Functional Architecture of AMSES: An Automatic Management System for E-service Systems

Yong Yan, Zhichen Xu, Raj Kumar
Mobile and Media Systems Laboratory
HP Laboratories Palo Alto
HPL-2002-296
October 15th , 2002*

A general development and automatic management platform for e-service systems is essential for e-service systems to always deliver satisfying, up-to-date, and trustworthy services cost-effectively to its customers, no matter where they are, when they want them, and how they access them. Following an application-driven approach, this paper derives the functional architecture of a general development and automatic management system, named AMSES.

The AMSES aims for providing automatic management for the lifecycles of multiple distributed e-service systems. The AMSES is composed of two kinds of functionalities: one for managing the operational functions and development of an e-service system, and another for managing the interactions among multiple e-service systems. To support the specified functionalities of AMSES, this paper presents four general management mechanisms: VOE (Virtual Operating Environment), VS (Virtual Service), i-resource (Intelligent resource) and a-resource (Active resource). The VS mechanism supports diverse services and resource outsourcing, and to provide protection as well as performance monitoring. i-resource mechanism and a-resource mechanism cooperatively implement resource virtualization of heterogeneous resources and distributed self-management of resources and service. The VOE mechanism provides encapsulation and protection for an e-service system.

In the AMSES, an e-service runtime system is represented as a group of distributed i-resources, where each i-resource is realized with a group of a-resources. The self-management is realized in four control loops with different response time-scales. At last, the functional architecture of the AMSES is derived based on the four mechanisms. The AMSES is an endeavor that aims to provide a common vision to derive a general and automatic management platform for e-service systems. This paper also tries to lay out a consistent framework for those related research activities for further research in this area.

* Internal Accession Date Only
Approved for External Publication

1. Introduction

With the emergence of Internet technology, building e-service systems[1] to serve customers over the Internet has become an important task for all companies. Whether a company can stay ahead of its competitors, to a great extent, depends on how well it is able to meet the goal of always delivering satisfying, up-to-date, and trustworthy services cost-effectively to its customers, no matter where they are, when they want them, and how they access them. While e-service systems have increased greatly in number over the past several years, meeting this goal remains a challenge to both researchers and practitioners.

The difficulties in achieving the goal mentioned above come from a wide range of complicated factors, mainly dynamism, wide distribution and interaction of both services and resources. The Internet, being a shared communication media, allows for ubiquitous services. The access pattern of the customers can be very dynamic and distributed, and will become more so, as more mobile devices are used [4]. This trend makes it very difficult for any organization to manage the resulting fluctuation in service demands and resource demands, while still being able to deliver predictable and secure services to its customers cost effectively. Moreover, with the further development and application of Internet technology, more and more new services will be composed from existing services to serve a more diverse set of clients. This trend greatly complicates the interaction among services and resources, making the management of services and resources harder. In addition, increasing geographic distribution and heterogeneity in future e-service systems will further complicate the matter.

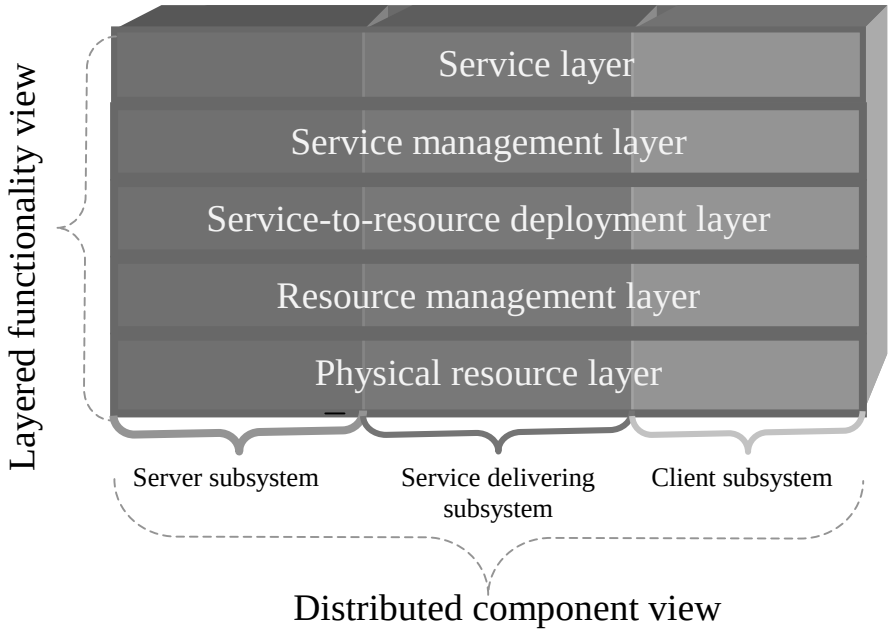

Figure 1 An abstract model of the e-service system.

While an e-service system is becoming more complicated, it is also becoming more demanding for an e-service system to reduce the development cost and the time-to-market of new services in order to keep a business staying on the top of competition. Standardization and automation are two complementary approaches to meet this demand. Even though e-service systems differ in their business logics, they fit in a common abstract model as shown in Figure 1, where an e-

[1] An e-service system is a system that provides e-services to its clients through the Internet.

service system is composed of three distinguished distributed subsystems and each subsystem has a 5-layer functional architecture. This makes it feasible to have a general development and management platform for e-service systems to standardize tools, API, and environments. Automation is the best way to help to minimize the cost of an e-service system.

The **AMSES**, **A**utomatic **M**anagement **S**ystem for **E**-service **S**ystems, is an endeavor that aims to provide a common vision to achieve the afore-mentioned goal. Under this context, this paper makes three major contributions: (1) specify the functionalities of the AMSES; (2) design four general mechanisms for support service and resource outsourcing, resource virtualization, self-management, encapsulation and protection of e-service systems; (3) derive a concrete functional architecture for the AMSES, and lay out a consistent framework for those related research activities. Overall, an AMSES strives to provide automatic management for the whole lifecycle of an e-service system by moving majority of the operational activities out of an e-service system.

In the design, we make an effort to ensure that the AMSES mechanisms are general and flexible enough to make AMSES be a general development and management platform for e-service systems. These mechanisms support diverse services and resource outsourcing, and to provide protection as well as performance guarantees. To make use of aggregated resources over the Internet, there must be intelligent ways to manage these resources and deploy services on them. We believe that autonomy via self-monitoring and self-tuning is the key to distributing control throughout the distributed services and resources, and to ensuring predictable behavior. Our AMSES architecture distributes decision-making and policy enforcement among different entities in an AMSES so that each of them is predictable and that they collectively enable predictable behavior of the services deployed on them.

The remainder of the paper is organized as follows. In Section 2, we specify the functionalities of the AMSES. In Section 3, we address protection, predictability, resource virtualization, and self-management issues by proposing four AMSES mechanisms. In Section 4, we derive the functional architecture of the AMSES based on the four AMSES mechanisms, and describe the major components of an AMSES, including logic design, auto-deployment, runtime self-management and resource management. The flexibility of the AMSES to support different e-business models is shown in Section 5. We compare our work with related work in Section 6, and conclude in Section 7 with a detailed discussion of the important research issues.

