Controlling the XenoServer Open Platform

Steven Hand, Tim Harris, Evangelos Kotsovinos, Ian Pratt
University of Cambridge Computer Laboratory
J J Thomson Avenue, Cambridge, UK, CB3 0FD
{firstname.lastname}@cl.cam.ac.uk

Abstract— **This paper presents the design of the XenoServer Open Platform: a public infrastructure for wide-area computing, capable of hosting tasks that span the full spectrum of distributed programming. The platform integrates resource management, charging and auditing. We emphasize the control-plane aspects of the system, showing how it supports service deployment with a low cost of entry and how it forms a substrate over which other distributed computing platforms can be deployed.**

I. Introduction

The XenoServer[1] project [23], [13] is building a public infrastructure for wide-area distributed computing, creating a world in which XenoServer execution platforms are scattered across the globe and available for any member of the public. This allows users to run programs at points throughout the network to reduce communication latency, avoid network bottlenecks, and minimize long-haul network charges. Also, it can be used to deploy large-scale experimental services, and to provide a network presence for transiently-connected mobile devices.

Our approach is distinguished from existing work on mobile agents, execution platforms, code hosting and the like by two principles:

1. *Tackling difficult problems at the same time.*
Acceptable designs for execution environments, resource management, resource discovery, authentication, privacy, charging, billing, payment, and auditing are all crucial to the success of our platform as an infrastructure service open to and accepted by the public. Existing work has tackled individual subsets of these problems, but tensions between the issues concerned mean that solutions proficient in some dimension are lacking in another.

2. *No brave new world.*
Our platform will host applications written in today's programming languages against existing APIs – and, we believe, those written with tomorrow's languages and libraries. We do not want to mandate a particular code distribution format or a particular middleware toolkit for distributed programming.

[1]The name derives from the Greek word "$\xi\varepsilon\nu o\varsigma$" (xenos), which means foreign or unknown, much like the tasks that XenoServers accept and safely execute.

The XenoServer platform can be classed as *open* in two different ways. Firstly, as a service deployment infrastructure, it provides an extremely low cost of entry when compared with dedidicated-hosting facilities. This is both a low intellectual cost, by supporting existing programming environments, and a low monetary cost, by supporting fine-grained resource management over flexible timescales, and at a level much smaller than renting complete machines. Secondly, the system is structured so that it can host new kinds of execution environments, new kinds of resource discovery systems, new models for charging, new tradeoffs between privacy, performance, and so on. Our open platform provides the mechanisms necessary for its constituent servers and organisations to define the policies for themselves.

Overview

In Section II we introduce the general architecture of the XenoServer Open Platform, showing the entities involved and the minimum requirements for deploying tasks, performing authentication, and charging for resource consumption. Sections III–V then build on this to provide higher-level interfaces for publishing XenoServer status information, performing resource discovery and constructing flexible XenoServer systems.

In companion papers, we focus on the technical aspects of our current implementation of the XenoServer component and the hardware virtualization it achieves [2]. This allows authenticated clients to deploy tasks over a flexible, resource-managed XenoServer, supporting Linux and Java execution environments. Elsewhere, we introduce the XenoStore distributed file system, built over the foundations presented here, to provide a shared global storage network [22], and propose a trust management architecture for our platform [8].

Related work

Many research groups have been developing computational grids [15]. These construct virtual supercomputers dynamically from geographically dispersed and heterogenous resources linked by high-speed networks. Such infrastructures include Globus [12], NEOS and Condor [11], SNIPE [9], Javelin [18] and Globe [25]. The PlanetLab project is following a similar approach, which substantiates an overlay network to serve as a testbed for a new class of widely distributed network services [21]. GridBank [3] combines features found in computational grids with accounting mechanisms. XenoServers are fundamentally different from grids and overlay approaches, as the latter provide application-level programming models and interfaces for sharing existing resources rather than system-level support in a large-scale, federated system with competing users and tasks. In many ways, we share the goals of Public Computing Platforms [24], although our architectural approach is broader in scope.

Virtual machine (VM) technologies allow unmodified guest operating systems to run in virtualized systems multiplexed over a single physical machine. This approach is taken by VMWare [26] and Virtual PC [6]. The vMatrix [1] project is based on VMWare, building a platform for moving code between different machines. However, hosting unmodified operating systems usually has negative effects on both reliability and performance. Moreover, vMatrix does not address the issues related to a large-scale deployment of the system, like authentication, discovery of participating machines, charging, auditing, and so on – nor how to accommodate tasks that require different kinds of execution environments.

As explained in Section V-A, in the XenoServer Open Platform there is no need to check for "safe" code, or for guaranteed termination – the only person hurt is the client deploying the code, not the hosting XenoServer or its other clients. This allows XenoServers to accept a broad range of existing code execution environments: the XenoServer Open Platform immediately becomes the ideal testbed for wide-scale deployment of research prototypes. This flexibility is, we believe, unique to our platform. There is no requirement for binaries to be digitally signed by a trusted compiler (as in SPIN [4]), to be accompanied by a safety proofs (as with PCC [19]), to be written in a particular language (as in SafetyNet [16] or Java-based systems), or to rely on a particular middleware (as with mobile-agent systems [17]).

