

# A Lightweight Service Grid based on Web Services and Peer-to-Peer

Markus Hillenbrand, Joachim Götze, Ge Zhang, and Paul Müller

University of Kaiserslautern, Germany  
Department of Computer Science  
{hillenbr, j\_goetze, gezhang, pmueller}@informatik.uni-kl.de

**Abstract** What is *The Grid*? This question has been asked by many people and has been answered by many more. On our way to the One Grid, it is currently only possible to distinguish between several Computational, Data and Service Grids. This paper introduces Venice, a lightweight Service Grid that allows for easy service deployment, easy maintenance and easy service usage. It contains management services, information services and application services that can be used to build distributed applications upon. Its main focus is on providing a flexible and dependable infrastructure for deploying services that do not require special knowledge and expertise in Grid computing.

## 1 Introduction

In 1969 Leonard Kleinrock imagined "the spread of computer utilities, which, like present electric and telephone utilities, will service individual homes and offices across the country" [1]. Ian Foster and Carl Kesselmann in 1998 initially tried to define a Grid that could help implement such computer utilities [2]. After several iterations, Ian Foster finally published his three point checklist [1] that must be met by a system to be called a Grid.

Today, a distinction between *Compute Grids*, *Data Grids* and *Service Grids* can be made to distinguish between the Grid systems available. There is no common definition for these Grid types, but here they are understood as follows. A Compute Grid provides resources for high throughput computing and makes possible time consuming calculations over the Internet (e.g. by giving standardized access to clusters). A Data Grid makes available disk space to securely store large data sets. A Service Grid finally uses these Grid types as a commodity and adds more value to the end-user by providing special services like virtualized applications or accurate algorithms and offering intuitive graphical interfaces to access those services.

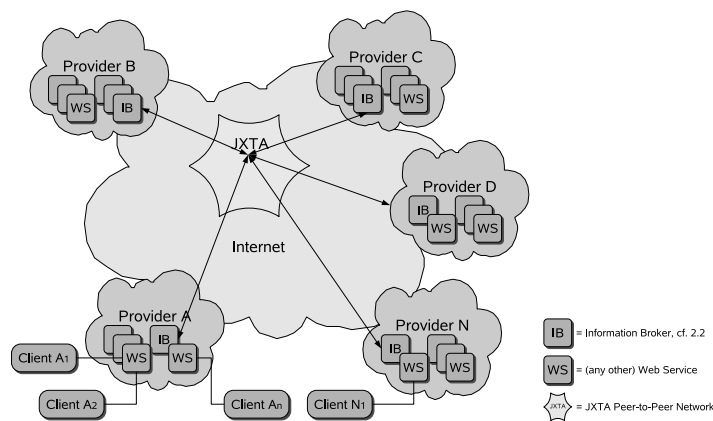
A lightweight Service Grid should additionally be *easy to deploy* (i.e. service developers have the ability to create and deploy services very quickly and with tools provided by the Service Grid), *easy to maintain* (i.e. service providers get tool support for managing, monitoring and configuring services at runtime) and *easy to use* (i.e. service consumers have the ability to use the services of the Service Grid without having to install special software and can access service functionality with graphical user interfaces). A Service Grid should be built on open standards and be applicable to different usage scenarios – not only high performance computing or high throughput data processing.

While Compute Grid like Unicore [3], gLite [4] and Globus Toolkit [5] (GT4) as well as Data Grids like the EU DataGrid Project [6] are deployed on a larger scale and used in world wide research projects, pure Service Grids are not yet available. GT4 could be regarded as an early version of a Service Grid, but its legacy (especially its Grid Security Infrastructure and the pre Web services components) makes it neither lightweight nor easy to deploy, use or maintain.

In the following, the Venice Service Grid will be outlined on a top level view. Some details about several services have been published previously, but they have since then been reorganized and completed to create a Service Grid. The roots of the Venice Service Grid are in a telephony project [7] that successfully made available signaling protocols like SIP, H.323 and several dozen supplementary services as Web services. In section 2 the overall architecture is explained and the services available in Venice are introduced. A performance measurement of the basic use case for accessing a service is shown in section 3 while section 4 concludes the paper in gives an outlook.