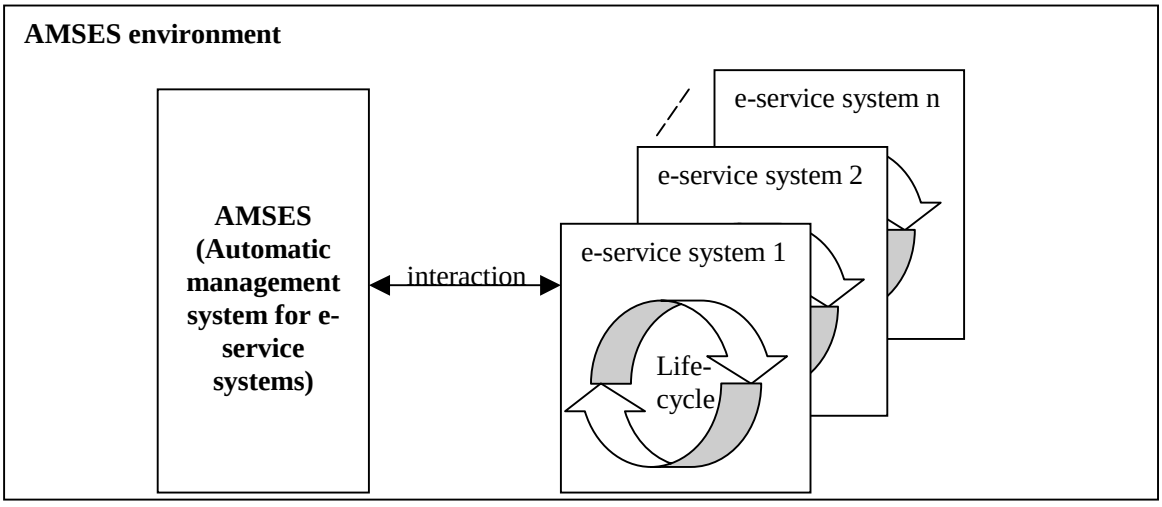

Figure 2. Relationship between the AMSES and the e-service systems it manages.

2. Functionality Specification of the AMSES

The AMSES' objective is to provide automatic support for the functions involved in the lifecycle of one or more e-service systems. Figure 2 conceptually illustrates the relationship between an AMSES and the e-service systems it manages. To clearly specify the functionality of the AMSES, we first must analyze the functions involved in each phase of the lifecycle of an e-service system. This will drive the functionality specification and architectural design of an AMSES. Based on this, we specify the AMSES functionality where the goal is to simplify the operation of an e-service system as much as possible. We also discuss the interaction between an AMSES and the systems it manages.

2.1. Functions in the Lifecycle of an E-service system

The engineering process of an e-service system goes through a typical system-engineering lifecycle. The cycle starts with an e-service system specification, followed by a logical design phase, a service deployment or implementation phase, a service delivering phase (where the e-service system is in operation), and a maintenance phase. An e-service system specification is specified by a AMSES user. An AMSES user is a service provider or the owner of an e-service system. The e-service system lifecycle and the input/output of each phase are illustrated in Figure 3. Here, we need to go a little bit deeper to understand the major functions in each phase of the e-service system lifecycle.

Logic design phase: The logic design phase is driven by its input, an e-service system specification. An e-service system specification usually specifies service demands and service level agreements (SLAs). The service demands specify what kinds of services should be provided to which types of clients. The SLAs specify service requirements (e.g., Qos, security) and system requirements (e.g., reliability, availability, and scalability (RAS)). The major functions present in this phase include:

- Determining the e-service system process, which includes the services to be provided to clients, the dataflow, the control flow, and the management flow. The process also derives resource demands based on the service demands and the SLAs. The resource demands specify resource type, capacity, cost, and performance requirements. Workflow [4][11] is usually used here.
- Building and composing services based on available services (applications), both local and remote.
- Determining a virtual execution environment. The virtual execution environment is a concrete plan for meeting the resource demands.
- Deploying services onto the virtual execution environment based on the SLAs. The SoS work [8] in HP labs addresses the problems in this space.
- Repeating this process incrementally at runtime if necessary (see the lifecycle diagram in Figure 3).

The output of this phase is a logic design that consists of service descriptors, resource descriptors, deployment of services onto resources, and system requirements. At the end, a simulator may be used to verify the logic design. One representation of the logic design is described in more detail in Section 4.2.

Service deployment phase: Given a logic design, services are deployed onto physical resources via the following four activities:

- If necessary, acquire required physical resources through resource discovery mechanisms over the Internet.

- Configure physical resources and services.
- Configure policy-enforcement mechanisms.
- Reconfigure resources and services; modify policies (if necessary).

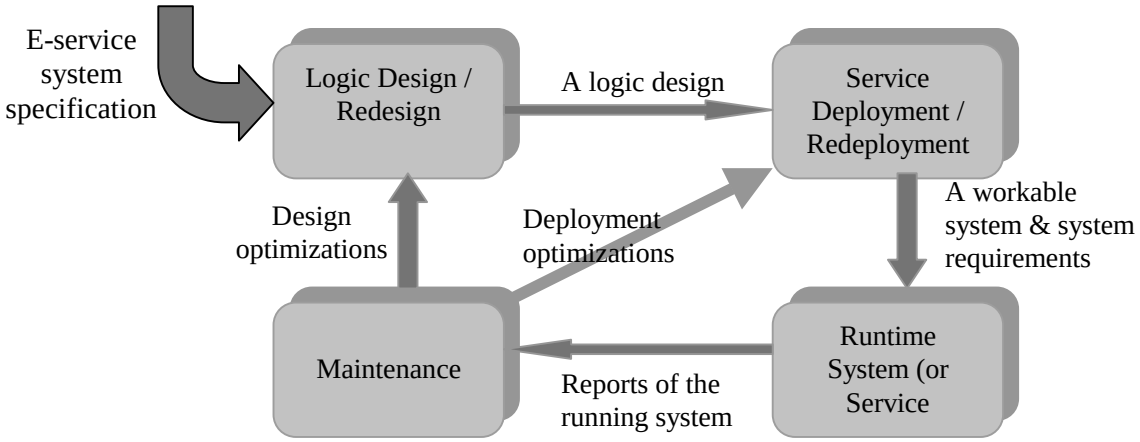

Figure 3. The typical lifecycle for an e-service system.

The output of this phase is a working e-service system ready for execution.

Runtime system (service delivering phase): With inputs of a configured e-service system and SLAs, this phase fulfills the following functions:

- Ensure QoS, security, and other requirements for the services being provided to the clients of an e-service system.
- Ensure RAS requirements for the runtime system.
- Monitor the runtime system; evaluate performance and check for system requirement violations.
- Generate performance reports and system requirement violation reports as needed. These reports are the output of this phase.

Maintenance phase:

- Conduct cost/performance analysis, policy violation analysis, and demand/supply prediction to generate optimizations for the deployment phase. These optimizations usually result in changes to policies, resource allocation, and service deployment.
- Conduct policy violation analysis to generate design optimizations for the logic design phase. These optimizations range from fine-tuning system requirements to overhauling the logic design.

2.2. Functionality of the AMSES

The ultimate goal of the AMSES is to automate the entire lifecycle of an e-service system; a service provider (i.e., the owner of the e-service system) submits a business specification, and the AMSES automatically produces an operational e-service system. To be more practical in the near future, in this design, we let an AMSES take care of all operational functions in e-business lifecycles. (We would like the AMSES to be a general platform to support all kinds of e-service systems.) Based on this design goal, the functionalities of the AMSES can be divided into two kinds: (i) functionalities that manage the operational functions and development of an e-service

system, and (ii) functionalities that manage the interactions among multiple e-service systems. This section specifies these two types of functionalities of the AMSES.

Table 1. Functionality partitioning between an AMSES and the system managed by it.

Lifecycle	Functionality Partitioning	
	AMSES	**e-service system managed by the AMSES**
Logic Design	• Management of common service and application repositories. • Management of design tools, frameworks, simulation tools, and design schemes; logic design verification. • Management of logic design.	• Conduct logic design using automated tools, frameworks, and schemes. • Derive system requirements from the e-business specification. • Conduct simulation and verification.
Deployment	• Acquire required resources and configure them. • Configure services. • Configure policy enforcement mechanisms.	N/A
Runtime system	• Ensure system requirements (for both service quality and the runtime system). • Conduct continuous monitoring. • Collect accounting data (for billing AMSES customers). • Generate performance reports and system requirement violation reports.	N/A
Maintenance	Conduct cost/performance analysis, violation analysis, and demand/supply prediction to generate deployment optimizations.	Conduct violation analysis to generate design optimizations to the logic design phase.

