Towards Service-Centric System Organization

Vadim Kotov
Computer Systems and Technology Laboratory
HP Laboratories Palo Alto
HPL-2001-54
March 21st, 2001*

E-mail: kotov@hpl.hp.com

Internet services, e-businesses, e-commerce, e-service, core service, service management, system management

The service-centric computing, which clearly manifests itself in quickly growing Internet services, e-businesses and e-commerce, compels the service-oriented organization and management of both applications and systems. The service in this context is the central concept playing the role similar to that of an object in object-oriented programming.

There are many ways of defining services and service environments (serviceware). E-services are mostly stress the customer-oriented organization and management of applications (service discovery, brokering, negotiating, etc.). This report draws attention to the service-oriented system organization, specifically to those concepts and mechanisms of service representation, placement, and management that are appropriate at the system level. The report introduces the (core) system service as a bridge between the e-services and their execution and management at the system level. It represents a set of applications and data, which participate in implementation of the service, plus a descriptor-interface that identifies the service and its external properties, plus a wrapper that organizes and controls the execution of these applications and takes responsibility for the service external communication.

The core service concept is primarily intended for the system design and management in large-scale service systems. Being simple, core services yet catch the most basic service features needed for optimization of the global performance of such systems. It may be refined in both directions: either towards e-services or towards system-level services. Specification of the core service and main related notions (server, service request and response) are outlined. A simple metrics for monitoring service resource demands and server resource capacity is proposed.

* Internal Accession Date Only Approved for External Publication

TABLE OF CONTENTS

1. Introduction

As service-centric computing is expanding, there is a clear need to reinvent the way how the service-oriented computer systems are organized, that is, built, used, and managed. This need is specifically evident in the case of large-scale distributed systems, such as data centers, global enterprise IT environments, e-business and e-commerce systems that gradually evolve into a planetary scale unified computing infrastructure.

The large volume of both external and internal interaction and communication characterizes these systems. As result, their performance is specifically sensitive to organization of interactions, to local and global workload fluctuations, and to data traffic congestions that, in their turn, depend on how well applications and data are placed in the system. That's why a systematic, uniform system organization is crucial to meet the basic requirements to the service systems: reactivity, high availability, security, scalability, and smoothness with relation to a bursty workload.

A simple but powerful concept should be in the foundation of a new service system organization (an object in object-oriented programming and hypertext in WWW are good examples). It is quite natural to have a ***service*** as such a concept and to develop a ***service framework*** (serviceware) that allows services to be easy developed, deployed, communicate, relocated, monitored and managed. The framework will include other objects related to services: ***servers*** that host and enable services providing them with demanded resources and ***messages*** that make possible communication among services. The author came to this conclusion while working on evaluation of the architecture of global enterprise systems of large corporations (FedEx, Boeing, retail companies and banks). Considering such a system as a hierarchy of services deployed over a hierarchy of servers, we were able to build the global system models that adequately approximated the system behavior and allowed us to analyze (and improve) it [1]. The next natural step is to add these notions to the real system environment, mostly for the monitoring and management of the global system behavior. This report proposes such a step.

There are many ways of defining services and service frameworks stressing either customer-oriented or system-oriented features. The term "service" first appeared in the system context years ago as implementation of the client-server paradigm (file services, e-mail services, …). Recently, e-service became the central concept of E-speak [2], Jini [3], and many other efforts to develop high-level specifications and protocols for the e-businesses, e-commerce, and e-services.

Each end-to-end transaction follows a long path through many layers of the integrated system space that can be decomposed into a ***service stack***. The service penetrates this space both from the bottom and from the top, but it is not currently present in the middle of the space. This suggests an idea to introduce a "core" service concept and a "core" service layer into the "middle" of the service stack (above basic software and application layer) and then expand it to all layers of the service stack with common template service

mechanisms instantiated appropriately for each layer. That means that we aim at the (total) service-centric system organization, specifically at those principles and mechanisms of services representation, placement, and management that are common for all service stack layers. Some general aspects of such an organization are considered also in [4] for the case of (virtual) data centers.

Reports [5] and [6] discuss e-services as the realization of federated and dynamic e-business components in the Internet environment. Figure 1a borrowed from [5] shows a very general view of the service stack.

