.cam.ac.uk:7:0, 110, 2
                 CPU-R, defender.cl.cam.ac.uk:6:0, defender.cl.cam.ac.uk:7:2, 800, 0.05
                 MEM, defender.cl.cam.ac.uk:6:9900, defender.cl.cam.ac.uk:7:100, 3936, 3
                 MEM-R, defender.cl.cam.ac.uk:6:9900, defender.cl.cam.ac.uk:7:101, 3936, 4
                 MEM-R-T, defender.cl.cam.ac.uk:6:9900, defender.cl.cam.ac.uk:7:102, 32, 0.1
                 NET, defender.cl.cam.ac.uk:6:9902, defender.cl.cam.ac.uk:7:200, 100, 8
                 NET-R-1, defender.cl.cam.ac.uk:06:9902, defender.cl.cam.ac.uk:7:201, 60000, 1
                 NET-R-2, defender.cl.cam.ac.uk:6:9902, defender.cl.cam.ac.uk:7:202, 80, 5
                 IPv4-FULL, defender.cl.cam.ac.uk:6:9903, defender.cl.cam.ac.uk:7:300, 9, 20
                 IPv4-PORT, defender.cl.cam.ac.uk:6:9903, defender.cl.cam.ac.uk:7:301, 200000, 0.1
                 STG, defender.cl.cam.ac.uk:6:9901, defender.cl.cam.ac.uk:7:400, 100, 1
                 STG-R, defender.cl.cam.ac.uk:6:9901, defender.cl.cam.ac.uk:7:401, 60000, 0.002
                 STG-R-T, defender.cl.cam.ac.uk:6:9901, defender.cl.cam.ac.uk:7:402, 60000, 0.005
                 STG-R-B, defender.cl.cam.ac.uk:6:9901, defender.cl.cam.ac.uk:7:411, 20000, 0.005
```

Figure 1: XenoServer advertisement format

As described in [10], a XenoServer will periodically submit a server advertisement – similar to the one shown in Figure 1 –, which includes information about its status and resource availability, by placing it in its XeST container.

The first few lines in an advertisement correspond to the static properties of the XenoServer, such as owner, administrator, hostname, IP address and location. The next lines represent the resources available on the server, with each line indicating a resource.

Each resource is of the form

```
<Name, ResourceKind, PricingUnit, Availability, CostPerUnit>
```

where `ResourceKind` and `PricingUnit` are XOPIDs that are mapped to human-readable resource kinds and pricing units. For instance, as seen in the first line of the XenoServer advertisement in Figure 1,