In order to simplify the development and operation of an e-service system as much as possible, we move operational functions of an e-service system to the AMSES while providing support for e-service system development in the AMSES. This determines the first kind of functionalities of the AMSES. Table 1 shows the partition of functionality between the AMSES and the systems it manages. We can see that most of the operational functions have been moved to the AMSES. This results in a very simple e-service system lifecycle.

Of the four phases of an e-service system lifecycle, the logic design phase is most closely related to the business logic. The logic designer needs to derive the business process from a well-defined business specification and represent it in a format that is easy to develop. How to automate this phase is beyond the scope of this document. However, the AMSES simplifies this phase by providing a set of design tools, design frameworks, and schemes for different business models.

In the maintenance phase, some business-related decision-making may be needed to resolve system requirement violations reported from the runtime system; for example, to increase investment on the e-service system in order to meet increased reliability requirements for some services. It is better to leave this decision-making in the e-service system lifecycle so that the service provider can participate.

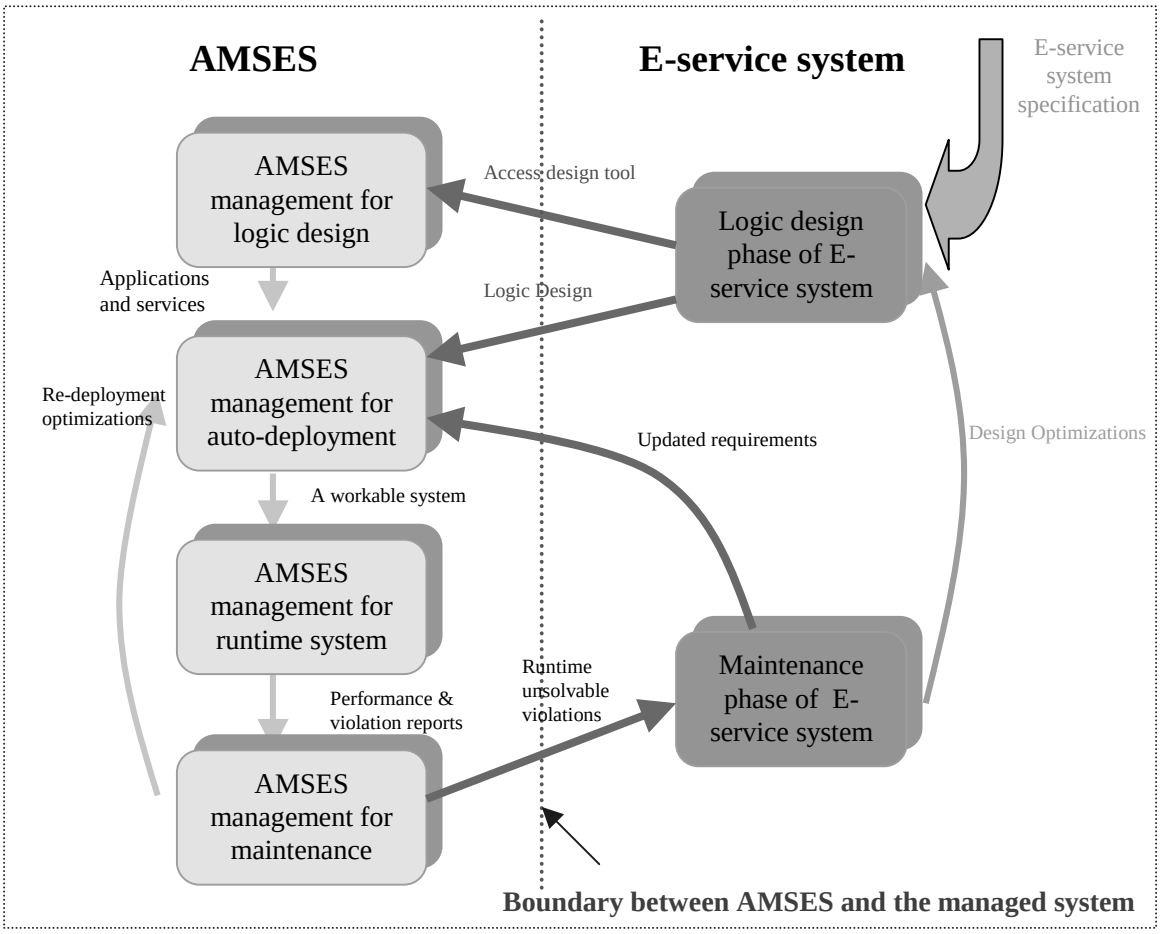

Figure 4. Interaction between the AMSES and the e-service system it manages.

The e-service systems managed by the AMSES are typically wide-area distributed systems. This characteristic determines the need for a wide-area distributed management system in the AMSES. This is consistent with another goal of the AMSES: to explore the potentially infinite resource pool that exists on the Internet. To support the security and "always-on" requirements of an e-service system, the AMSES itself must be always-on and highly secure.

Based on the functionality partitioning given in Table 1, Figure 4 shows the interaction between an AMSES and the system it manages during the e-service system lifecycle. The AMSES is pictured on the left-hand side and the simplified e-service system lifecycle is presented on the right-hand side. An e-service system conducts logic design by accessing design tools that are provided and managed by the AMSES. A logic design is produced at the end of this phase, and the logic design will be used in the AMSES deployment phase. In the maintenance phase of the e-service system, the e-service system analyzes the runtime violations that the

AMSES cannot resolve, and attempts to resolve these violations by either updating the system requirements or by redesigning the system.

Besides managing the entire lifecycle of a single e-service system, the AMSES also supports the operation of multiple E-service systems. This entails the following second kind of functionalities of the AMSES to manage the interaction among multiple e-service systems:

- A mechanism for representing, protecting, and managing each e-service system that is running in the AMSES.
- Prioritization among multiple e-service systems that are running in the AMSES, which is used for resolving resource allocation race conditions.

3. AMSES Mechanisms

The design goals for the AMSES mechanisms are: (1) The AMSES mechanisms should enable seamless transitions between the various phases of an e-service system lifecycle; (2) The AMSES mechanisms should facilitate the natural partitioning of responsibilities among the various distributed AMSES components for enforcing system requirements; (3) The AMSES run-time system should be self-adjusting (i.e., it should be adaptive to both changing application demands and resource availability); and (4) The AMSES mechanisms should provide support for the entire lifecycle of an e-service system and protect e-service systems properly.

In the rest of this section, we describe the mechanisms for resource virtualization by defining *intelligent resources* and *active resources*. We discuss how these mechanisms can help us to achieve goals (1) through (3). We also describe a *virtual operating environment* that encapsulates and protects "everything" an e-service system needs in its simplified lifecycle (see Section 2.2), including: tools and application frameworks for logic design, and tools for maintaining the e-service system (to adapt to changes in e-service system requirements or resource availability).

3.1. Resource Virtualization

Resource virtualization aims for providing an illusion of a resource with guaranteed service level agreements (SLAs) (i.e., service requirements and system requirements) to e-service systems. *i-resource* and *a-resource* are two general mechanisms that are designed to implement resource virtualization of any kind of physical resources.