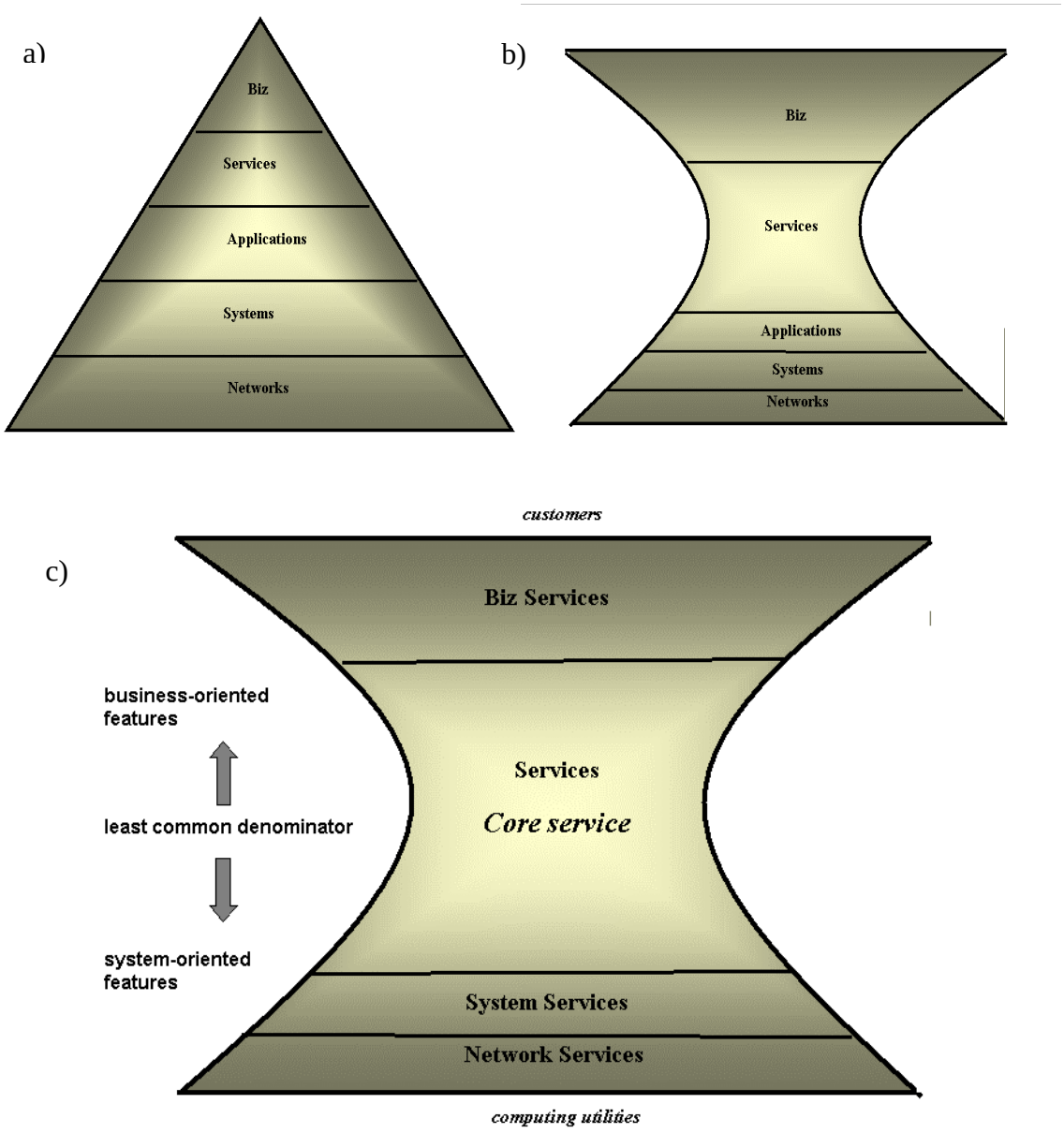

Figure 1. Service stack (a) , service-centric view (b), and total service-centric organization (c).

Figure 1a shows the penetration of the service concept into the system space from the top, from the customer side. We want to make service the central notion in this space and expand it further down to the application layer. Hence, we redraw Figure 1a in such a way that service is in the center of the IT universe, as it shown in Figure 1b. The corseted shape is chosen to stress the central position of services and also to emphasize the fact that the service layer should represent a simple, kernel-type mechanism that bridges the huge number and variety of customers' transactions and the huge amount of distributed computing resources.

The layer of services is not flat. Inside this layer, there are many different manifestations of services at different levels of consideration and for different classes of e-businesses and systems. E-services are typically defined and studied in the context of service-driven ***application integration***. We are interested in service-driven ***system integration,*** in an open architecture of large-scale systems that are built and work as federations of collaborating servers.

In this report, we get inside the service layer and stratify it further. We introduce a ***core system service*** (and related notions) that is placed in the gorge of the service layer and serves as the kernel concept of ***service programming*** (playing in the system organization the same basic role that message plays in the network organization or object in the object-orienting programming). Features related either to business logic or to system concerns may augment the core service. In the first case, we are moving up in the service space in Figure 1b towards e-business organization; in the second case, we are moving down towards systems and service implementation issues. The choice of the proper level depends on objectives of a system design. The higher level, the easier system management problems are solved. The lower level services are justified if the performance optimization is the goal.

As it was mentioned above, the term "service" first appeared in the system context as implementation of the client-server paradigm in which software is split between server tasks and client tasks. File services, SQL services, e-mail services, DNS are works performed by (program) servers for clients often over a network. There is similarity between core services, as they are introduced in this report, and those "traditional" system services. This suggests an idea to extend the service concept over all layers of the computing space, as shown in the Figure 1c. The result of this extension will be total service-centric organization of the whole system space with similar common service mechanisms at all layers and specific modifications at each layer. It makes the system space more uniform and modular; the interfaces and protocols simpler; the communication more natural; scaling and management easier

This report makes an initial step in this direction. In Section 2, the service layer is discussed in more detail. Section 3 introduces the core system service, server, service request and response. These notions are in the focus of the report. Sections 4 and 5 describe specifications of the service and server environments. In section 6 we propose a special simple metrics to measure service demands to servers capacities that is required by the service management (Section 7) to administer the deployment of services among

severs, to monitor and control utilization of services and servers, and for other performance management purposes. Section 8 combines the service, server and management environments into a service framework. Simple examples of specification of services, requests and responses are given in Section 9.