II. General Architecture

Figure 1 illustrates the high-level architecture of the XenoServer Open Platform, distinguishing the various roles and interfaces. On the left hand side we see a XenoServer, on the right hand side a client, and at the top an entity called XenoCorp. XenoServers host tasks that are submitted by clients and XenoCorp acts as a trusted third party.

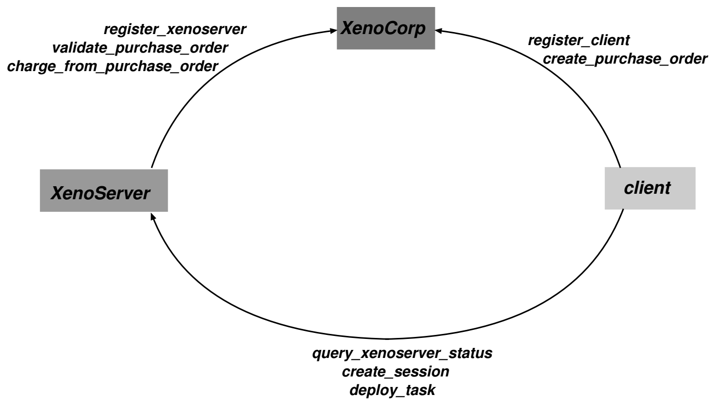

Fig. 1. The XenoServer Open Platform architecture

For exposition, it is easiest to assume a single XenoCorp. However our architecture is designed to support multiple competing entities, providing that they follow the same basic interfaces – much as the commercial world supports several banks. However, to set the general scene, it is important to realize the separation between a XenoCorp and the organizations running XenoServers. The former provides authentication, auditing, charging and payment, and has contractual relationships with clients and with XenoServer operators – much as VISA or MasterCard act as intermediaries between credit card holders and the merchants from which they make purchases.

The XenoServers themselves may be run by disparate organizations much as server hosting facilities are currently operated. We expect XenoServers will operate on well-maintained machines with long-term network presence – not in "spare cycles" on users' desktop systems.

A. Usage overview

The general operation of the XenoServer Open Platform can be described by considering four successive stages: registration, advertisement & discovery, deployment, and management. These stages are analyzed further in the following paragraphs. Following this we describe the operations depicted in Figure 1 in more detail.

A.1 Registration

Registration is the process of establishing an identity and obtaining credentials allowing participation in the platform. Both XenoServers and clients must be registered with a XenoCorp before they can host or submit tasks. XenoCorps are discovered using external mechanisms – advertising, word-of-mouth, and so on. Registration is an infrequent operation; it is undertaken when a company launches a new XenoServer or when a client wishes to use the platform for the first time. The formats and inference rules of attribute-based access control systems are ideal for representing and managing registration credentials [20].

XenoCorps may differ in the charging model that they enforce or in the privacy that they provide clients – perhaps whether a XenoServer hosting a job knows the client's "real world" identity. XenoCorps may also differ in how XenoServer operators receive payment; for instance, whether the amounts XenoServer operators receive are guaranteed, or they whether are a share in XenoCorp's overall profit. This can shift the exposure to fraud borne by XenoCorps and XenoServer operators.

A XenoServer must be registered in order to be eligible to claim payment for hosting jobs. XenoCorp may require that the XenoServer's operators enter into a contractual relationship with it, for instance agreeing to correctly host the jobs placed on it.

Clients register with XenoCorp in order to be able to place jobs on XenoServers and to set up an account for the charges incurred. The client must present some means of settling these charges. For example, in our public wide-area deployment, by providing a credit card number to bill.

A.2 Advertisement and Discovery

XenoServer operators compete for the business of clients. Crucial to the success of this is a way for XenoServers to *advertise* their capabilities, resources and prices, and for clients to discover servers which match their requirements.

In designing the platform, we are faced with the choice of how much support should be provided for these "matchmaking" services. One extreme option is for XenoCorps to be involved in XenoServer selection. However, aside from the complexity of designing appropriate job-description formats and scalable reqiurements-matching algorithms, this would complicate the relationship that XenoCorps hold with their affiliated XenoServers. Retaining XenoCorp as simply a trusted third party avoids accusations that it may favour particular XenoServer operators.

Within the core architecture, all that is required is some means by which a client may obtain the current capabilities of a particular XenoServer. This interface is represented in Figure 1 by the *query_xenoserver_status* operation exported by XenoServers.

It is unreasonable to expect that each client will individually poll all of the XenoServers in order to assess their suitability. Indeed, the core architecture does not in itself even provide a mechanism for a client to enumerate the servers. Our view is that this is rightly so; such a facility can be constructed over the core architecture rather than being ingrained within it. We return to this point in Section III when we introduce how a XenoServer Information Service can be constructed.