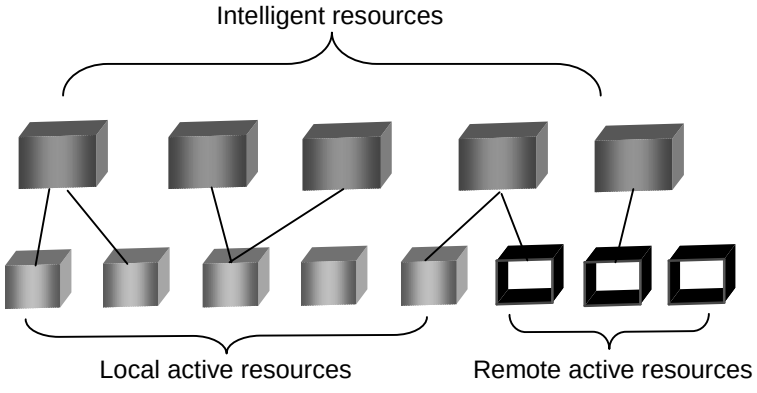

Figure 5. Active resources and intelligent resources

An i-resource facilitates the seamless transition from the logic design to the physical deployment by directly implementing the *virtual resources* that are used in a logic design (see Section 4.2). An i-resource provides guaranteed service requirements (such as QoS, security,

etc.) and system requirements (such as reliability, availability, and scalability, etc.) by intelligently acquiring and managing a-resources. The SLA serves as a contract between the demand and the supply from the e-service system. The e-service system sees the guaranteed service level agreements, but it does not see the physical resources, mechanisms and policies used to implement and enforce these guarantees. This allows an i-resource to adapt to changes in service demands and resource availability, resulting in a smooth maintenance loop. For example, an i-resource can be informed to double its capacity due to a change in the load on the e-service system, but the details of the actual acquisition and configuration of the new physical resources to meet this increase in demand are hidden. On the other hand, an i-resource could hide the failure of certain physical resources while keeping the e-service system running, uninterrupted. The intelligent resource mechanism allows transparent resource sharing, resource replication and migration. These mechanisms are the basis for reliability, availability, and scalability.

To make this scheme work, ultimately we need physical resources to do the work. *Active resources* (*a-resources*) are the software abstract of the physical resources that are configurable and controllable by an AMSES. Note that not all physical resources are directly configurable or controllable. For example, a CPU and a memory chip are not individually controllable, but a host is configurable and controllable. Ideally, an a-resource is extensible in the respect that its policies can be modified and/or replaced through the actions of an i-resource.

The illusion of providing a resource with guaranteed SLAs is cooperatively fulfilled by the i-resource and the a-resource. The i-resource has two kinds of functionalities: (1) implement system requirements (such as reliability, availability and scalability) through dynamic resource allocation, service migration and service replication; and (2) provides policy control and service requirement enforcement control to a-resources. The a-resource mainly focuses on enforcing service requirements and part of system requirements (such as reliability). An a-resource can be fully or partially used by one or more i-resources. An i-resource provides guaranteed SLAs by properly configuring and managing the a-resources it uses. The a-resources used by an i-resource do not have to be local to an AMSES. They can be acquired from other systems as long as there is a control interface that the (local) i-resource can utilize. The relationship between a-resources and i-resources is illustrated in Figure 5. In the sections that follow, we describe the i-resource and a-resource in more detail.

3.1.1. Intelligent Resource (i-resource)

An i-resource provides guaranteed capacity, quality of service, reliability, availability and scalability by automatically allocating, replicating and migrating active resources. An i-resource is illustrated in Figure 6. Logically, an i-resource includes attributes, a runtime system, and a control API. The attributes associated with an i-resource include (but are not limited to):

- A unique name in the namespace of i-resources.
- The type of the i-resource (for example, network (i-net type), storage (i-storage type), or computing power (i-server type)).
- The guaranteed capacity, performance and protection that the i-resource should provide. This serves as a contract between the e-service system and the resource runtime.
- Service descriptors that identify the services/applications (to be) deployed on the active resources used by this i-resource.
- A reference to the virtual operating environment (VOE) to which it belongs: The VOE, which will be defined in Section 3.3, is an AMSES representation of an e-service system. Because an i-resource is created for implementing a virtual resource in a logic design, it belongs to exactly one VOE. The reference to the VOE can be used by the i-resource (or any a-resources it uses) to send functional violation reports to the VOE.

- Information regarding how the i-resource can be instantiated and initialized. (This is derived from the resource auto-configuration graph in the logic design output; see Section 4.2 for details).
- Information about the a-resources it uses, how the a-resources are configured, and how the a-resources perform.
- Information about i-resources of i-net type. These implement the virtual channels that connect this i-resource to other i-resources.
- Policies for supporting system requirements (reliability, availability, and scalability). This set of policies includes the following: (i) Policies regarding how the load is partitioned among the a-resources used by an i-resource. This could range from complete load replication to load distribution; (ii) (Data or services) replication requirements and policies; (iii) Load-balancing scheme, if the load is distributed; and (iv) Policies to allow monitoring of i-resources by other i-resources.

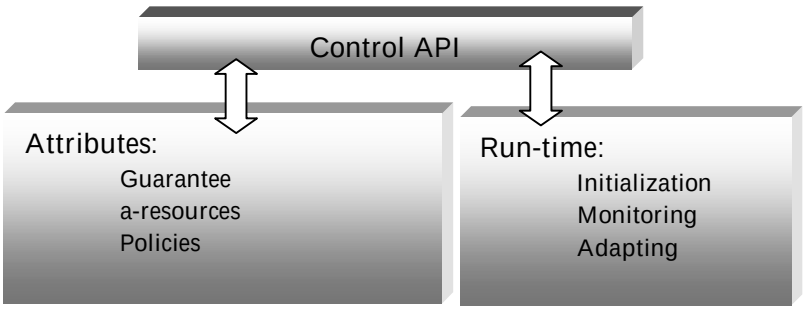

Figure 6. i-resource.

The *runtime system* engages three kinds of activities:

- Initializing the i-resource: It contacts resource management system to acquire a-resources and configures them. It deploys services onto a-resources according to the service auto-configuration information in the meta-data system. It can also trigger the initialization of other i-resources.
- Monitoring the a-resources it uses as well as other i-resources. To make the AMSES self-reliant (i.e., to make the AMSES runtime self-repairing), it is sometimes necessary to have the i-resources monitor each other. If a monitored i-resource dies, the monitoring i-resource is responsible for creating another i-resource to replace it. (Finding the optimal mutual monitoring algorithm for different i-resources is an interesting research problem and can be couched as a pure graph problem. In addition, techniques for check-pointing and restarting services over different kinds of i-resources could be interesting as well).
- Responding to changes in service demand and resource availability. This process involves decision-making regarding resource allocation, configuration, service replication and migration. This action could be initiated by the monitoring activity, by system violation events generated by a-resources used by the i-resource, or by prediction.

The control API provides the means for configuring the logic design information into an i-resource, for changing the management policies, and for querying i-resource attributes.

3.1.2. Active Resource (a-resource)

An a-resource is a software system that runs on top of a physical resource. It implements exclusive manageability of the physical resource, and has attributes, a runtime system, a service API, and a control API. The attributes associated with an a-resource are:

- *IP address*: An a-resource has at least one IP address.

- Information for measuring of the capacity of the system and the current system load.
- Policies to be enforced by the a-resource.
- Resource configuration information
- Service configuration information

The runtime system enforces SLAs according to the requirements and policies. The runtime system also includes a monitoring subsystem that tracks the behavior of the resource and updates the meta-data system as necessary. It also reports SLA violations to the i-resource that uses it. The runtime system could be implemented as several control loops.

The control API allows the *resource management system* or the administrator to query the state of the resource, acquire the resource, configure the resource, deploy services on the resources, and download or change policies that control its operation (e.g., access control policies, security enforcement policies, QoS enforcement policies, and so on).

The service API allows user identification (either by a UID or by providing a unique API or handle for each user). The API should provide feedback to the user when the requirements specified by the user can no longer be satisfied. The user of the API is either an i-resource or the AMSES administrator.

3.2. Virtual Service

A *virtual service* is a handle for accessing a service that is provided by another VOE that either belongs to the same AMSES or belongs to a different AMSES. A virtual service consists of three components: (1) *Internal attributes*, which include a *unique name* in the namespace of services, the *address* of the target service, and *handlers* for accessing the target service. (The existence of multiple handlers allows sharing of a virtual service.) It may also include access permissions and information about the state of the remote service; (2) *A runtime system* that hands off the access to the remote service, multiplexes the remote service if it is shared, and monitors the quality of the services; and (3) An API for controlling and accessing the remote service.