2. Inside Service Layer

Thus, we split the service layer from Figure 1b into at least two layers, as shown in Figure 2. The upper layer neighbors the e-business layer above. It is influenced by the e-business logic, e-business documents and business-to-business or business-to-consumer electronic exchange patterns. For customers, services are businesses providing values to them. Customers look for services, ask specific questions, fill forms, think on the content of the proposed service, and may negotiate with the service provider. These are ***e-services.*** They have content and semantics that are not directly related to programs, computers, and computing.

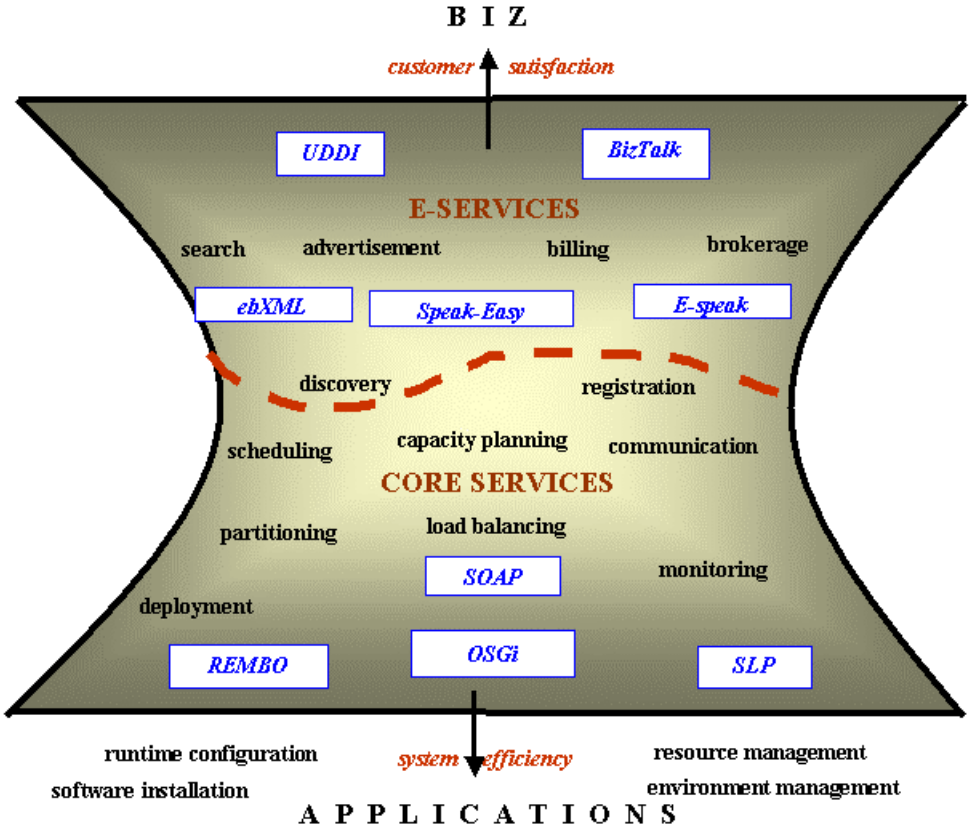

Figure 2. Refined service layer.

UDDI (Universal Description, Discovery and Integration) [7] was created as a specification for distributed Web-based information registries of customer services in order to standardize the way information about the customer services is published and discovered. In UDDI, the emphasis is on the description of e-services and service providers, that is, on the customer service content.

BizTalk [8], ebXML [9], Speak-Easy [10] are examples of well-developed protocols and environments to describe and manage this type of services. In E-speak [2], services are neutral to the customer services content, but e-speak introduces procedures that regulate interactions between clients and services, such as service search, brokerage, negotiation, billing, etc. Thus, E-speak services are abstractions of the customer services; they mostly address the templates of interaction between customers and service providers. Another area of the e-service concerns is integration of applications into e-service packages and service composition.

At the system level, we need to depart from the customer context and switch to the context of Internet-based network computing where main questions are how to organize, deploy and manage e-services, that is, the applications and data that participate in delivering e-services, in an efficient way. This context includes layers of applications, software and networks, hierarchies of computing systems, distributed functionality, etc.

However we want to fix a relatively high system level at which we want to deploy, monitor and manage services as self-containing system objects. We introduce, at this level, ***a core system service*** (that we will often refer to just as to service in the context of this report). It denotes and encapsulates a set of computing activities that are bundled to fulfill a service task. A service is implemented as a package of applications and data, which actively cooperate with each other while working on a service task.

The main objective to introduce core system services is to reduce the complexity of the management of large-scale service-centric systems by raising the level of the managed objects both above the granularity of individual applications and individual computers and above implementation level. (Problems often can be just eliminated by moving them from lower to higher levels.) Because of the scale, this concept is of higher level of abstraction than typical current system level objects (such as files, programs, etc.). It is simple to understand, implement and use it compared to complete, end-to-end service protocols and approaches like those based on Common Information Model (CIM) [11] that cover all aspects of information processing (systems, software, users, networks and more) in one global framework and in any desirable level of detail.

The core services layer faces the application layer beneath it. It delegates to that layer the implementation of the application details, specifically those details that are related to the low-level, smaller-granularity resource management, such as software installation and resource allocation. A number of approaches and tools are under development currently, such as Linux distribution systems, REMBO server [12], Service Location Protocol (SLP) [13] and many others.

Thus, a core system service serves as a bridge between e-services and lower application and system levels. That is why it should be very simple and generic with relation to all kinds of e-services. It may be referred to as the ***least common denominator*** of all services. The simplicity of the core service framework is a fundamental requirement, as its introduction into the service stack should not add performance and other problems.