A.3 Deployment

Once a client has selected a XenoServer and determined that it is suitable for running a particular task, the next step is *deployment*. We have a fairly broad notion of what might be meant by deploying a task, for example:
- starting a well-defined server (e.g. Quake-3 v1.07),
- instantiating a new execution environment on the XenoServer, booting a particular operating system on it, and then shipping binaries to it for execution,
- transferring a binary, or sources to be compiled, to an existing virtual machine, and then running the executable.

This functionality is broken into two steps: firstly a *session* is created which establishes an agreement between a client and a XenoServer regarding the resources to be provided, and the payment to be made; secondly, a task or tasks are *deployed* on the XenoServer in the context of that session. In the following we elaborate on the mechanisms used in the establishment of sessions.

Session requirements

During deployment, a client must come to agreement with the intended XenoServer regarding the terms and conditions of their future cooperation – in other words, the client needs to specify its expectations from the XenoServer, and the latter has to acknowledge that those can be met while the session is operating. A client is responsible for creating an appropriately-provisioned session before deploying tasks within it.

In our design, resource requirements and availability are represented using XML. At the deployment stage, the client sends an XML description of its requirements to the XenoServer. The XenoServer then tries to match them with the available resources. The integrity and consistency of those descriptions can be checked using the built-in XML Schema mechanism [10]. An illustrative example of such a description is shown below:

```
<?xmlversion="1.0" encoding="UTF-8"?>
<EnvironmentalRequirements>
  <Basic>
     <EnvName>MyEnvironment</EnvName>
     <EnvType>Linux</EnvType>
     <EnvKernel>2.4.18</EnvKernel>
     ...
  </Basic>
  <Network>
     <EnvIPAddresses>1</EnvIPAddresses>
     ...
  </Network>
  <QoS>
     <CPUmsps>150</CPUmsps>
     <NetMbps>5</NetMbps>
     ....
  </QoS>
  ...
</EnvironmentalRequirements>
```

In the above example, the client is asking for a Linux execution environment, using the 2.4.18 kernel. Also, the XenoServer is asked to allocate a globally-valid IP address to the execution environment (rather than using network address translation), and to provide it with a network bandwidth of 5 Mbps –this is not a network-wide guarantee; it is similar to installing a 5Mbps network interface on that XenoServer. The session is to receive 150 ms of CPU time for every second of real time. Extensions to this basic format may specify the burstiness of these allocations, or provide separate peak and mean-rate requirements.

Purchase orders

Apart from the requirements specifications, the session creation request must be accompanied by a *purchase order* created by a XenoCorp with which the XenoServer is registered and subsequently signed by the client initiating the session. This identifies the account to charge for running the session, and may contain limits imposed by the XenoCorp, or constraints made by the client on the resources that can be consumed, on the type of session that may be requested, or on the XenoServers on which the purchase order may be spent.

Before the session is created, the XenoServer must *validate* the purchase order with the issuing XenoCorp. Again, XenoCorps will differ in how they implement this step. Order validation represents the point at which the XenoServer's operator has accepted a session, has confirmed that it can provide the requested execution environment, and has made a positive admission control decision. Beyond this, the semantics of order-validation are something that must be agreed contractually between each XenoCorp and its associated clients and XenoServers.

At one extreme, the validation interface may be implemented in a daemon task that the XenoCorp places on registered XenoServers – perhaps a simple check that the purchase order was issued by that XenoCorp. At the other extreme, the validation interface may be implemented remotely by XenoCorp, and cause pre-payment of the maximum amount, perhaps incorporating a digital coin into the purchase order [5].

A.4 Management

Once a session is running, it is important that both the client and the relevant XenoServer perform some ongoing management. In the case of the client we allow any management infrastructure desired – it may simply deploy a management task alongside the others it places within a session. For example, if the client requests a simple resource-managed Linux environment then a management interface may return the initial "boot time" messages produced as that environment starts, before presenting the user with a shell. Alternatively, the management interface might accept control operations over some existing network protocol.

From the point of view of the XenoServer itself, the most important ongoing task is to account for resource usage and to make appropriate charges against the purchase orders that fund those sessions. An interesting problem here relates to the provision of an audit-trail – this is required in case a task performs anti-social or illegal activities. In our current model, XenoServers use the same *charge_from_purchase_order* interface to provide this log by annotating charges with information about the current activities of the session, effectively identifying what it is that is being paid for. If sufficiently authorised, an enforcement agency may request XenoCorp to correlate payments with purchase order creation.

B. Interfaces

We now consider in more detail the operations between each of the core components of the XenoServer platform.