```
defender.cl.cam.ac.uk:6:0
```

describes an entity that is in the area of responsibility of the XenoCorp called `defender.cl.cam.ac.uk`. The described entity is a resource (Kind=6), and more particularly a pentium2 (ID=0).

The mappings are XenoCorp-specific and XenoCorp-wide, which means that different XenoCorps can allow the use of different resources or pricing schemes. This gives XenoCorps a greater flexibility in defining and controlling what can be sold and bought.

To understand the format of resource representation, let us consider the first two lines describing resources that are for sale on the XenoServer in the advertisement shown in Figure 1. As explained in Figure 2, the first line indicates that best-effort access to a pentium2 CPU may be sold on this server at a price of 2 XEN. The second line shows that clients can also buy ms of CPU time per wall-clock sec at a price of 0.05 XEN per ms.

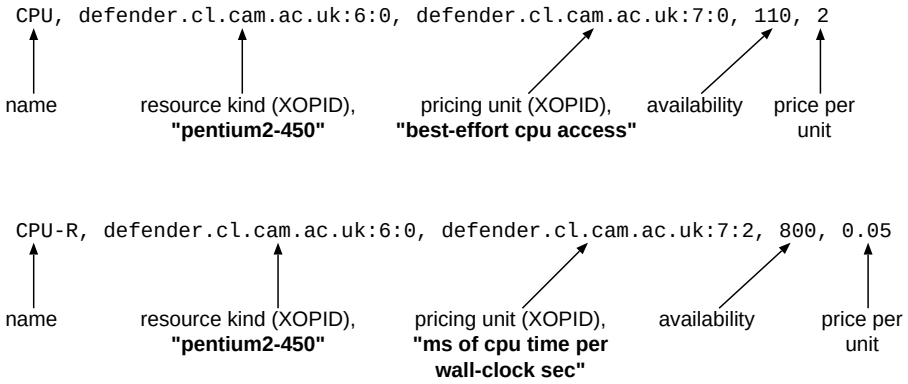

Figure 2: Representation of resources in a XenoServer advertisement

XIS will pull those advertisements from the individual containers regularly, and store them in its own XeST container in a structured manner. Clients who wish to locate a XenoServer that will be suitable for their needs can then either query XIS directly, or use XenoSearch [27] to perform more complex search operations.

2.2 XenoTrust

The public and open nature of the platform imposes a need for a trust management system. In the real world, servers and clients operate autonomously. Servers may be unreliable; they may try to overcharge clients, may not run programs faithfully, or may even try to extract client secrets. Clients may attempt to abuse the platform; they may try to avoid paying their bills, or to run programs with nefarious, anti-social or illegal goals.

In [6], we describe this threat model and propose the trust and reputation management architecture that is used in the XenoServer Open Platform. In [7], we provide a more concrete design of the above architecture, and

suggest that using an event-based publish/subscribe methodology for the storage, retrieval and aggregation of reputation information can help exploiting asynchrony and simplicity, as well as improving scalability for our system.

3 Role-Based Resource Management

Expressing and applying resource allocation policies in federated distributed systems is a hard task; resources are controlled by a large number of individual physical and administrative entities. There is a need to express policies defined by the administrators of each resource as well as by central "authorities", and combine the policies in a reliable and well-defined manner.

This need is becoming increasingly pressing in settings such as utility and public computing infrastructures [10], overlay testbeds [2] and computational grids [13], making the problem of resource management in federated systems an important one.

To allow representing, applying, and combining complex resource usage policies originating from a large variety of sources, we propose a *role-based resource management* scheme. The owner of the resources and other stakeholders can specify the desired policy, expressing which users or groups of users can be allowed access to which parts of resources.

Our approach can be seen as a development of Role-Based Access Control (RBAC), which has received significant research focus in the past. In RBAC, "access control decisions are determined by the roles individual users take on as part of an organization". The basis of RBAC is the concept of a *role*, which is essentialy a grouping mechanism used to classify users. RBAC allows the administrator to specify which users should enter which role, as well as which roles should be granted access to which resources. The nature of RBAC decisions is binary; one can either be granted access or not. Thus, when role entry conditions are conflicting, the most common approach is that if there is one that denies access then it simply overrides the others, in accordance to the "least privilege" principle [25].

When considering applying the same principles on distributed resource management, a key technical difference that emerges is that, in contrast to access control, which is binary, resource management is *quantitative*; the question becomes how much access to grant a user to a resource, rather than simply whether to grant access or not. Moreover, in global-scale systems there is usually no notion of a central authority controlling the distributed resources. Therefore, one can expect that – possibly conflicting – policies and roles will

need to be defined by a number of heterogeneous entities, that co-exist under separate administrations.

A problem that emerges as a result of these challenges is *conflict resolution*. Imagine, for exposition, a case where Bob, the resource owner, allocates 10% of its resources to Computer Science students and 8% to IEEE members, and user Alice belongs to both roles. Then, would Alice be given 10%, 8% or 18%? Is it Alice that Bob wanted to allocate 8% to, or is it all IEEE members on aggregate? And what happens if all 8% is already allocated to IEEE members when Alice's request appears?