We distinguish between a virtual service and a local service because this distinction has implications in different phases of an e-business lifecycle. For instance, in the logic design phase, one can specify that a service should be outsourced rather than deployed locally. In this case, at runtime neither i-resources nor a-resources will be provided. However the remote service will be monitored for performance, security, and accounting reasons.

3.3. Virtual Operating Environment (VOE)

In Section 3.1, we described the notion of i-resources and a-resources, and discussed how they enable self-management, the natural and effective partitioning of responsibilities, and the smooth transitions between the various phases of an e-business lifecycle. Now, we describe the virtual operating environment.

A virtual operating environment is an AMSES representation of an e-service system. The owner of a VOE is a service provider who owns the e-service system represented by the VOE. VOE owners are the customers of an AMSES. A VOE is an object managed by the AMSES. It encapsulates and protects "everything" that an e-service system needs in its simplified lifecycle (see Section 1.1) including: tools and application frameworks for supporting the logic design, and tools for maintaining the e-service system (to adapt to changes in the e-service system requirement or resource availability). A VOE has a control API for the VOE owner to conduct all activities in the simplified e-service system lifecycle. The AMSES manages and protects the

runtime instantiation of the logic design. The protection is enforced by the i-resources and a-resources on behalf of the VOE. The AMSES maintains the mapping of the logic design onto the e-service system runtime in the AMSES, which will allow a VOE owner to query where a service is running.

A VOE consists of the following components:

- A unique name in the namespace of VOEs.
- Public entities, provided by the AMSES and shared by all VOEs running in the AMSES, that includes repositories of publicly-available design tools**,** common applications, *virtual services* which are services acquired by the AMSES to be used by all VOEs.
- Private entities, only accessible to the VOE owner (not to others, such as the AMSES administrator and other VOE owners), which include logic design (see Section 4.2), functionality violation reports, and private repositories of application, service, and virtual service.

4. Functional Architecture of the AMSES

This section derives the functional architecture that fulfills the AMSES functionalities specified in Section 1.1. This section starts with an overview of the functional architecture of the AMSES. It then describes the output of the logic design phase, auto-deployment, self-management, resource management, and the user interfaces of the AMSES.

4.1. Overview of the Functional Architecture of the AMSES

The AMSES mechanisms turn an e-service system into a fully self-adjusting system. This greatly simplifies the management task of the AMSES, resulting in a simple functional architecture for the AMSES as shown in Figure 7.

After opening an AMSES account, a client starts the e-service system design by utilizing the VOE that is created by the AMSES as part of the account. The VOE enforces privacy protection, in that all activities and data in a VOE are only accessible to the owner of the VOE. The client starts the development at the logic design phase in the VOE based on a well-defined e-service system specification. When the logic design is finished, the VOE triggers the auto-deployment process. From this point on, the system functionality will be automatically taken care of by the AMSES, including auto-deployment, runtime, resource management, and maintenance. This greatly simplifies the task of a service provider **—** one of the major goals of the AMSES.

For the administrator of an AMSES, the major responsibilities include service and resource registration and discovery, building new a-resource templates for new types of physical resources, and accounting (billing data is collected by the a-resources and sent back to a billing system).

4.2. Logic Design

The user starts the development of the e-service system with a well-defined e-service system specification. She uses the design tools such as application frameworks to produce a logic design, and may verify the design using a simulator. A logic design contains all the necessary information that an AMSES needs to conduct automatic deployment and runtime management for an e-service system. A logic design could include the following information:

- *Mapping between services and resources*: One way to represent the mapping between services and resources is via a virtual resource graph, where a node in the graph represents a *virtual resource*. A virtual resource represents the resource demands; it

specifies the type of the resource (e.g., computing or storage), capacity, cost, and performance requirements. A virtual resource also specifies how services are deployed; a virtual resource contains descriptors of the services and applications that are to be deployed on it. Each service descriptor has a service requirement, which specifies QoS and security requirements. An edge between two nodes (virtual resources) is a virtual communication channel (VCC) with an aggregated communication resource requirement. A VCC consists of a set of virtual communication sessions (VCS). A VCS connects a service on one end to a service on another end, and it also contains QoS requirements. The aggregated communication resource requirement of a VCC is derived from the communication QoS descriptions of the VCSs in the VCC.

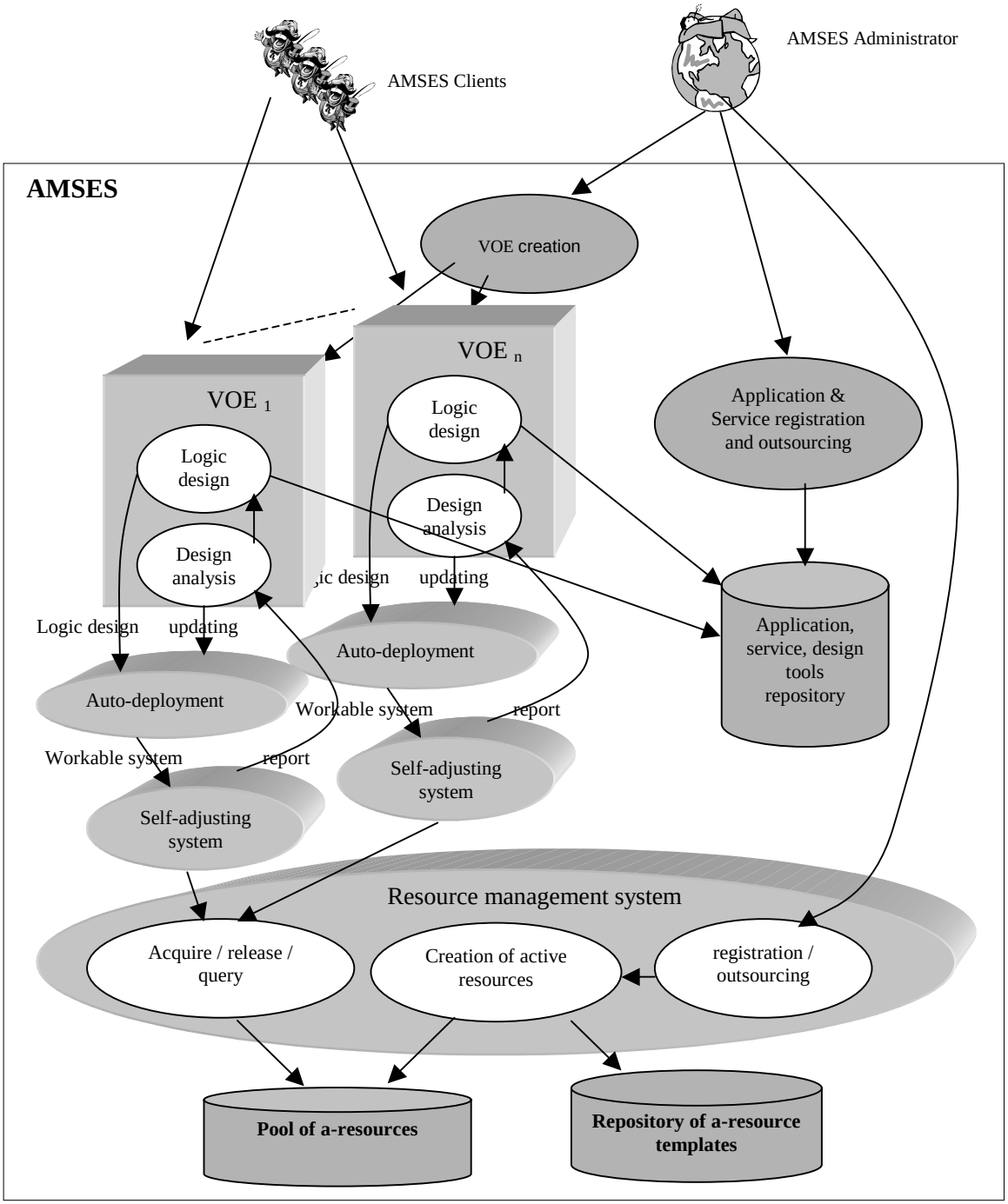

Figure 7. Functional architecture of the AMSES.