B.1 Operations exported by a XenoCorp

1. **Register client:** Clients send registration requests to the XenoCorp, specifying the details necessary for the proposed account, such as name, address, and charging information. For instance, in our prototype deployment

of XenoServers, the first XenoCorp will provide a simple web-based form for performing registration.
The result of this is the production of unique credentials for the clients, with which they can identify themselves to XenoCorp or to XenoServer operators. XenoCorps will vary according to the information revealed in the credentials – while they must identify the issuing XenoCorp, they may vary between indicating that "this is a client known to me" and indicating the actual identity of the client.
2. **Register XenoServer:** Potential XenoServers have to subscribe to a XenoCorp in order to join the platform and start servicing clients. To register, a XenoServer will need to provide information about its owner, such as name, address, and bank account or credit card details, as well as the specifications of the machine, including the hardware architecture family, type of CPU, available memory, and network connectivity. The result of this is the production of unique credentials for the XenoServer.
3. **Create purchase order:** A purchase order represents a client's commitment to funding a session, subject to certain constraints. The creation of a purchase order is a two stage process. First of all the client requests that a XenoCorp issues it a basic purchase order up to a certain amount. At that stage the XenoCorp may check the credit-worthiness of the client, may ring-fence the portion issued as a purchase order, and may endorse the order with restrictions that must be met in order for it to honour payment – for instance that the purchase order must be properly validated before the XenoServer selected to perform the session starts work. The client, before using the order to fund deployment, may then annotate it with further restrictions – for instance specifying that only the XenoServer that it selects may cash in the order.
4. **Validate purchase order:** The validation interface provides an ahead-of-execution step with which a XenoServer operator may check the correctness of a payment order funding a job it has received. A purchase order therefore flows from XenoCorp, to the client to whom it is issued, to the XenoServer on which it funds a session, and finally back to XenoCorp at the point of validation. The validation policy will vary between XenoCorps, but it provides a point at which the XenoCorp can agree to each transaction – for instance vetting that the purchase order has not already been presented to another XenoServer. Note that although the validation interface is conceptually part of a XenoCorp, its implementation need not be centralized – XenoCorps may rent space on XenoServers that they trust to perform validation functions.
5. **Charge from purchase order:** XenoCorp receives resource consumption claims from XenoServers, requesting for charging a certain amount against particular purchase orders. These requests are annotated with resource usage information to form an audit trail. The rate and detail of these messages will again vary between XenoCorps.

B.2 Operations exported by a XenoServer

1. **Query XenoServer status:** Registered clients query the XenoServer in order to obtain information about its status. The XenoServer supplies its current resource usage and availability information, along with its identification credentials.
2. **Create session:** A registered client connects directly to a selected XenoServer in order to create a session. This request has to include the client's expectations from the XenoServer, such as Quality of Service requirements (optional, like CPU percentage, minimum network bandwidth), environmental needs (necessary, specifying a version of an operating system or JVM needed to run its tasks), and any other kind of general constraints (like maximum amount to be spent on resource consumption).
3. **Deploy task:** A client deploys a task in the context of an existing session — a handle describing the relevant session is passed to the XenoServer along with an XML description of the task to be deployed.

III. XenoServer Information Service

In Section II we introduced the three core components of the XenoServer Open Platform. These provide the basic functions of authentication, task deployment, and charging for resource usage. However, in considering the core components, we deferred the question of how a client of the system selects an appropriate XenoServer.

One possibility would be to use a meta-directory service such as MDS-2 [7] – an LDAP-based implementation of the Grid Information Service (GRIS). However the strictly two-level hierarchy and the fairly rigid schema means that another solution may be preferable. This is particularly true since we wish to support a broad range of execution formats and capabilities – incorporating servers which can host only certain operating systems, or which offer relatively more or fewer resources at a variety of prices. Furthermore, we expect a large and dynamically changing number of competing XenoServers.

Therefore, although clients can directly query individual XenoServers to determine their status, it would be impracticable for them to do so in any non-trivial deployment. Instead, information about XenoServers is aggregated within a highly available information service. We term this the *XenoServer Information Service* (XIS); its role in the architectural picture is shown in Figure 2.

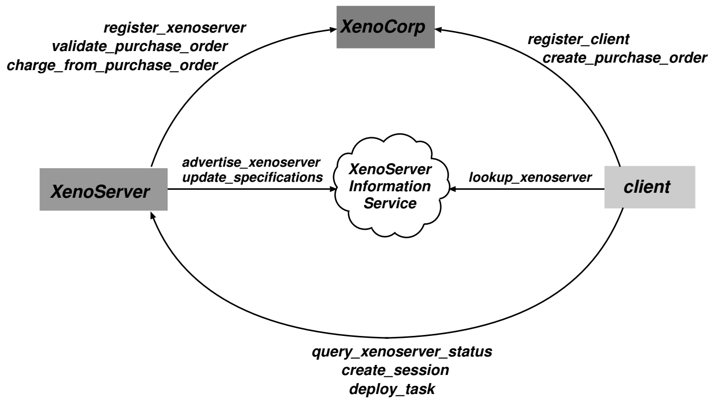

Fig. 2. XenoServer platform with XIS

As illustrated, this acts as an intermediary between XenoServers, who advertise information within it, and clients, who perform queries on the service. There are no architectural conventions requiring there be only one XIS although we envisage that this situation will develop naturally since it makes sense for XenoServers to advertise as widely as possible, and for clients to draw on the largest number of XenoServers when performing lookups. By analogy, the vast majority of the Internet prefers to use a single DNS hierarchy, even though there is no technical restriction to the deployment of separate namespaces (however, we note that alternatives which have been attempted do not find currency).