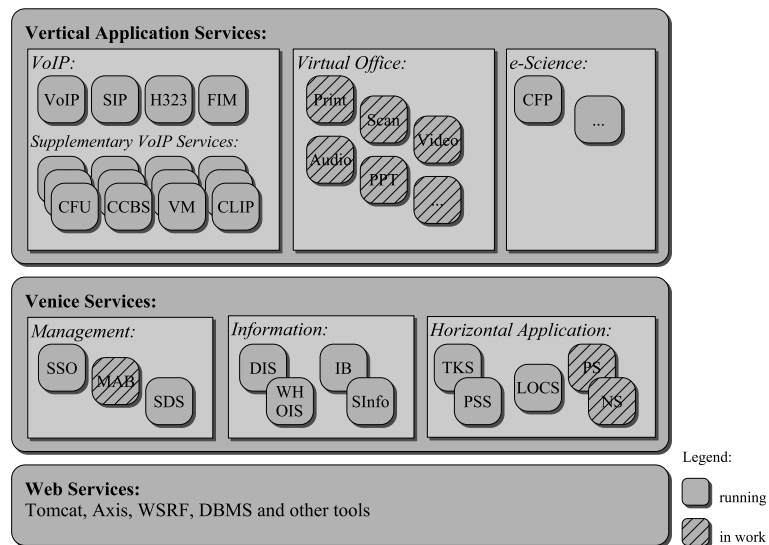
## 2 Architecture

The Venice Service Grid is an open framework for distributed applications that allows the easy creation, deployment, integration, and usage of services. The means to achieve this are virtualization of the resources used and abstraction from the underlying technology using services. By using the Venice Service Grid it is possible for a service provider to run a single security domain with services for the users of his domain. But it is also possible to connect other domains in order to gain access to the services provided by these domains. Such a service federation gives seamless access to all authorized services within the service federation. Every provider is primarily responsible for his own customers and services, and Venice provides a secure and standardized passage for data exchange and service access over the Internet using Web services and Peer-to-Peer (P2P) technology (as illustrated in Fig. 1).



**Figure 1.** The Providers' Perspective

The services offered by the Venice Service Grid are implemented as Web services directly on top of the Tomcat [8] servlet container and the Axis [9] SOAP engine. The services are described using the Web Services Description Language (WSDL) and communicate with SOAP over HTTP. The interface definitions are separated in an abstract service interface definition containing the port types and messages, a concrete interface definition containing the concrete binding, and a concrete service implementation definition containing the ports and locations of the real implementation. All data exchanged between the service entities are specified in publically available XML schema files. This allows for re-use of the WSDL and XML schema files in the service descriptions.



**Figure 2.** Architectural Overview

The services of the Venice Service Grid can be divided into three distinct categories (cf. 'Venice Services' in Fig. 2). *Management Services* are services necessary for the common usage and maintenance of users, resources, services, and other entities in the Grid. The *Information Services* are responsible for collecting, managing and distributing data needed by users or services. The *Application Services* finally are horizontal services that can be used by any application built on top of the framework and vertical services that are mostly valuable in a specific application domain. An application consisting of several vertical services ties together the Management, Information and horizontal Application Services to predefined user accessible workflows. Common to all those services is that they are equipped with at least one graphical user interface that can be started using a special software deployment service (cf. 2.1). Thus, every service can also be used separately.

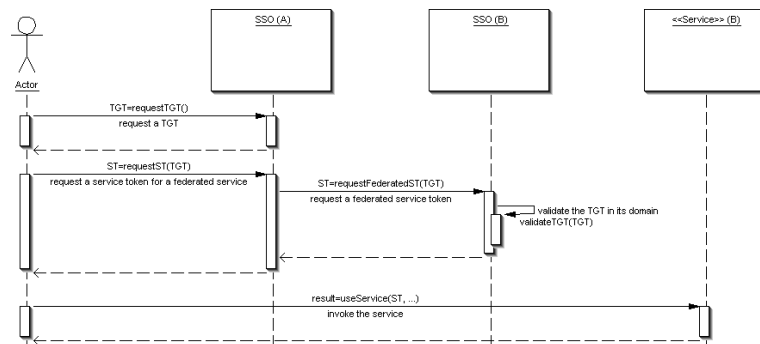
In the following subsections, the services already developed and currently being developed will be briefly introduced.

## 2.1 Management Services

In order to create an open and dependable service-oriented system, several basic services have to be provided that deal with management and maintenance of all services within the system. The following management services are part of Venice and make available authentication, authorization, accounting, billing and software deployment.

**Single Sign-on Service (SSO).** In order to access all services of a provider, a token-based single sign-on strategy [10] is the most promising approach for a lightweight Service Grid. A user has to authenticate once and will then receive a token allowing him to prove his identity in any further communication by providing this token. If the user is accessing a service, this token will allow the user not only to prove his identity, but also to prove his right to access this service. As there may be several services involved in a usage scenario, it is very important to use an authentication and authorization strategy based on single sign-on. This strategy allows the user to be authenticated to any service inside the authentication domain without the need to enter his credentials a second time.

The process of authentication in the Venice Service Grid is closely related to the authentication protocol Kerberos [11], but it is extended to fulfill the needs of a federated SSO environment on the basis of Web services. The Kerberos Protocol has the advantage of providing an easy to use basis for a SSO environment utilizing the security token to prove a successful sign-on.