An interpretation of a service as a self-contained component, accessible via a defined service interface, is close to that of service in OSGi (Open Services Gateway Initiative) [14] that standardizes specifications for delivery services over networks to devices at homes and small businesses. (We consider the wider context of data centers, large-scale e-commerce and e-business systems.)

The core services should be maximally location independent, platform independent, and even relevant to changes in the constituent applications. For example, we can replace some application in a service implementation by another equivalent application without any changes in the service external appearance and behavior.

We also need to characterize the core services by complexity of their execution, required computing resources, expected and actual service time and response time. We need procedures for service deployment, protocols for communicating with services and between services, and other high-level system attributes and features.

3. Services and Servers

A ***core system service*** is introduced as a system notion of a higher granularity than an application program. From the system side of view, the service is just

- a collection of ***applications*** and ***data***, which participate in implementation of an e-service, plus
- ***descriptor***-facade that describes the service, plus
- ***wrapper***, the service control point that organizes the execution of these applications and takes responsibility for the external communication.

A service is triggered by some ***service request*** and results in ***service response*** forwarded back to the request originators.[*] Customer services are primary (but not the only) originators of the service requests. Services generate secondary requests to other application services if they need help in executing their requests. Thus, services interact both with customers and with each other by sending requests and responses. This interaction creates data traffic that is the primary contributor to the quality of service in utility computing.

A service may be simple or aggregated, consisting of ***subservices***. It is assumed that a "parent" service depends on some (or all) of its (direct) subservices to accomplish its task

[*] We do not consider other possible messaging paradigms here.

and is able to communicate with all of its direct subservices. The possible communication between peer services should be explicitly indicated.

A ***server*** is a virtual execution environment (a computer, a computer cluster, a storage, a database, etc.) that can host a service, that is, can enable the service activities. A server may host many services. A service may be replicated and deployed over several servers. A deployed service may be either active or inert.

Servers are usually aggregated into clusters of servers, tiers, data centers, up to networks of data centers, forming a ***server environment***.

When services are deployed on servers, a service environment is mapped onto a server environment. The service placement may be good, resulting in a smooth flow of the request and response traffic to and from services and servers, and it may be bad, creating congestions and bottlenecks that in the end cause delays in end-to-end responses of services.

We may also regard services, servers and other related notions introduced here as forming one more level, a ***service level***, on the top of the networking application level. We will use XML [15] as this level standard definition and interaction notation. Assuming that the services interact using XML (or one of the XML dialects), there is yet more to define: the service level structures and protocols are needed to be agreed upon. Protocols for interoperation in different e-service areas exist. However, these protocols are disparate and have relatively high implementation costs while we need a simpler format, easy to use and understand. The solution resides in building our service level protocol on the top of one simple, lightweight protocol: a protocol like the Simple Object Access Protocol (SOAP) [16] that is flexible, platform neutral, and XML-based.

4. Service Environment

In this section we describe possible formats of specifications for services, requests and responses. These are just sketchy drafts illustrating the service idea, rather than a proposal for a specification standard and protocols.

4.1. Service

The service specification consists of the service descriptor and the service wrapper. Templates of both descriptor and wrapper are stored as a ***service template*** in some ***service repository*** of the service environment (or of some domain of the environment). When a service is installed at a server, a ***service instance*** is created using its template from the repository.

4.1.1. Service Descriptor

The ***service descriptor*** includes the following elements:*

- service name that identifies it as a representative of its kind,
- service URL (that includes as its prefix the URL of the hosting server),
- reference to the parent service,
- references to direct subservices (possibly with a condition or probability of using subservices),
- names of peer services whose help may be requested (possibly with a condition or probability of each request),
- required resources (in service shares, see section on service metrics),
- used resources (of the hosting server, in service shares),
- current service status (passive, active, broken, …) and management handles,
- SLA, access and security attributes,
- hosting server URL.

There might be other components of the service descriptor related to special requirements of the service to hosting servers or to priorities or to other things involved into the service specifics in the service framework. They may have also special attributes that allow request originators to compare different instances of the same service and similar properties. We are not discussing them here.

The service descriptor is decoupled from its implementation that consists of the service wrapper and participating applications and data.