A. Interface

In designing the XIS it is necessary to balance the clients' desire for being able to perform expressive queries versus the desire for a straightforward and scalable implementation of the XIS. Crucially, we must identify (i) what information a XenoServer may store into the XIS and (ii) what kinds of query the XIS supports over this information. The approach we take here follows the general techniques that we have used in designing the XenoServer platform: the XIS is relatively agnostic about what kinds of data it holds and it defers the implementation of more advanced queries either to the clients themselves or to specialized Resource Discovery Systems (see Section IV).

A.1 Operations exported by XIS

1. **Advertise XenoServer:** XenoServers advertise their status within the XIS using an XML format as we did for specifying task requirements. While the scheme specifies some basic information that all XenoServers must provide – such as the XenoCorps with which they are registered – this can be augmented with information about current location, server and network load, average CPU utilization, number of clients connected, available bandwidth, and execution environments that are supported.
2. **Update Specifications:** XenoServers update their records in the XIS regularly. In the absence of updates, the XIS ages information and may ultimately discard it.
3. **Lookup XenoServer:** Clients view the XIS as providing an inverted file which indexes the information received by XenoServers. That is, as a mapping from each token that can occur in the XML specification onto a list of XenoServers which contained it. Clients perform queries, passing as parameters the token to query and the acceptable range for the token's value –for example, "token=CPUmsps, values=100-150" would cause a lookup for XenoServers that can provide 100 to 150 ms of CPU time per second to the task. The operation returns a number of matches and their corresponding places in the result stream. This interface allows clients to perform simple searches directly (perhaps specifying a token that represents a particular kind of execution environment) and to perform boolean searches by making several "lookup" requests on different tokens and combining the results.

B. Implementation

The initial XIS implementation is a distributed storage service optimized under the assumptions that (i) writes are always total rewrites (ii) XenoServers arrange that there is only ever one writer for each piece of data, (iii) reads of stale data are always either safe or can be verified, and (iv) information held in the XIS is for use by tools rather than humans, allowing inelegant internal design choices like explicit versioning.

The storage service comprises a number of distributed *nodes* which each hold some portion of the stored information; each node will hold a portion of the inverted file mapping from one or more tokens on to matching XenoServers. These nodes export the XIS interface to users interacting with it. They communicate with one another using a separate protocol. This design builds on that of our Pasta distributed file system, using the same techniques for replicating information between nodes of the XIS, and "drawing out" and caching this information toward the users that are requesting it. Self-certifying names [14] are used to ensure the authenticity of retrieved information and to allow clients to complain to the XenoServer's XenoCorp about inaccurate advertisements.

The initial deployment will place XIS nodes on constituent XenoServers – as we shall see in Section V the prototype XenoServer can host arbitrary isolated Linux environments. The first XenoCorp will require, as part of the XenoServer registration process, that the server identifies a node with which it has arranged to have its advertisement information injected into the XIS. This provides an incentive for XenoServer operators to host XIS nodes.

Of course, placing XIS nodes on XenoServers introduces the possibility that XenoServer operators may attempt to subvert the system – particularly if they are hosting a node which carries information about competitor XenoServers. This can be addressed by a variety of mechanisms, some technical and some contractual. Firstly, the XIS implementation will look as an ordinary task to the XenoServer; it is not in a XenoServer owner's interest to get a reputation for interfering with tasks. Secondly, the same part of the inverted file will be replicated over multiple XIS nodes. This aids access to the data and requires a larger conspiracy to lose part of it.

IV. Resource Discovery System

In Section II we introduced the three core aspects of the platform; XenoServers which hosts tasks, clients who submit tasks, and XenoCorps which act as trusted third parties. In that initial setting, clients needed an out-of-band mechanism for introduction to appropriate XenoServers. We then showed, in Section III how the XenoServer Information Service (XIS) provides a basic mechanism for XenoServers to publish their functionality, resource availability, and pricing. A low-level interface, presenting these advertisements as an inverted file, allowed clients to perform elementary searches.

We now introduce the higher-level Resource Discovery (RD) Systems which provide a more effective means for selecting XenoServers. Although the RD Systems build on the primitive operations exported by the XIS, they implement internally more intelligent searching algorithms and with clearer scope for differentiation between competing RD Systems.

A. Interface

An RD System is responsible for performing the matchmaking process between clients and XenoServers. It receives *find_xenoserver* queries from the clients containing specifications of the environmental and QoS requirements from the interface to be deployed. For instance, these requirements could be represented using XML, as was described in Section II-A.3.