It is easy to see that there is a need for a flexible, expressive and comprehensible system, able to combine role entry and resource allocation constraints, to allow efficient resource management in global-scale public computing infrastructures. The research challenges imposed are enhanced by the variety of entities who define roles, constraints and relationships as well as possible overlapping and conflicts between those constraints.

For the required resource management system to be able to operate successfully in a distributed, global-scale setting, we need to make sure it is designed and implemented in a decentralized fashion. There is no notion of a central authority responsible for the way resources can be managed, and to make our system efficient it is necessary to avoid bottlenecks and single points of failure. The *scalability* of our system can clearly only be served well by distribution of computation.

The rest of this paper is structured as follows. In Section 2 we describe the Role Description Language that is used. In Section 3 we analyze the proposed architecture, and in Section 4 we conclude.

3.1 Example settings

In this section we investigate some real-world settings where a flexible resource management system, such as the one we propose, is required to address the needs and challenges imposed by their large-scale, distributed and heterogeneous nature.

3.1.1 The XenoServer Open Platform

The XenoServer project [10, 24] is developing an infrastructure for global-scale public computing. The key entities in the XenoServer Open Platform are *XenoServers*, which are machines that lease computing resources in ex-

change for money, *clients*, who buy those resources, and *XenoCorps* that act as a trusted third party – much like the way VISA mediates between clients and retailers.

The relationships between clients and XenoServers, clients and XenoCorps, and XenoServers and XenoCorps are all many-to-many. XenoServer owners may wish to differentiate in the way they allocate resources to clients according to the XenoCorp they co-operate with, their location, or even their reputation [6].

Another challenge that emerges in this setting is that of the diversity of entities that can define roles and constraints. While it is up to the XenoServer owners to define how resources are to be apportioned between the clients, XenoCorps also have the right to enforce resource allocation policies on the XenoServers that they co-operate with.

3.1.2 PlanetLab

PlanetLab [2] is a distributed architecture consisting of a number of machines, termed *nodes*, scattered around the world. It allows an application to run across all or some of those nodes. The main purpose of PlanetLab is to serve as a testbed for overlay networks.

The main resource abstraction is termed a *slice*. Slice creation represents the allocation of a set of distributed resources to an application. A software component running on each node, called the node manager, is the ultimate regulator of how many resources are available to specific slices on that node.

Regardless of the resources promised to the slice at creation time, it is at the node manager's discretion to allocate less than the expected resources if necessary. Many nodes are donated by parties that impose specialized usage restrictions on the equipment, including limits on slice behaviour or blacklists. In addition, the node manager may enforce temporary resource restrictions, such as disciplinary actions. There is some ongoing work on mechanisms to allow imposing resource restrictions [3], but there exists a substantial need for a flexible way of representing complex policies in this setting.

3.2 Policy Description

The main elements of our Role-Based Resource Management architecture are *users*, who request *resources* and are grouped in *roles*, and *constraints*,

which define how resources are to be apportioned between roles.

Below, we explain how roles and constraints can be defined and combined to express complex resource allocation policies in a flexible, efficient and comprehensible manner.

3.2.1 Roles

A *role* is essentially a grouping mechanism for users. There are a number of actions to be taken to define the roles in which users may participate, specify which users are eligible to enter which roles, and associate these roles with resource management policies.

Roles have to be *declared*, in the same sense that variables would be declared in a programming language, along with the parameters that may apply. A role declaration includes the name of the entity that declares the role, the name of the role and its parameters. There is a different name space for each entity that declares roles.

The *entry conditions* for each role need to be defined. Entry conditions specify how a client can enter the role, and how roles interact. When a client enters a role, he remains a member of the role until membership is explicitly or implicitly revoked.

The format and usage of role declarations and role entry conditions are described in more detail in the following sections.

Role declarations To define a role, the administrator has to perform a *role declaration*. Roles can have parameters, in order to avoid declaring different roles for different occurrences of the same role.

In an open, large-scale system, it would be difficult to enforce a single system-wide name space for roles. Instead, the approach that we take is to name roles *hierarchically*, so that each entity that defines roles can have its own role name space. Role declaration commands are of the form:

```
Advertiser:RoleName(Parameter1, Parameter2, ...);
```

where the `Advertiser` is the entity, according to which the role is defined. For example, to define a role for users that, according to Bob, reside in the UK, and another one for those who Bob finds to live in a particular `City` in the UK,

```
Bob:InUK();
Bob:InUKCity(City);
```

Role entry conditions The administrator can specify which users can enter a role, by defining one or more *role entry conditions*. Again, for flexibility and openness, role membership is *subjective*. A user is not a member of a role globally, but a member of a role according to an `advertiser`.

Criteria for entering a role can range from physical properties to membership of other roles. Role entry conditions can be of the form:

```
roleEntry(Advertiser:Role(Parameters),
          Attribute1, Range1,
          Attribute2, Range2,
          ...);
```

in which case users whose property indicated by `Attribute1` has a value within `Range1`, and property indicated by `Attribute2` has a value within `Range2`, and so on, can enter `Role`. For example,

```
roleEntry(Bob:InUK(), "Country", "UK");

roleEntry(Bob:InUKCity("Cambridge"),
          "City", "Cambridge",
          "Country", "UK");

roleEntry(Bob:InUKCity("Oxford"),
          "City", "Oxford",
          "Country", "UK");

roleEntry(Jerry:Engineer(), "Occupation", "Engineer");
```

Alternatively, to allow clients to enter `Role` based on prior role membership of `Role1, Role2, ..., RoleN` according to the advertisers `Advertiser1, ..., AdvertiserN` respectively,

```
roleEntry(Advertiser:Role(Parameters),
          Advertiser1:Role1(Parameters),
          Advertiser2:Role2(Parameters),
          ...,
          AdvertiserN:RoleN(Parameters));
```

For example, to allow entry to the `CamEngineers` role to users who are engineers, according to Jerry, and who, according to Bob, reside in Cambridge,

UK, Alice can define the following entry condition – effectively resulting to the `CamEngineers` role being the intersection of the other two roles.

```
roleEntry(Alice:CamEngineers,
          Bob:InUKCity("Cambridge"),
          Jerry:Engineer)
```

whereas to allow entry to the `OxbridgeEngineers` role to users who are engineers, according to Jerry, and reside in Cambridge or Oxford, according to Bob, the following entry conditions can be used to make `OxbridgeEngineers` contain the union of the two roles:

```
roleEntry(Alice:OxbridgeEngineers,
          Bob:InUKCity("Cambridge"),
          Jerry:Engineer)

roleEntry(Alice:OxbridgeEngineers,
          Bob:InUKCity("Oxford"),
          Jerry:Engineer)
```

3.2.2 Constraints

To express a reservation or usage limitation on a resource, *constraint definitions* are used. A constraint definition is associated with a role. Thus, the constraint is applicable to all members of the role.

As constraints can be defined by different entities and can conflict with other constraints, for instance in cases where a user is a member of two roles for which there are two different constraint definitions for the same resource, we need an explicit way to prioritize constraints and resolve conflicts. This can be done by defining *constraint relationships*.

Below, we describe the format and usage of constraint definitions and constraint relationships.

Constraint definitions A *constraint definition* limits or guarantees the amount of a resource that members of a role can get. Constraint definitions are of the form:

```
Advertiser:Role(Parameters)
           ConstraintKind(Resource, Parameters);
```

where `Advertiser` is the entity that is imposing the constraint, and `Role` is

the role whose members are subject to the constraint. `ConstraintKind` is an identifier that describes what kind of limitation or reservation the constraint is meant to indicate, such as `limitEach, limitGroup, reserveEach or reserveGroup`.

Resource is the kind of resource that the constraint applies to, and Parameters indicate the extend of the limitation or reservation. For example:

```
Alice:CamEngineers() limitGroup(CPU, 8%);
```

would limit the aggregate CPU usage by Cambridge Engineers to 8%.

Constraint relationships In order to allow the administrator to define how conflicting constraints are resolved, we introduce *Constraint relationships*. Each relationship gives a series of pattern-matches for existing constraints, and then a replacement constraint to be generated in their place. The format of constraint relationships is:

```
ConstraintRelationship(Constraint1,
                       Constraint2,
                       ...  ,
                       Expression1,
                       Expression2,
                       ...  ,
                       Replacement);
```

For example, to express that where different entities give constraints about maximum CPU usage, the minimum ought be taken, one can use:

```
ConstraintRelationship(
    "X:R1 limitEach(CPU, A)",
    "Y:R2 limitEach(CPU, B)",
    "X != Y",
    "Alice:R1 limitEach(CPU, min(A,B))");
```

Notice that, as the two conflicting constraints are replaced by the new one, which of the two roles the new one refers to does not make any difference, as for the conflict to exist the user must be a member of both roles. To express that, when a role is a sub-role of another one, constraints imposed by the more specific role override the ones imposed by the less specific role,

```
ConstraintRelationship(
    "*:R1 limitEach(CPU, A)",
    "*:R2 limitEach(CPU, B)",
```

```
    "R1 < R2",
    "Alice:R1 limitEach(CPU, A)");

ConstraintRelationship(
    "*:R1 limitEach(CPU, A)",
    "*:R2 limitEach(CPU, B)",
    "R2 < R1",
    "Alice:R2 limitEach(CPU, B)");
```

3.3 Resource allocation

In the previous sections we have described the Role Description Language used in our system, and explained how roles and constraints can be defined. This section examines how our system uses the set of roles and constraints defined, in order to determine whether to grant or deny a resource allocation request.

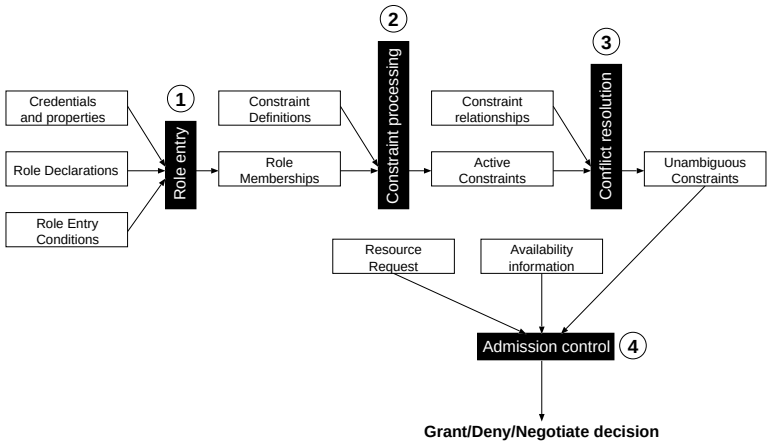

Figure 3: Role-Based Resource Management architecture

3.3.1 Role entry

The first step to reach an admission control decision is to decide which roles a user is a member of. This process requires as input the credentials and properties of the user as well as the role declarations and entry conditins.

Entry conditions have to be examined and checked against the user's properties or credentials for the role memberships to be determined. For instance, the `"Country"` property of the user will determine if he should be allowed to enter the `"inUK"` role, and presentation of valid credentials would allow

the client entry to an `"Authenticated"` role.

Membership of roles that have more complex entry conditions, such as prior multiple role membership, can be decided by starting checking membership from the simplest roles and going up to the more complex ones. If the user is found not to be a member of one of the roles, the process stops and the user does not enter the complex role. While the administrator has to make sure that no loops exist in role entry conditions, the algorithm produces a consistent set of role memberships, as there are no role entry conditions that allow membership in a role on the condition that there is no membership to another one.

The output of this process is the set of *role memberships* in which the user participates.

3.3.2 Constraint processing

One of the purposes of the proposed Role-Based Resource Management system is to allow the server administrator to express policies like "Allow Cambridge Engineers access of up to 10% on the CPU, and guarantee Oxford Engineers access of 5% of the network bandwidth". To express such policies, we use *constraint definitions*, as described in 3.2.2.

Once the role memberships have been determined, they are associated with these constraints by checking the `RoleName` property of each constraint definition and associating it with the respective role. The set of constraints is reduced as constraints that are not associated with any roles are ignored further on since they are unable to affect the admission control decision.

This process results to a set of potentially conflicting *active constraints*. It consists of the constraints that have been associated with at least one of the roles, which the user is a member of.

3.3.3 Conflict resolution

Conflicts between resource allocation constraints are resolved using the constraint relationship rules.

The outcome of the first two stages can result in a set of constraints that can possibly conflict when two or more constraints applying to the same resource. Then, there is a need for a mechanism to resolve such conflicts in a flexible and effective way.