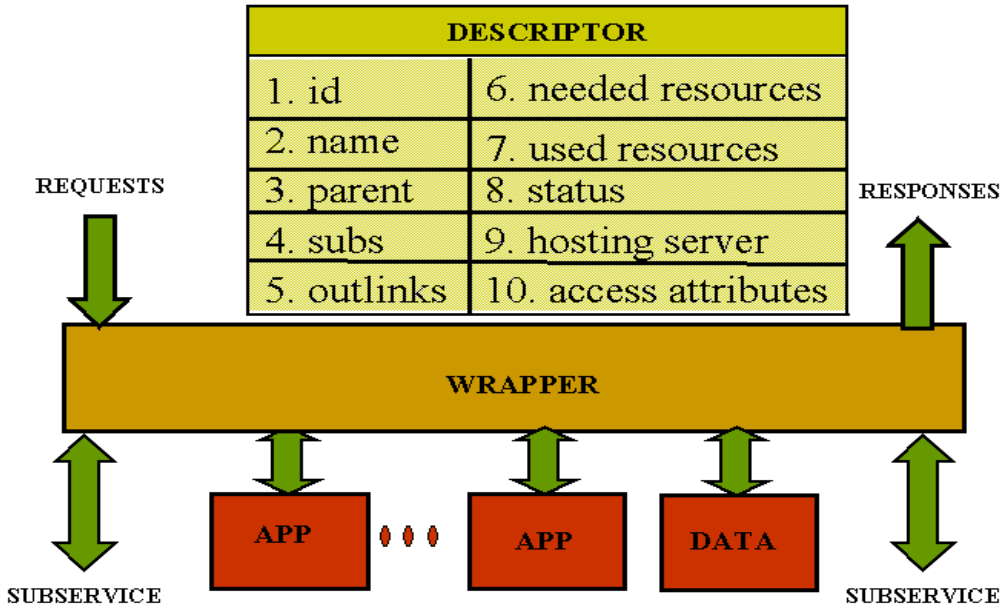

Figure 3. Service specification.

* Some fields may be optional.

4.1.2. Service Wrapper

The service's ***participating applications*** execute those subtasks that allow the service to accomplish its overall task.

The ***service wrapper*** is the service control point that organizes, monitors and controls the execution of the service's participating applications and its communication with the outside world, in particular with request originators and other services of the service environment to which this service belongs. The wrapper

- organizes and controls the deployment/removing of the service on a server,
- enforces access agreements, accepts requests for the service and analyzes them,
- receives requests to the service and decodes its headers (see the next section),
- processes the requests, that is, decodes their bodies and initiates and controls the execution of the service participating applications according to the information contained in the bodies,
- generates secondary requests to subservices and other services,
- generates responses to requests to this service,
- provides monitoring agents with information on the service status and used resources,
- executes control commands of the service managers,
- collects statistics.

Developers can provide multiple implementations to the same service descriptor. Developers who use a service can code against that service's descriptor without regard to its implementation. The Java ™ platform independence and dynamic code-loading capability makes it the most preferable candidate for the wrapper implementation.

4.2. Service Request

The structure of service requests follows the common message format used in SOAP: an envelope, which contains a header and a body section carrying the request content.

The ***request header*** includes data about:

- request identifier,
- originator URL,
- sender URL,
- requested service URL,
- timestamps of the request originator and intermediate senders,
- body length,
- request QoS parameter (e.g. the minimal response delay).

The request body contains specific information that service needs to be provided to fulfill the request. It follows the SOAP body format.

4.3. Service Response

The structure of service responses also follows the common message format used in SOAP: an envelope, which contains a body section carrying the response content, and a header section.

The ***response header*** includes:

- response identifier,
- responding service URL,
- destination server URL,
- response timestamps,
- response's body length.

5. Server Environment

A server is an enabler of services. It provides resources for services in terms that services demands and hides underlying details that are not of interest to services. A server represents the lower levels of the service stack, virtualizing them for core services.

A ***server template*** is stored as a template of the server descriptor in some ***server repository*** of the server environment (or of some domain of the environment). When a server is installed at a computer (or a cluster), a ***server instance*** is created using its the server template from the repository.

The server specification includes the following elements:

- the server name,
- the server URL,
- parent server,
- subservers,
- constraints on hosted services,
- hosted services,
- currently active services,
- maximal available resources (in service shares, see section on service metrics),
- currently available resources (in service shares),
- server status (passive, active, broken, …) and management handles.

6. Service Metrics

A service environment creates demands to the resources of a server environment that hosts it. The latter offers resource capacities to the service environment demands. We need a simple metrics that allow us to characterize the service requirements to server resources and to evaluate the server capability to meet these requirements.

We may measure the service complexity in various ways, for example, by required CPU time, or storage space, or network bandwidth, or their combination. We can measure the

server capabilities in similar terms. However, to simplify the design and control of e-service systems, we use a divide and conquer strategy and do not mix the service level objects with that of the service implementation (application and platform) level. That means that we do not refine services and servers into the platform components and platform parameters and will not use CPU cycles and Mbytes and Mb/sec. Instead, we want to introduce some *external, observable characterization* of the service complexity and server capacity. This will be the number of ***service shares*** either required to fulfill a request to a service (or some number of requests per a time unit) or available in a server to execute the incoming service requests. The service share combines both time (e.g. CPU time) and space (e.g. storage space) requirements, as well as requirements to communication parameters. It is supposed that conversion of time-space metrics into service shares is made by a server developer (either by using some capacity planning techniques or just by profiling services).

The ***required service shares*** measure a service complexity. A server capability to provide services is measured by ***available service shares***. The latter may be the maximal server capacity provided by a server or its current available capacity: when a service that requires N service shares runs on a server with M available service shares, then the server is left with $L = (M – N)$ currently available service shares for other request. If a request arrived for a service with the required service shares larger that L than this service should wait until L becomes larger than this number.

Both required and available service shares are defined by profiling the services of a given service environment on the servers of a given server environment. This can be made in the following way.

Let a service environment has n (types of) services and a server environment has m (types of) servers. For each pair (i,j), where $i < n$ and $j < m$, we find (or estimate) the maximal number k_{ij} of request to the i-th service that can be executed at the j-th server simultaneously (during some time unit). Then we can introduce service shares in the following way:

- for each server j, take the maximal number $k_{max,j}$ as the maximal number of shares available at this server,
- use this number as a normalizer to define the number $r_{ij} = k_{max,j} / k_{ij}$ of the service shares requested by the i-th service at the j-th server.