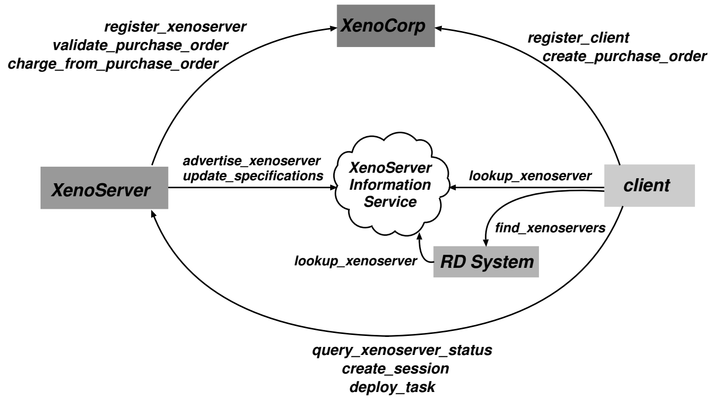

Fig. 3. XenoServer platform with XIS and RD System

After receiving the client's requirements and applying the search algorithms, the RD System suggests a number of suitable XenoServers to the client. This has to include information about all suggested XenoServers' pricing scheme, load, location, and so on. As with the XIS, the information returned is based on the advertisements received; clients may query the suggested XenoServers directly to obtain a spot price.

A.1 Operation exported by RD System

1. **Find XenoServers:** Given a resource requirement specification, the Resource Discovery System performs a search in the XenoServer Information Service on behalf of the client in order to find a number of suitable XenoServers.

B. Implementation

There may be multiple such RD Systems, either for simple competition (as exists between online search engines) or for specialisation to particular kinds of client, XenoServer or task. The algorithm with which the mapping is performed is entirely dependent on the RD mechanism. For example, different RD Systems may answer queries of the form:

- "Suggest 10 XenoServers with approximately equal network latency to the machines 131.111.202.88 and 141.163.61.250." This might be implemented using a distributed multi-dimensional search algorithm.
- "Suggest pairs of XenoServers which appear to have separate network links to 128.232.8.50".
- "Find a XenoServer which may run a job with these resource requirements at a total cost of £10".

We envisage many Resource Discovery Systems being implemented over the XenoServer foundations – as with networked services in general, XenoServers will provide an incremental deployment platform from which more resources can be acquired as the service grows (or from which only a low up-front cost is made if the service does not prosper).

There is no single platform-wide implementation of the matching algorithms, as the co-existence of several independent RD Systems will offer choice and diversity in resource discovery mechanisms and charging models. Some RD Systems might provide intelligent searching capabilities – such as finding a XenoServer that will minimize the total round-trip time for a given set of clients, while others will offer just basic searching functionality. Also, some RD Systems might be configured to charge clients for using the search mechanisms or to charge XenoServers for putting them higher in the suggestions list. Others can offer free services.

V. XENOSERVER CONTROL ARCHITECTURE

In Sections II–IV we have presented the core architecture of the XenoServer Open Platform and shown how, over that, we structure services for advertisement and resource discovery. For the final part of this paper, we turn our attention to the XenoServers themselves and their internal structure. We do so at two levels. First of all, Section V-A introduces the prototype XenoServer. Secondly, in Section V-B we present the decomposition of this system into a number of components, to extract common functionality and to aid the deployment of XenoServers based on existing platforms for mobile agents and code execution.

A. The Xen-based XenoServer

Figure 4 shows the general structure of our prototype XenoServer. This is based on a low-level component, termed the *Xen hypervisor*, which virtualizes the physical resources of the machine, apportioning them between the various environments that it hosts, by creating a virtual machine for each one. Each of these environments is called a *domain*, and is isolated from the other domains in terms of security and resource consumption. The hypervisor accounts the resource usage that each domain makes.

Thus, unsafe and unverified tasks can only be mischievous inside their execution domain, harming no one but the client who instantiated them. Each domain runs an instance of a *guest operating system*. These operating systems are specially ported to operate over Xen, accessing the virtualized hardware through appropriate device drivers. The companion paper describes Xen and its interfaces in detail, and shows the benefits of eschewing full virtualization of the underlying hardware [2]. Currently, we have developed one guest operating system providing a complete Linux environment, and have two further systems in progress to provide NetBSD and Win32 environments.

There can be multiple concurrent domains running the same guest operating system and so creating a resource-guaranteed session on a Xen-based XenoServer corresponds to booting a fresh domain for it.

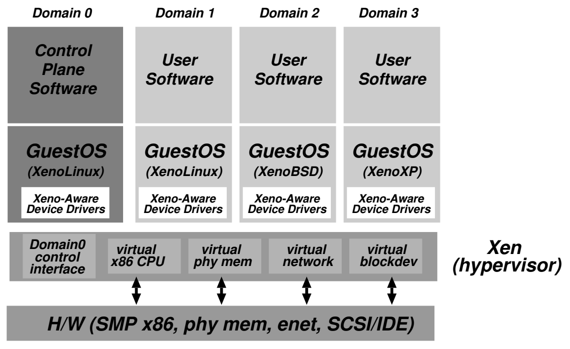

Fig. 4. The Structure of the Xen XenoServer

In addition to supporting guest operating systems, Xen exports a privileged control interface to the initial "Domain0" environment that it starts at boot-time. This domain's role is to run the control plane aspects of the system and, in particular, to export the session creation, deployment, and status interfaces seen earlier. Depending on their requirements, other domains may host code supplied by the XenoCorps with which the XenoServer has entered into a relationship – for instance performing validation of credentials locally if the security implications are deemed acceptable.