To understand the difficulty of resolving conflicting resource allocation policies, one can consider the following scenarios. The resource owner specifies a policy suggesting that each customer paying with VISA card should be guaranteed 5% of the network bandwidth. The owner defines two more policies: the first one specifies that domestic customers should be guaranteed 8% of the network bandwidth as a means of attracting local users. The second one limits the use of the network bandwidth to 2% for users that do not pay their bills regularly.

For user Alice, who pays by VISA card and does not pay her bills regularly, the system would work fine if system made sure that, when constraints are conflicting, the most restrictive constraint replaces the rest. Alice would get 2% of the network bandwidth. However, for user Bob, who pays by VISA and is a domestic customer, applying the most restrictive constraint is not helpful; Bob would get 5%, while he should be getting at least 8% as any other domestic customer. It is clear that the question is not one of a trivial, technical nature, and that no global or hard-coded conflict resolution policies can be enforced.

A simple solution could be a first-match one; when a number of constraints apply to a resource, the one that was defined first overrides all others. However, this would be too inflexible, and the decision would rely on the order in which constraint definitions would be declared. Another easy fix could be to prioritize constraint definitions in a static manner, for instance by including a priority number along with each constraint definition. The problem with this approach is that it puts a very large administrative burden on the server owner, as resolving conflicts manually can become extremely difficult for large numbers of constraints.

The approach that we take is to allow the administrator to define how conflicts should be resolved explicitly, by declaring *constraint relationships*, as described in 3.2.2. The set of active constraints is checked against the constraint relationships, and sets of conflicting constraints are replaced by single constraints and resolved.

When this stage is finished, a set of *unambiguous constraints* is produced.

3.3.4 Admission control

The resource allocation request is checked along with the current resource availability as well as the set of unambiguous constraints, in order to reach a grant/deny decision.

In order to specify the general default behaviour of the system, a simple

parameter can be set to specify either that access to resources can be allowed if not explicitly prohibited by a constraint, or prohibited if not explicitly permitted by a constraint.

Then if the request does not exceed availability or violate any of the constraints in the unambiguous constraints set, and if the default policy is to allow access unless a constraint prohibits it, the request is granted. If the policy is to prohibit access unless a constraint permits it, then there will have to be a reservation constraint for the resource to be granted.

3.4 Related work

Access control is an area that have received extensive research focus over the last three decades [19].

Our work uses some of the techniques invented in role-based access control, but is fundamentally different in that we use roles in a different, soft, quantitative setting, to express complex resource allocation policies, where roles and policies need to be combined instead of overriden in accordance to the "least privilege" rule.

One of the first works establishing role-based access control as we know it today was [8], outlining the ideas of RBAC and providing a concrete formal description of role definition and membership, as well as recognizing the importance of the separation of duty problem — a refinement of those ideas, as well as an implementation, are provided in [9]. In [26], a family of well-defined RBAC models is introduced.

Several role-based systems have been devised over the last ten years. [22] provides a framework for the administration of roles and access rights, and focuses on the organization of roles by allowing the explicit declarationon of relationships between them. [21] defines roles as sets of rights and duties, which is similar to our distinction of roles from constraints. Relationships between roles are considered, as well as meta-policies for resolving conflicts.