We can simplify the metrics by ignoring the service dependences and accepting “the worst case scenario” when the number of requested shares for each service hosted by the j-th server is the same for all services hosted by this server and is equal to the maximal number among r_{ij} where $j < m$. Such a simplification is justified when services not differ much in requested resources.

7. Service Management Environment

The ***service management environment*** maps services onto servers, deploys the services, monitors both the services and servers, and re-deploys the services to improve the quality of service, utilization or other properties, if changes in environments or in workload occur. It also takes care about registration and discovery of services and servers, as well as about communication between services, using help of the management of both higher and lower service levels.

The management environment is not a part of the proposed service specification. It interacts with services and servers via the management handles in their descriptors. It may contain some registries for service and server names, URLs, statistics databases and other features. A service management environment for distributed service systems is now under development and is the subject of a separate consideration. Its focus is on distributing services among servers and on an efficient utilization of servers. The task is to provide a smart deployment of services based on the analysis of the data traffic in service systems and its fluctuations. The analysis is made using static and dynamic information on

- server environment topology,
- service environment topology,
- request/response workload,
- service and server utilization.

This information is collected by agents and processed by managers that make use of the system models and optimization methods to find adequate management solutions.

Among other important issues, with which a service framework deals, are service naming and authorization, service location, security, availability, and access containment. They are not discussed here, as the focus of this report is just on promoting of the core service as a basic system notion. Research is currently in progress in these areas and some time is needed to extract generic solutions for the services. (There may be no universal solutions for some issues that will be valid for all service layers and types of services.)

8. Service Framework

A ***service framework*** includes a service environment, a server environment and a service management environment. Figure 4 shows how the service framework and its environments, as well as virtualization mechanisms in each environment that provide interfaces to lower (application and system) layers. The relation between the core service layer and the lower application layer is shown in Figure 5.

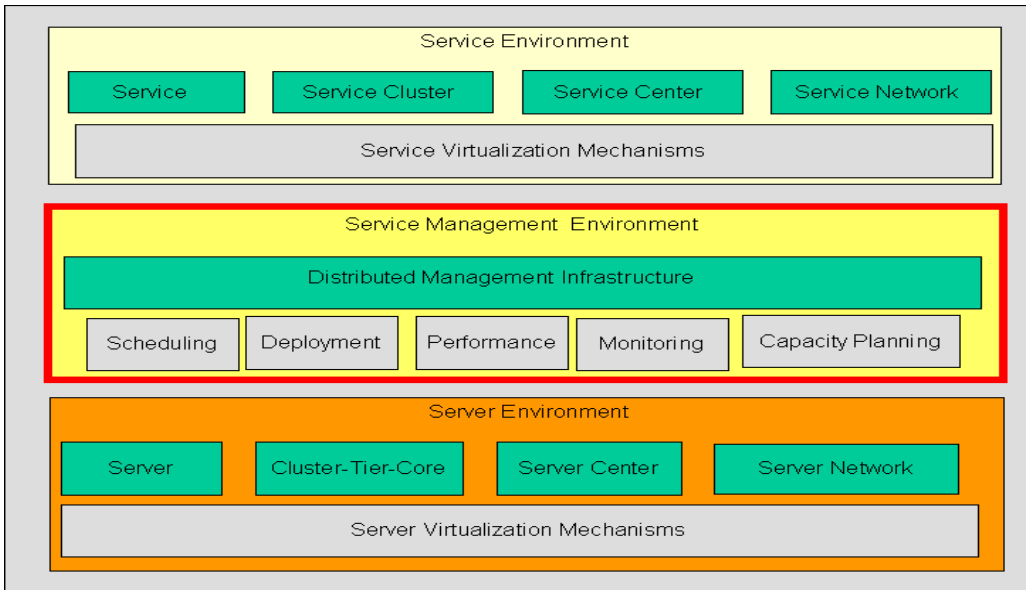

Figure 4. Service framework.

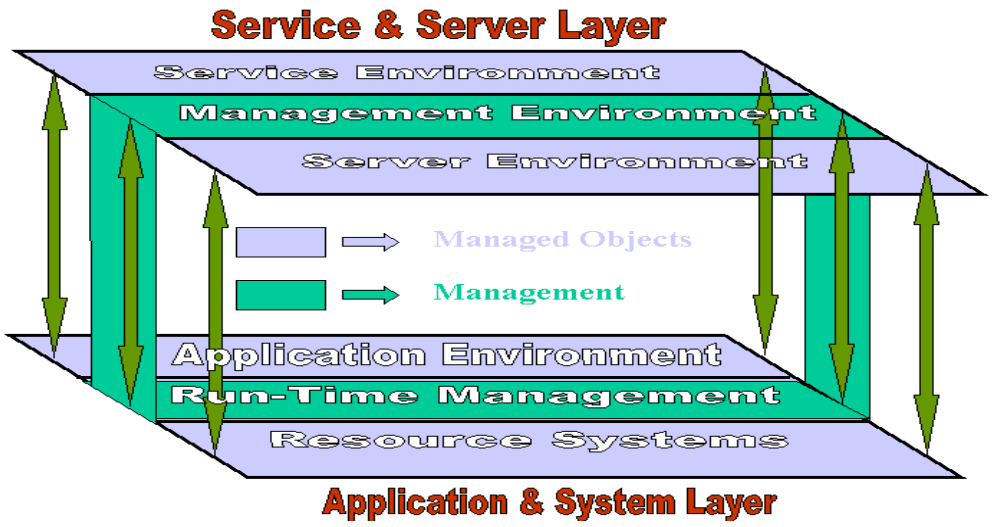

Figure 5. Service and application layer frameworks.