B. XenoServer Control Architecture structure

We will now introduce how the control aspects of a XenoServer can be structured. This is motivated by two examples. Firstly, we expect that most installations of XenoServers will co-locate groups of machines. For such clusters it is worthwhile to aggregate the session deployment and query interfaces to act over the cluster as a whole rather than distinguishing each machine individually. Secondly, clients may wish to use the platform to deploy tasks other than complete operating system instances over Xen – for instance, a XenoServer could host Enterprise JavaBeans (EJB) or .NET components using existing application-server packages. Our design reduces the effort necessary to deploy new kinds of execution environment.

These observations lead us to introduce an additional level of indirection between the system that ultimately hosts tasks and the interface with which clients interact. Figure 5 illustrates this structure, showing a configuration in which the deployment interface is exported by an *environment manager* which in turn starts and manages tasks through internal *task management* interfaces which control particular execution environments.

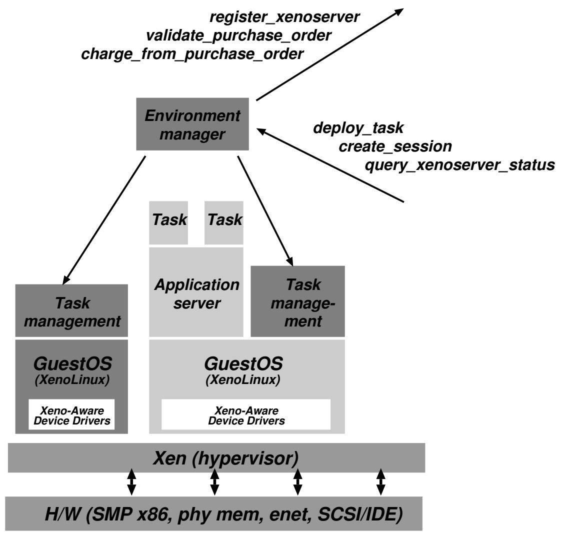

Fig. 5. The environment manager

The figure shows a XenoServer configuration in which one task manager is responsible for tasks running directly over the Xen hypervisor, and another is responsible for controlling tasks within a particular application server that is hosted in one of the domains.

The environment manager, if implemented as a Linux process, could itself execute within Domain0 – perhaps over this same Xen hypervisor, or over another in the same cluster. It is responsible for interacting with clients during session creation and task deployment – for instance activating the appropriate execution environment, or selecting which particular machine in a cluster should host a given task.

VI. Conclusion

In this paper we have introduced the XenoServer Open Platform with a particular focus on the control-plane aspects of its architecture. Only a core part of this must be fixed across the system; the basic interfaces presented in Section II to enable the exchange of security credentials, charging information, and deployment functions. At this level we aim to specify only the minimum that is required; decisions on charging policies are left to the individual XenoCorps and decisions on task execution formats are deferred to the individual XenoServers. In both cases this promotes scalability, flexibility, robustness, and competition. We introduced one way of building on these foundations in Sections III–V.

Looking beyond these facilities, many other components are desirable for a full-service public platform. We have not discussed our ideas about data storage and replication – for example how a XenoServer aquires the code and data from which to instantiate a new task or a new kind of execution environment; techniques for secure boot are of interest here. Similarly, we have not discussed how clients name their deployed tasks, or how they ensure that XenoServers adhere to their contracts. Broadly, as with the XIS and resource discovery services, we consider these to be components that can and should be implemented over the core XenoServer Open Platform – recent research has had no shortage of innovative schemes for solving such problems; what it has lacked is the global substrate that will allow them to be evaluated and deployed "in the large". This is what the XenoServer project is providing. Please contact us if you would like to be involved with the initial public deployment of the XenoServer Open Platform.

VII. Acknowledgements

We would like to thank the anonymous reviewers for their valuable comments while preparing the final version of this paper. Also, we would like to thank Jon Crowcroft, Tim Deegan, Alex Ho, Sugih Jamin and Andrew Warfield for their valuable suggestions, as well as Marconi PLC for the financial support of Evangelos Kotsovinos' research.

References

[1] A. Awadallah and M. Rosenblum. The vMatrix: A network of virtual machine monitors for dynamic content distribution. In *Proceedings of the 7th International Workshop on Web Content Caching and Distribution (WCW 2002)*, Aug. 2002.

[2] P. R. Barham, B. Dragovic, K. A. Fraser, S. M. Hand, T. L. Harris, A. C. Ho, E. Kotsovinos, A. V. Madhavapeddy, R. Neugebauer, I. A. Pratt, and A. K. Warfield. Xen 2002. Technical Report UCAM-CL-TR-553, University of Cambridge, Computer Laboratory, Jan. 2003.