[14] combines roles and policies applied by a diverse set of sources, in order to assemble a global layer for the interoperability of heterogeneous databases. [12] takes it even further by escaping from the "central authority" model and understanding the challenges imposed by applying RBAC to open, large-scale systems, while also providing a flexible and comprehensible role description language. Individual services need to be able to perform naming and authentication, and to themselves define how their resources will be used.

A generalized version of RBAC is proposed in [4]. The work is motivated by the need to enhance RBAC in order to enhance ease of use and expressiveness for applying security policies in an "aware home". This approach goes beyond the common subject-centric approach to role management, by allowing object-centric or environment-centric policies to be defined. Object-centric roles are essentially the same as constraint definitions.

The Ponder language [5] provides a means of specifying security policies associated with roles, allowing the declaration of positive and negative authorisation policies, as well as meta-policies for conflict resolution and role inheritance. The RT framework [20] combines role-based access control with hard security trust management, for efficient access control in large-scale systems.

4 Dissertation Schedule

The following work remains to be completed:

- Complete the implementation of the Platform, refining security, auditing, access control, accounting and charging mechanisms. *(2 months)*
- Implement a Role-Based Management System, to support expressing and applying complex resource allocation policies. *(2 months)*
- Implement realistic test applications, as well as use off-the-shelf software, to perform experiments and evaluate performance, usability and efficiency of the developed code deployment architecture. *(1 month)*
- Investigate the possibility of allowing XenoServers to negotiate with clients when they are unable to fulfill their resource requirements, and evaluate the potential risks for the XenoServer Open Platform in cases of misuse of resources by clients. *(1 month)*
- Write the dissertation. *(6 months)*

The likely structure of the dissertation is:

Introduction. In this chapter, I will provide an overview of my area of research, and outline its importance.

Research Context. This chapter will include a survey of related research approaches – such as computational grids, virtual machine monitors, and utility computing infrastructures. Research challenges faced by large-scale public computing systems will be discussed.

Platform Overview. Here, an overview of the XenoServer Open Platform will be given. My work will be positioned in the research context described above, and the motivations for public computing will be explained.

Resource Discovery. The mechanisms for resource discovery in the XenoServer Open Platform will be analyzed in detail, as briefly described in 2.1. Trade-offs and alternative approaches may be considered.

Resource Management. In this chapter, I will describe the proposed architecture for efficient distributed resource management. This will include Role-Based Resource Management, as outlined in 3.3.

Trust Management. This chapter will include a threat model for the XenoServer Open Platform, and explain the motivation for a trust management system. A detailed description of the design of XenoTrust will be provided.

Implementation. Details about the implementation of the architecture.

Evaluation. This chapter will provide an evaluation of the architecture in terms of performance, usability, and convergence, as well as some initial estimates on whether the platform has a self-financing potential.

Conclusion. Here I will summarize my research, and discuss conclusions that may have been reached.

The plan is to have a first draft in July 2004. The final submission date will depend on feedback from the drafts, with the target being September 2004.

References

[1] Paul R. Barham, Boris Dragovic, Keir A. Fraser, Steven M. Hand, Timothy L. Harris, Alex C. Ho, Evangelos Kotsovinos, Anil V.S. Madhavapeddy, Rolf Neugebauer, Ian A. Pratt, and Andrew K. Warfield. Xen 2002. Technical Report UCAM-CL-TR-553, University of Cambridge, Computer Laboratory, January 2003.