9. Examples of Specifications

The simple examples of service specifications in this section are services that implement e-services related to the stock market analysis. Though we use the service names that sound like these are e-services, they are not; they are core system services. Investors send requests to an e-service *MyPortfolio* to get the end-of-day summary of the performance of stocks in their portfolios. This consumer e-service is actually provided by a core service with the same name *MyPortfolio*.

9.1. Services

The service *MyPortfolio* communicates with other two core services *Quotes* and *History* to receive from them daily stock quotes and historical quotes, respectively.

The service *MyPortfolio* encapsulates several application programs:

- *Charts* that draws the charts of portfolio stocks
- *BreakUpDown* that selects the stocks moving over or under the 200-days moving average
- *PriceVolumeMovers* that identifies the stocks with the highest product *v* * *p* where *v* is percent of the volume increase and *p* is percent of the price increase/decrease.

When the service *MyPortfolio* receives a request from an investor, it finds his portfolio and then forms a request to services *Quotes* and *History* that provides the first service with the data that is necessary to form the portfolio summary, including charts and alerts. After *MyPortfolio* receives responses from *Quotes* and *History*, it prepares the portfolio summary, and transfers it as the body of a response of *MyPortfolio* to a customer.

The service *MyPortfolio* descriptor may look as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<service-descriptor>
        <name> MyPortfolio </name>
        <talks-to> Quotes History </talks-to>
        <required-shares> 1 </required-shares>
        <used-shares> 0 </used-shares>
        <status> passive </status>
        <server addr="http://tuba.hpl.hp.com/servers/X138" />
        <template addr="http://sos.hpl.hp.com/service-repository/MyPortfolio"/>
        <url  addr="http://tuba.hpl.hp.com/servers/X138/services/MyPortfolio3238" />
        <server> http://www.portfolios.com </server>
</ service-descriptor>
```

Similarly, the descriptors of the services *Quotes* and *History* may look as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
< service-descriptor>
        <name> Quotes </name>
        <talks-to> MyPortfolio </talks-to>
        <required-shares> 1 </required-shares>
        <used-shares> 0 </used-shares>
        <status> passive </status>
        <server addr="http://www.interquot.com/servers/QS" />
        <template addr="http://www.interquot.com/service-repository/Quotes" />
        <url  addr="http:// www.interquot.com/servers/QS/services/Quotes1234" />
```

</ service-descriptor>

and

```
<?xml version="1.0" encoding="UTF-8" ?>
< service-descriptor>
        <name> History </name>
        <talks-to> MyPortfolio </talks-to>
        <required-shares> 1 </required-shares>
        <used-shares> 0 </used-shares>
        <status> passive </status>
  ‘     <server addr="http://www.histquot.com/servers/HQ" />
         <template addr="http://www.histquot.com/ service-repository/History" />
        <url  addr="http:// www.histquot.com/servers/HQ/services/History22" />
        </ service-descriptor>
```

The wrapper of *MyPortfolio* represents a script type program written in any language of choice of the service developer. The wrapper is activated when a request arrives to the service. It changes the service status to active and the number of used service shares to 1. It reads the request header and memorizes the request originator and his URL. Then it extracts the list of stocks from the request body and generates secondary requests to the services *Quotes* and *History*.

9.2. Service Requests

A typical request to *MyPortfolio* has the following SOAP envelope body format:

```
<SOAP-ENV:Body>
<req:ServiceRequest xmlns:req=” kotov&hpl.hp.com-
      tuba.hpl.hp.com/servers/X138/services/MyPortfolio3238-989611052502”/>
   <request_header>
        <originator addr="kotov&hpl.hp.com" />
        <from addr="kotov&hpl.hp.com" />
        <to addr="http://tuba.hpl.hp.com/servers/X138/ services/MyPortfolio3238" />
        <timestamps generation="5/11/01 12:57 PM" />
        <body-length>100</body-length>
        <type>REQ</type>
   </request_header>
   <request_body>
      <stocks>
        <stock symb=”A” />
        <stock symb=”AOL” />
        <stock symb=”BEAS” />
        <stock symb=”HWP” />
```

```
        <stock symb=”QCOM” />
      </stocks>
  </request_body>
</req:ServiceRequest>
```

The *MyPortfolio*’s secondary requests to the service *Quotes* has the following format (the secondary request to *History* looks similarly):

```
<SOAP-ENV:Body>
<req:ServiceRequest
xmlns:req=”tuba.hpl.hp.com/servers/X138/services/MyPortfolio3238-
 www.interquot.com/servers/QS/services/Quotes1234-989653001435” >
        <request-header>
        <originator addr="kotov&hpl.hp.com" />
        <from addr="http://tuba.hpl.hp.com/servers/X138/ services/MyPortfolio3238"
        />
        <to addr="http:// www.interquot.com/servers/QS/services/Quotes1234" />
        <timestamps generation="5/11/01 12:57 PM" />
        <length>100</length>
        <type>REQ</type>
        <body-length>   100 </body-length>
        </request-header>
        <request-body>
                <stocks>
                        <stock symb=”A” />
                        <stock symb=”AOL” />
                        <stock symb=”BEAS” />
                        <stock symb=”HWP” />
                        <stock symb=”QCOM” />
                </stocks>
        </request-body>
</req:ServiceRequest>
<SOAP-ENV:Body>
```