[3] A. Barmouta and R. Buyya. GridBank: A Grid Accounting Services Architecture (GASA) for distributed systems sharing and integration. Technical white paper, University of Melbourne, 2002.

[4] B. N. Bershad, S. Savage, P. Pardyak, E. G. Sirer, M. Fiuczynski, D. Becker, S. Eggers, and C. Chambers. Extensibility, safety and performance in the SPIN operating system. In *Symposium on Operating Systems Principles (SOSP '95)*, volume 29(5) of *Operating Systems Review*, pages 267–284, Dec. 1995.

[5] D. Chaum, A. Fiat, and M. Naor. Untraceable Electronic Cash (Extended Abstract). In *Advances in Cryptology – CRYPTO '88 Proceedings*, volume 403 of *Lecture Notes in Computer Science*, pages 319–327. Springer-Verlag, June 1989.

[6] Connectix Corp. The technology of Virtual PC, 2000. Technical white paper, available from http://www.connectix.com/ downloadcenter/pdf/vpcw_wp/vpcw_overviewwp_sep1301.pdf.

[7] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Proceedings of the 10th IEEE Symposium on High Performance Distributed Computing*, Aug. 2001.

[8] B. Dragovic, S. Hand, T. Harris, E. Kotsovinos, and A. Twigg. Managing trust and reputation in the XenoServer Open Platform. Submitted for publication., Jan. 2003.

[9] G. E. Fagg, K. Moore, J. J. Dongarra, and A. Geist. Scalable networked information processing environment (SNIPE). In *Proceedings of the 1997 ACM/IEEE Conference on Supercomputing (CDROM)*, pages 1–13. ACM Press, 1997.

[10] D. Fallside. XML Schema Part 0: Primer, May 2001. W3C Recommendation, World Wide Web Consortium. Available at http://www.w3.org/TR/xmlschema-0/.

[11] M. C. Ferris, M. P. Mesnier, and J. J. Mor. NEOS and Condor: solving optimization problems over the internet. *ACM Transactions on Mathematical Software (TOMS)*, 26(1):1–18, Mar. 2000.

[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] K. A. Fraser, S. M. Hand, T. L. Harris, I. M. Leslie, and I. A. Pratt. The Xenoserver computing infrastructure. Technical Report UCAM-CL-TR-552, University of Cambridge, Computer Laboratory, Jan. 2003.

[14] K. Fu, M. F. Kaashoek, and D. Mazieres. Fast and secure distributed read-only file system. *Computer Systems*, 20(1):1–24, 2002.

[15] J. N. I. Foster, C. Kesselman and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Technical report, The Global Grid Forum, Jan. 2002.

[16] A. Jeffrey and I. Wakeman. A survey of semantic techniques for active networks. Nov. 1997. Available from the SafetyNet Project web site, http://www.cogs.susx.ac.uk/projects/safetynet/.

[17] D. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, K. Kosaka, D. Lange, K. Ono, M. Oshima, C. Tham, S. Virdhagriswaran, and J. White. MASIF: The OMG Mobile Agent System Interoperability Facility. In K. Rothermel and F. Hohl, editors, *Proceedings of the 2nd International Workshop on Mobile Agents*, volume 1477 of *Lecture Notes in Computer Science*, pages 50–67. Springer-Verlag: Heidelberg, Germany, 1998.

[18] M. Neary, A. Phipps, S. Richman, and P. Cappello. Javelin 2.0: Java-based parallel computing on the internet. In *Proceedings of European Parallel Computing Conference (Euro-Par 2000)*. IEEE Computer Society Press, August 2000.

[19] G. C. Necula. Proof-carrying code. In *Conference Record of POPL '97: The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 106–119, Paris, France, 15–17 Jan. 1997.

[20] Ninghui Li and John C. Mitchell and William H. Winsborough. Design of a Role-Based Trust-Management Framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, May 2002.

[21] L. Peterson, D. Culler, and T. Anderson. PlanetLab: A Testbed for Developing and Deploying Network Services, June 2002. Technical white paper, available at http://www.planet-lab.org/pubs/vision.pdf.

[22] I. A. Pratt, T. Moreton, and T. L. Harris. Storage, mutability and naming in pasta. In *2002 International Workshop on Peer-to-Peer Computing*, Apr. 2002.

[23] D. Reed, I. Pratt, P. Menage, S. Early, and N. Stratford. Xenoservers: accounted execution of untrusted code. In *Proceedings of the fifth Workshop on Hot Topics in Operating Systems (HotOS-VII)*, 1999.

[24] T. Roscoe and B. Lyles. Distributed Computing without DPEs: Design Considerations for Public Computing Platforms. In *Proceedings of the 9th ACM SIGOPS European Workshop, Kolding, Denmark*, September 17-20 2000.

[25] M. van Steen, P. Homburg, and A. S. Tanenbaum. Globe: A wide-area distributed system. *IEEE Concurrency*, 7(1):70–78, March 1995.

[26] VMware Inc. The VMWare virtual platform. Technical white paper, 1999.