[2] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. Planetlab: An overlay testbed for broad-coverage services, January 2003. PlanetLab Design Note PDN-03-009.

[3] Brent Chun and Tammo Spalink. Slice creation and management, July 2003. PlanetLab Design Note PDN-03-013.

[4] M. Covington, M. Moyer, and M. Ahamad. Generalized role-based access control for securing future applications, 2000.

[5] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The ponder policy specification language. *Lecture Notes in Computer Science*, 1995, 2001.

[6] Boris Dragovic, Steven Hand, Tim Harris, Evangelos Kotsovinos, and Andrew Twigg. Managing trust and reputation in the XenoServer Open Platform. In *1st International Conference on Trust Management*, June 2003.

[7] Boris Dragovic, Evangelos Kotsovinos, Steven Hand, and Peter Pietzuch. Event-based distributed trust management. In *2nd IEEE International Workshop on Trust and Privacy in Digital Business*, September 2003.

[8] D. Ferraiolo and R. Kuhn. Role-based access controls. In *Proc. 15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[9] David F. Ferraiolo, John F. Barkley, and D. Richard Kuhn. A role-based access control model and reference implementation within a corporate intranet. *j-TISSEC*, 2(1):34–64, February 1999.

[10] Steven Hand, Tim Harris, Evangelos Kotsovinos, and Ian Pratt. Controlling the XenoServer Open Platform, November 2002. Under submission to OpenArch'03.

[11] Colin G. Harrison, David M. Chess, and Aaron Kershenbaum. Mobile

Agents: Are they a good idea? Technical report, IBM Research Division, T. J. Watson Research Center, Yorktown Heights, New York, 1995.

[12] R. J. Hayton, J. M. Bacon, and K. Moody. Access control in an open distributed environment. pages 3–14, 1998.

[13] William E. Johnston, Dennis Gannon, Bill Nitzberg, Leigh Ann Tanner, Bill Thigpen, and Alex Woo. Computing and data grids for science and engineering. In *Proceedings of the 2000 conference on Supercomputing*, page 52. IEEE Computer Society Press, 2000.

[14] D. Jonscher and K. R. Dittrich. Argos – A configurable access control system for interoperable environments. In T. C. Ting and D. Spooner, editors, *Database Security, IX: Status and Prospects*, pages 43–60. Chapman & Hall, 1996.

[15] Evangelos Kotsovinos. First year report, July 2002. Computer Laboratory, University of Cambridge.

[16] Evangelos Kotsovinos. Thesis proposal, July 2002. Computer Laboratory, University of Cambridge.

[17] Evangelos Kotsovinos and Tim Harris. Distributed resource discovery and management in XenoServers. In *7th CaberNet Radicals Workshop*, October 2002.

[18] Evangelos Kotsovinos and David Spence. The XenoServer Open Platform: Deploying global-scale services for fun and profit. In *SIGCOMM '03 poster session*, August 2003.

[19] Butler Lampson. Protection. In *Proceedings of the 5th Annual Princeton Conference on Information Sciences and Systems*, pages 437–443, Princeton University, 1971.

[20] Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust management framework. In *Proc. IEEE Symposium on Security and Privacy, Oakland*, May 2002.

[21] Emil C. Lupu, Damian A. Marriott, Morris S. Sloman, and Nicholas Yialelis. A policy based role framework for access control. In *Proceedings of the first ACM Workshop on Role-based access control*, page 11. ACM Press, 1996.

[22] Matunda Nyanchama and Sylvia Osborn. Access rights administration in role-based security systems. In J. Biskup, M. Morgernstern, and

C. Landwehr, editors, *Proc. 8th IFIP WG 11.3 Working Conference on Database Security (Database Security VIII: Status and Prospects) (Bad Salzdetfurth, Germany, Aug. 23–26*, volume A-60 of *IFIP Transactions*, Amsterdam, The Netherlands, 1995. North-Holland (Elsevier).

[23] I. Peterson, D. Culler, and T. Anderson. Planetlab: A testbed for developing and deploying network services, June 2002. Technical white paper.

[24] Dickon Reed, Ian Pratt, Paul Menage, Stephen Early, and Neil Stratford. Xenoservers: accounted execution of untrusted code. 1999.

[25] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, September 1975.

[26] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[27] David Spence and Tim Harris. XenoSearch: distributed resource discovery in the XenoServer Open Platform. In *High Performance Distributed Computing (HPDC-12)*, June 2003.