The service *Quotes* is activated (and changes its status to *active* and the number of used service shares to 1) when a request from *MyPortfolio* arrives. It finds the end-of-day quotes for the stocks listed in the request body and forms a response to *MyPortfolio.*

9.3. Service Responses

A response from *Quotes* to *MyPortfolio* has the following format:

```
<SOAP-ENV:Body>
<resp:ServiceResponse
xmlns:resp=”www.interquot.com/servers/QS/services/Quotes1234-
```

```
tuba.hpl.hp.com/servers/X138/services/MyPortfolio3238--989653010342” >
        <response-header>
                <service> Quotes </service>
                <originator addr="kotov&hpl.hp.com" />
                <from addr=" www.interquot.com/servers/QS/services/Quotes3238" />
                <to addr="http://tuba.hpl.hp.com/servers/X138/
                services/MyPortfolio3238" />
                <timestamps generation="5/11/01 12:57 PM" />
                <body_length>526</body_length>
                <type>RESP</type>
        </response-header>
        < response-body>
                <quotes>
                        <quote ticker=”A”
                                last=”53.25”
                                high=”55.75”
                                low=”52.375”
                                open=”54.25”
                                volume=”1630000” />

                        <quote
                                ticker=”AOL”
                                last=”40.0”
                                high=”41.125”
                                low=”38.5”
                                open=”40.75”
                                volume=”12300000”  />
                        .
                        .
                        .
                </quotes>
        </ response-body>
</resp:ServiceResponse>
<SOAP-ENV:Body>
```

The similar response with a larger body will be generated by the *History* service.

10. Conclusion

The main position statements of this report are the following:

- The more computer systems become service-oriented in usage, the more their organization should be service-centric and use services and related notions as basic building blocks.
- The world of e-services is not flat. The e-service stack may be refined into layers and domains that deal with different types of problems and businesses.

- There exists a “kernel” layer, the least common denominator of e-service abstractions. This report attempts to identify it and refers to it as core system services.
- The core system service is the basic concept of the service-centric system organization. It may be refined by adding either business-oriented or system-oriented service features.
- This kernel level seems to be an appropriate starting point for the development of the service management environment in large-scale service systems, as it is simple, yet catches the most important service features needed for the system management and performance optimization.
- To manage services at this level, we need a simple metrics to measure and control the server and network utilization. It is proposed that such a metrics contains just one number (the number of service shares) that measures both resources required by services and resources available at servers.
- There are some important issues of the service organization that are not covered by this report. They will be addressed after the basic assumption are validated and refined.
- Service-centric system organization will lead to a radical reduction of cost, time and risk of delivering and maintenance of large-scale fully integrated service systems, hence to tremendous value-added opportunities for system providers.

11. Acknowledgements

The acknowledgements go to Sven Graupner and Holger Trinks. Discussions with them helped to clarify (to myself, in the first instance) some basic concepts related to core services. Sven also suggested some terminological improvements that were readily used in the report.

12. References

1. Vadim Kotov. Facilitating Analysis and Optimization of Large-Scale Enterprise Systems. Journal of Parallel and Distributed Computer Practices, to be published.

2. E-speak. Architectural Specification. Release A.0. Hewlett-Packard, 2000. http://www.e-speak.net/library/pdfs/a.0/Architecture.pdf.

3. Edwards, W.K. Core Jini. Prentice Hall, 1999.

4. Vadim Kotov. Virtual Data Centers and Their Operating Environments. HP Laboratories Technical Report HPL-2001-44, 2001.

5. Vijay Machiraju, Mohamed Dekhil, Martin Griss, and Klaus Wurster. E-service Management Requirements. HP Laboratories Technical Report HPL-2000-60, 2000.

6. Akhil Sahai, Vijay Machiraju, and Klaus Wurster. Managing Next Generation E-Services. HP Laboratories Technical Report HPL-2000-120, 2000.

7. UDDI Technical White Paper. September 2000.
http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf.

8. BizTalk Framework 2.0 Draft: Document and Message Specification. Microsoft Corporation. June 2000. http://msdn.microsoft.com/xml/articles/biztalk/biztalkfwv2draft.asp.

9. ebXML Technical Architecture Specification. http://www.ebxml.org/specdrafts/ebXML_TA_v1.0.pdf.

10. Greg Snider. Speak-Easy. A Conversation Architecture for E-Speak. July 1999. Hewlett-Packard Co., Unpublished Memo.

11. CIM Schema White Papers. MMTF, 2000.http://www.dmtf.org/educ/whit.html.

12. REMBO: A Complete Pre-OS Remote Management Solution. Rembo Technology.
http://www.rembo.com/docs/rembo/.

13. Service location Protocol. SUN Microsystems. http://www.sun.com/research/slp/.

14. OSGi Service Gateway Specification. Release 1.0. Open Service Group Initiative. 2000.
http://www.osgi.org/about/spec1.html.

15. Elliotte R. Harold. XML: Extensible Markup Language. IDG Books Worldwide, 1998.

16. Simple Object Access Protocol (SOAP) 1.1. W3C Note 08 May 2000. http://www.w3.org/TR/SOAP.

17. Karl Moss. Java Servlets. McGraw-Hill, 1999.