- *Resource auto-configuration information*: This should give the auto-configuration ordering among virtual resources. This can be represented as a directed graph where the directed edges give auto-configuration ordering among virtual resources. This graph has one origin, which is the starting node of the auto-configuration process. No directed cycles are allowed.
- *Service auto-configuration information*: This should give the auto-configuration ordering among services. A multi-layer directed graph can be used to represent this information, where directed edges represent auto-configuration ordering among services. The services in each virtual resource are sorted into different configuration layers, based on the configuration dependencies among services. No directed cycles are allowed.

4.3. Auto-deployment

The auto-deployment process can be triggered by either (i) logic design or (ii) requirement update requests (see Figure 4).

Triggered by logic design: After a logic design is produced, the AMSES creates an i-resource for each virtual resource and for each virtual communication channel. It configures all the information in the logic design into i-resources and maintains the mapping of virtual resources to i-resources in a persistent database. Then, it initiates the auto-configuration process by first initiating the resource configuration process, followed by the service configuration process.

During the resource auto-configuration process, an i-resource contacts the resource management system to acquire necessary a-resources based on the resource requirements. The i-resource then configures the a-resources. During the service auto-configuration process, an i-resource deploys the specified services onto a-resources. When auto-configuration is complete, a working runtime system results.

Triggered by requirement update requests: When the AMSES receives a requirement update request from a VOE, it first locates the i-resources to be updated based on the request information. Then, it sends the update request to the corresponding i-resources. When an i-resource receives an update request, it automatically conducts resource re-allocation to adjust its behavior at runtime.

4.4. Self-Management of Runtime and Maintenance

At runtime, all automatic management functionalities of the AMSES for e-service runtime systems are distributed in four major control loops. We describe these four control loops starting from the innermost control loop:

(a) The innermost control loop resides in each a-resource. Each a-resource continuously monitors itself and adjust its own behavior via congestion control, admission control, etc;

(b) The control loop encompassing the innermost control loop resides between an i-resource and its a-resources. The a-resources report system requirement violations to the i-resource, and the i-resource dynamically conducts resource allocation, service replication, service migration, or policy adjustments to repair the violations. In addition, the i-resource may also conduct some self-adjustment based on demand and supply prediction.

(c) The next control loop involves both a-resources and i-resources. It spans the auto-deployment, runtime, and maintenance phases of the e-service system. When an i-resource

fails to resolve a violation, it reports the violation to its VOE and triggers the high-level maintenance process in the VOE. The high-level maintenance process in the VOE tries to repair the violation by generating a system requirement update and feeding it into to the auto-deployment phase. This control loop does not involve redoing the logic design.

(d) The outermost control loop spans the VOE maintenance phase, VOE logic design phase, and auto-deployment phase. A design optimization is generated to trigger the logic design in the VOE when a violation is reported to maintenance process in the VOE. This will result in a new logic design.

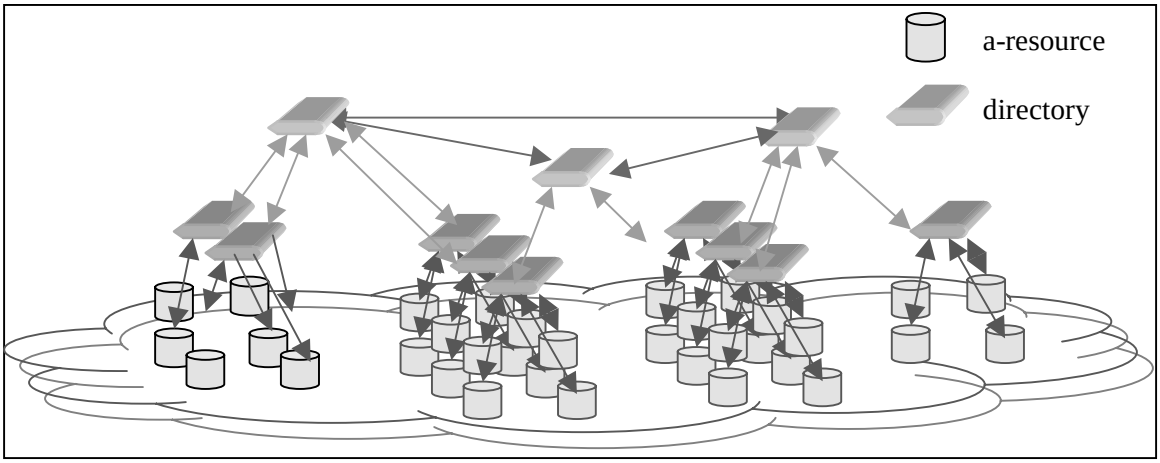

Figure 8. A federated, hierarchical organization of a-resources

4.5. Resource Management

The resource management system of the AMSES manages all a-resources owned by the AMSES in a federated hierarchical resource pool, as shown in . In the diagram, the a-resources are the leaves of the hierarchical resource pool. They are clustered into different groups based on their network domain partitioning. Network domain partitioning is mainly guided by two considerations: domain size and network controllability. Each group of a-resources are managed by one or more directory nodes. A directory node is a non-leaf node in the resource pool. Similarly, multiple directory nodes at a certain level can be managed by another directory node at a higher level. The design of the federated hierarchical structure can be affected by the lookup efficiency of different query operations; the overhead associated with collecting state information; and tradeoffs among reliability, manageability, and security in system design.

A directory is an active component that contains a set of *attributes*, a *runtime system*, and an *API*. The attributes include a unique identifier in the directory space and an a-resource map. The runtime system of a directory strives to maintain an up-to-date picture of all a-resources and the network topology underlying it. Two methods can be used here: *polling* (where a directory periodically queries the state information), and *event-triggering* with a timeout (where a directory waits for state information from a-resources for a given timeout period and does polling when the time expires). How to effectively combine both methods is an interesting research question. An a-resource can be sharable or non-sharable based on the characteristics of the physical resource. At runtime, an a-resource can be in one of the following states: free, fully-used, or partially-used. The directory API is used for a-resource monitoring and information querying.

At runtime, the resource management system provides an API for the i-resources to query, acquire, and release a-resources. Besides the a-resource pool, the resource management system also maintains a repository of a-resource templates. When a resource request cannot be satisfied, the resource management system conducts an on-line resource discovery in order to acquire resources from parties outside the AMSES (such as other AMSESs). When a new resource is acquired or registered, an a-resource is created using a template from the template repository and is put into the resource pool. In addition, an administration API is provided for the AMSES administrator to register new resources.

4.6. User Interfaces

The AMSES has different interface views, as shown in Figure 10, for the administrator and the user of the AMSES. For an AMSES user, two kinds of APIs are provided: a service API for contacting the AMSES (such as applying for public services, applications, and tools), and a VOE API for the user to do the logic design. For the AMSES administrator, a control API is provided to control the AMSES, and a service API is provided to manage user-related activities such as billing, account, maintenance and system queries, etc. The administrator "sees" the VOEs created for users as "black boxes"; the VOEs are not accessible by the administrator.

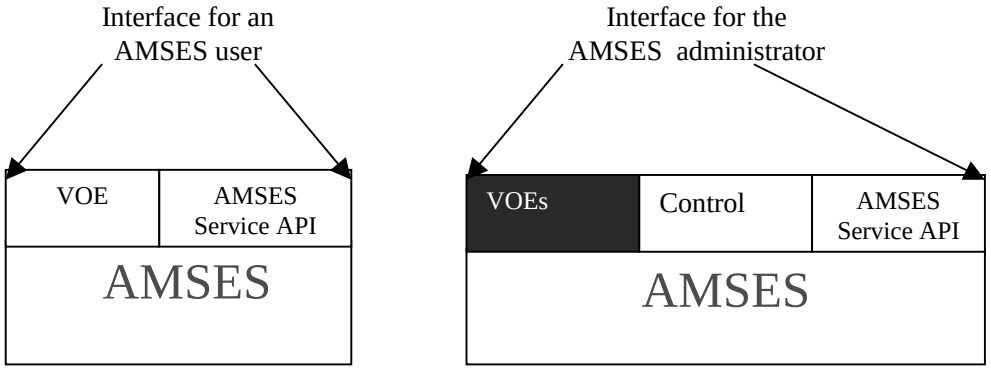

Figure 10. Interface views for the administrator and the AMSES user.

5. Usage Models Supported by the AMSES

The AMSES architecture presented in this paper is flexible and can be tailored to fit many different e-business models. We discuss some of the models below:

Single service provider: In the simplest case, a service provider can implement its e-service system using only a single VOE and utilize only resources owned by the AMSES. A service provider can also provide different services and protect them from each other by using multiple VOEs. A service provider can employ a hierarchy of VOEs, where each VOE potentially hosts a different type of service via the virtual service mechanism. A VOE hierarchy provides finer grained protection and more administrative independence than using a single VOE.

ASP hosting: This business model is naturally supported by the AMSES architecture. In an AMSES, multiple VOEs can be created to support different ASPs. In turn, the VOEs can share their physical resources.

Utility model: The AMSES architecture can also support a *utility model* such as the one proposed by J. Kubiatowicz et al [13]. The resource management system of an AMSES harvests resources that are widely distributed over the Internet into a resource pool. The i-resource mechanism can be used to provide predictable and persistent storage, computing power, or connectivity for many different users.

Pure service provider: A service provider can have no local resources at all; it can build its services using only resources that are acquired from other AMSESs.

Mixed model: One or more service providers can employ a hierarchy of AMSESs to provide aggregated services, such as e-commerce systems. Each AMSES can, in turn, have its own internal VOE hierarchy to enable flexible and independent administration and protection domains.

6. Related Work

Within HP labs, the Eos [17] is a related project to AMSES that exploits the emerging resource economy for e-service systems. AMSES aims for a general development and auto-management platform for e-service systems, a much bigger area than what the Eos targets. This paper represents an effort to derive a detailed functional architecture of an AMSES, following an application-driven principle.

There are several projects investigating issues that are related to our work [6][10][12][13].

The work that is closest to the AMSES effort is Grid computing [6]. It focuses on large-scale cooperative resource sharing to support *virtual organizations* that are built upon multiple physical organizations, such as ASPs (application service providers) and SSPs (storage service providers). The name grid is analogous to the (electric) power grid. The AMSES focuses on automatic management for e-service systems.

The IBM Océano Project [12] addresses many issues similar to those addressed in the AMSES and Grid endeavors, but in a more controlled setting, limited to a single "data center". The Océano vision is similar to a computing utility power plant.

Legion [10] is an object-oriented distributed operating system for wide-area high-performance distributed computing. It builds on existing operating systems. Legion uses an object model to represent everything in a uniform way. Based on this, a file system manages distributed resources and provides persistent storage for objects. Legion adopts an object-based security model, and supports object replication and migration.

The AMSES architecture differs from Legion in several respects (even though the AMSES architecture also follows an object-oriented principle). First, Legion focuses on distributed computing, while an AMSES focuses on e-service systems. An e-service system can be much more dynamic and complex than a typical distributed computing application. Second, Legion provides only the basic system mechanisms, while the AMSES i-resource mechanism can turn a distributed e-service system into a self-managing and self-tuning system. This will allow an e-service system to adapt to changes either in the load or in resource availability. This greatly simplifies system management. Third, besides supporting reliability and security, the AMSES i-resource mechanism also supports QoS and availability. Last, the tree-structured Legion file system may not suit the AMSES's resource management requirement, which needs to effectively handle concurrent accesses in a wide area. This limitation also applies to the tree-structured wide-area file system described by Pike et al [15].

OceanStore is an on-going project at Berkeley [13]. It is a utility infrastructure that spans the globe and provides continuous access to persistent information. It uses a highly redundant version of a randomized hierarchical distributed data structure for locating global resources. This data structure could be valuable to the design of the global resource management in the AMSES. As to self-tuning and self-management, OceanStore uses a computation-observation-optimization-computation loop, called Introspection.

Alvarez et al [1] and Brown et al [2] have also described a similar self-tuning and self-management loops. Alvarez et al attempt to conduct self-tuning and self-management over the entire lifecycle of a storage system. ISTORE [2] combines self-monitoring hardware components and an extensible software framework. The extensible software framework conducts self-management and self-tuning to adapt to environmental changes in an application-specific way. In the AMSES, the self-tuning and self-management loop is distributed among the VOEs, the i-resources, and the a-resources. For example, the intelligent resource mechanism of an AMSES can be used to provide an illusion of a predictable high-performance storage system using commodity storage devices distributed over the Internet. The monitoring and tuning activities are also distributed over the Internet.

Providing availability, reliability, and scalability for e-service systems has also been explored by using clusters of commodity workstations. Microsoft Cluster Service (MSCS) [16] extends the Windows NT operating system to support high-availability services. The execution environment is a cluster of self-contained Windows NT™ nodes that are managed within a resource management system. MSCS implements resource monitoring and migration in order to offer virtual NT servers. This work is limited to the Windows NT operating system domain. Fox et al [7] show, using two network services, that clusters of commodity workstations interconnected by a high-speed SAN can be very cost-effective in achieving scalability and availability for e-services. The AMSES addresses a more general problem: delivering scalable and always-on services in a cost-effective way using heterogeneous commodity resources over Internet.

There is much effort focused on supporting the logic design phase of an e-service system. For example, the Ninja project [9] proposes a framework for structuring scalable Internet-based applications, aiming to provide service composition, customization, and robust accessibility. Ninja uses strongly typed operators to represent service components; a composition of multiple operators is called a path. The AMSES supports service composition across an e-service system boundary via the virtual service mechanism.

Network virtualization has been addressed by Mainwaring et al [14]. They attempt to provide applications with the illusion of having a dedicated high-performance network over shared network resources. This work focuses on the programming interface, OS resource management in the context of Solaris, and network interface support for virtualization. Fairness is achieved among multiple flows through queuing and scheduling at the network interfaces. Super-networking [3] builds trusted enterprise networks out of untrusted public infrastructures. It focuses on security using tunneling. In the AMSES, the goal is to provide predictable and secure network services, which is a more challenging goal. The AMSES also focuses on global network virtualization.

7. Conclusion

On the one hand, the fluctuation in customer usage pattern can make delivering customer-satisfying services cost-effectively very difficult. On the other hand, resources available over the Internet form an infinite load that can potentially handle any fluctuation in demand. It is very important to develop intelligent systems that can take advantage of the infinite amount of resources available over the Internet to provide trusted and predictable services. This creates a great new opportunity for both researchers and businesses.

The functional architecture presented in this paper describes several distinguishing features of an AMSES. The AMSES provides a trusted and predictable platform with potentially infinite serving power for e-service systems. The AMSES architecture is general enough to support many

different e-service models. It simplifies e-service system development and operation by moving as much of the operational functionality out of the e-business lifecycle as possible. It automates the deployment, runtime, and maintenance phases, and supports the logic design phase. Service providers can focus on business specification and logic design. This will allow service providers to stay ahead of their business competition in a cost-effective way. The AMSES conducts self-management, self-tuning, and continuous monitoring via the AMSES mechanisms.

To bring the AMSES into reality, much research work is needed. We list some of the outstanding research questions below:

- *Logic design:* One interesting problem is how to effectively support logic design of an e-service system, including how to represent the design and how to develop tools to support and automate this phase. To effectively represent a logic design, we need to be able to represent the data-flow and control-flow of the e-service system, the applications and services that are used, and its demands on resources. We believe that the representation should be "wide-spectrum", in that the design starts with a high-level declarative description and evolves into a detailed and readily-deployable system. The representation should be rich enough to specify the requirements, the configuration of different kinds of resources, and the services. Much effort [4][8][11] has been expended to attempt to simplify the logic design phase.
- *Resource virtualization:* Resource virtualization includes the abstract representation of the various resources and how these abstractions are implemented. In other words, we need to show how to implement i-resources and a-resources for different kinds of physical resources such as network and storage. The questions we would like to answer include, among other things, (i) Can different resources share the same abstraction? (ii) If not, how much of the abstraction can they have in common?
- *AMSES runtime*: The AMSES runtime is an important part of the resource virtualization. An important goal is to make the AMSES runtime self-adapting and self-repairing. The research will include the design of both mechanisms and policies. It may also include developing enabling techniques such as how to effectively checkpoint, migrate, and restart services on the resources.
- *Monitoring architecture:* The runtime should provide monitoring functionality. Monitoring by itself includes quite a lot of interesting research problems. For example: (i) How can i-resources mutually monitor each other? (ii) How can an i-resource effectively monitor the a-resources it owns? (iii) How does the AMSES translate the enforcement of the service level agreements (SLAs) in the service level into the monitoring activities at the resource level?
- *Security architecture*: The existence of a security architecture is important to the success of the AMSES. Research issues in this area include authentication, authorization, and auditing, as well as how to trade off security and efficiency requirements.
- *Resource management*: The resource management system manages the pool of a-resources that an AMSES can utilize. The challenges in this area include (i) how to deal with the possible distribution of resources across different geological and trust domains; (ii) how to maintain the current state of the a-resources; (iii) how to make good performance/cost tradeoffs with regard to resource allocation and especially co-allocation of resources; (iv) how to account for resource utilization fairly and accurately. In addition, how does the AMSES ensure the "fair use" of resources? Specifically, what measures (if any) does the AMSES take to ensure that a logic design does not contain a configuration that (a) is not feasible, given the resource pool, or (b) is theoretically

feasible, but uses all of the available resources (to the detriment of the other users)? Can the AMSES protect itself against greedy or malicious users?

Acknowledgements: We would like to express our special thanks to Michele W. Chan, Amy C. Dalal, and Lance Russell for their quality-proof reading and great suggestions for polishing this paper. The virtual service mechanism was motivated by the black hole concept proposed by Lance Russell. In addition, we are extremely grateful to many other people in HPL labs for their insightful comments and constructive suggestions.

8. References

[1] G. Alvarez, K. Keeton, A. Merchant, E. Riedel, J. Wilkes, "Storage Systems Management, " Invited tutorial at *SIGMETRICS' 2000*, June, 2000.

[2] A. Brown, D. Oppenheimer, K. Keeton, R. Thomas, J. Kubiatowicz, and D. A. Patterson. "ISTORE: Introspective Storage for Data-Intensive Network Services, " *HotOS-VII*, March 1999.

[3] G. Caronni, S. Kumar, C. Schuba, and G. Scott. "Virtual Enterprise Networks: The Next Generation of Secure Enterprise Networking," ACM Annual Computer Security Applications Conference, December 2000.

[4] W. Du and A. Elmagarmid, "Workflow Management: State of the Art vs. State of the Product, " HP Technical Report HPL-97-90, 1997.

[5] M. Esler, J. Hightower, T. Anderson, and G. Borriello, "Next Century Challenges: Data-Centric Networking for invisible Computing, " *MOBICOM'99*, August 1999.

[6] I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid -- Enabling Scalable Virtual Organizations, " To be *published in International Journal of Supercomputer Applications*, 2001. (Available at http://www.globus.org/research/papers/anatomy.pdf)

[7] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier, "Cluster-based Scalable Network Services, " *SOSP' 1997*, Oct. 1997.

[8] S. Graupner, V. Kotov, H. Trinks, "A Framework for Analyzing and Organizing Complex Systems," *HP Technical Report: HPL-2001-24*, Feb. 6, 1001.

[9] S. D. Gribble, M. Welsh, R. V. Behren, E. A. Brewer, D. Culler, N, Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Joseph, R. Gummadi, J. Hill, A. Joseph, R. H. Katz, Z. M. Mao, S. Ross, B. Zhao, "The Ninja Architecture for Robust Internet-Scale Systems and Services, " To appear in *Computer Networks on Pervasive Computing*. (Available at http://ninja.cs.berkeley.edu/pubs/pubs.html, Jan, 2001)

[10] A. Grimshaw, A. Ferrari, F. Knabe, M. Humphrey, "Legion: An Operating System for Wide-Area Computing, " *IEEE Computer*, 32:5, May 1999.

[11] IBM, *IBM MQ Series Workflow Concepts and Architecture*, Printed in Denmark by IBM DanmarkA/S, 1998.

[12] IBM Research. *The Océano project*. Jan. 2001. http://www.research.ibm.com/oceanoproject/

[13] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "OceanStore: An Architecture for Global-Scale Persistent Storage, " *ASPLOS' 2000*, Nov. 2000.

[14] A. M. Mainwaring and D. E. Culler, "Design Challenges of Virtual networks: Fast, General-Purpose communication, " *PPOPP' 1999*, May 4-6, 1999.

[15] Rob Pike, et al., "Plan 9 from Bell Labs," *Bell Laboratories*, 1995.

[16] W. Vogels, D. Dumitriu, K. Birman, R. Gamache, M. Massa, R. Short, J. Vert, J. Barrera, and J. Gray, "The Design and Architecture of the Microsoft Cluster Service," *Proceedings of the 28th International Symposium on Fault-tolerant Computing*, June 1998.

[17] J. Wilkes, J. Janakiraman, P. Goldsack, L. Russell, S. Singhal, A. Thomas, "Eos, the Dawn of the Resource Economy," *HotOS'2001*, May 2001.

Biographical information about Authors

Yong Yan

Yong Yan is a researcher at Hewlett Packard Labs, where his current research mainly focuses on multimedia streaming infrastructure (CDN), resource management system, e-service system management automation and high-performance server for e-services. He has extensively published in areas of distributed system, computer architecture, system performance, parallel computing, and network. Before he joined HP, he worked in Fujitsu System Technology as a performance and computer architect. Before he came to U.S.A, he worked as an associate professor in Huazhong University of Science and Technology. Yong Yan received his Ph.D in computer science from College of William & Mary, his Master and Bachelor, both in computer science, from Huazhong University of Science and Technology. He is a member of ACM and IEEE.

Zhichen Xu

Zhichen Xu is a researcher at Hewlett Packard Labs. His research interests are in operating systems, computer security, programming languages, and computer networks. He is investigating research topics related to Internet-based service and computing. Before he joined HPL, he received his Ph.D. degree in Computer Science from the University of Wisconsin at Madison in December 2000. From 1987 to 1993, he was a research assistant in the Computer Science Department at Fudan University (P. R. China). He also worked as a Software Engineer in the New Century Software Development Center (China) during 1990 and 1991. He received his M.S. degree in Computer Science from the University of Texas - San Antonio in May 1995, his B.S. degree in Computer Science from Fudan University in July 1987. He is a member of ACM and IEEE.

Rajendra Kumar

Rajendra Kumar is an Architecture Manager at Hewlett Packard Labs, where he is engaged in research in the area of systems architecture. His current interests include IA-64 systems performance modeling; streaming, multi-media, content delivery networks; distributed resource management of heterogeneous clusters; and compressed memory systems. In the past he has contributed to development of a VLIW processor architecture which led to IA-64, large-scale multiprocessor system architectures, and optimal cache hierarchies. He has published extensively, has been granted 9 US patents, and has taught graduate classes at Santa Clara University, and UC Berkeley extension. Kumar received his Ph.D., and B.Tech. degrees, both in electrical engineering, from Indian Institute of Technology, Delhi, and has done post doctoral research at University of Waterloo, Canada. He is a member of IEEE Computer Society.