**Figure 3.** Authentication and Authorization for a Federated Service

An authentication and authorization infrastructure consists of four basic components. The first component is the user or client. This component has to be integrated in the system of the client and handles any operation needed for authentication and authorization. The authentication service – providing everything needed to allow a user to claim an identity and proof that this claim is justified – is the second component. If the user has successfully claimed an identity, he receives a token (TGT) to prove his identity. This TGT has to be used with the third component: the authorization service. This service provides the user with service tokens (ST) that have a limited life-time

and allow him access to services, if he has the privileges to use these services. Finally there is the service component. The task of this component is to ensure that the user of the service can provide a proper ST, i.e. he is successfully authenticated and received authorization to access the service, and that the access rights are sufficient to utilize the requested service functionality.

In the Venice Service Grid, the authentication and authorization functionality is bundled in the SSO service. The Venice Service Grid offers a distributed authorization management. Every service is responsible for providing and interpreting authorization information. Therefore, the SSO service requests from a service the possible authorization attributes and their range. The service provider can then assign authorization information to user roles. This data will then be encoded by the originating service and stored into the SSO's database where it will be used to create service tokens. Also the SSO service can handle different types of service requests. A user signing on can belong to the local authentication domain or to a federated domain. Additionally, a user can call local services or services provided by a federated domain. A local user calling a federated service is illustrated in Fig. 3. Combined with the distributed authorization management a flexible cross-domain authentication and authorization infrastructure can be provided. A more detailed description of this token-based single sign-on solution developed for the Venice Service Grid can be found in [12].

**Metering, Accounting, and Billing Services (MAB).** It lies within the nature of a service-oriented architecture with a sophisticated authorization management to offer services that provide the same functionality with different quality of service for different users – e.g. according to their authorization level. And these services can be offered by different autonomous service providers. In the end, a flexible and open mechanism for metering and accounting service usage has to be provided as an infrastructural service. On top of that, a transparent and secure billing has to be guaranteed. The Venice MAB services provides an accounting and billing framework for measuring Web services, collecting and processing Web service usage records, generating corresponding accounting information, and charging and billing for the Web services usage according to the billing strategies. The MAB consists of four layers: the probe layer, the metering layer, the accounting layer, and the billing layer.

The *probe layer* comprises different types of probes which are responsible for measuring different types of Web services or other objects. These probes can either be integrated into the Web service or in the host where the Web service is running [13]. Different measured objects and different metrics require different probes; e.g. probes for measuring the duration of a Web service execution or probes for measuring the CPU utilization of a Web service instance. The *metering layer* provides services for gathering measurement information from probes, organizing, storing and transferring the measured data, and managing the behavior of probes according to accounting policies. The meter data is stored in meters temporarily and must be kept until it is collected by an accounting service. The *accounting layer* provides services for gathering meter data from different meters in its administrative domain, processing the meter data to generate Usage Records (UR) according to the accounting policies, and sending the URs to a billing service. Accounting information about composite Web services will be

correlated in this layer [14]. The accounting service is also responsible for controlling the behavior of the meters. URs generated by the accounting service are stored in an accounting information database. The URs stored in this database should be permanent and must be stored securely against unauthorized access. In a multi provider scenario, a token based accounting using P2P technology (like [15]) is under investigation. The *billing layer* provides services for collecting the URs from the accounting services, for calculating the resource usage for different users according to the pricing scheme and for generating the billing records. The billing records can be used to generate different reports. Users can then retrieve their billing reports. If a certain resource usage provision must be met, notifications will be sent to the authorization service to control different users' resource consumption.

**Software Deployment Service (SDS).** In order to provide an easy-to-use application to the end-user, it is important to dispense the user from tasks like installing and/or updating software. But software deployment and installation is a tedious task. In a service-oriented (or client / server) environment, a lot of different client and server types may participate. Clients have different kinds of hardware and software prerequisites they bring along while trying to access services. Notebooks and desktop computers are built using completely different processor architectures and operating systems than PDAs or cell phones provide. But within a service-oriented architecture, all those clients should be able to access and use the same service, regardless of their capabilities and prerequisites.

The Venice Software Deployment Service allows the user to run an up to date application without any further effort if his client is supported by the application provider. For most of the Venice services, this client software is just a graphical user interface and the client libraries needed to access Venice services. Additionally, such a service enables a service provider not only to maintain a state of the art software infrastructure, but also to automatically and dynamically replicate certain services to ensure service availability. This compensates for high load situations and establishes a replica management. As a result the Software Deployment Service reduces the administration effort on both sides, the user and the service provider. A comprehensive overview of the Software Deployment Service can be found in [16].

## 2.2 Information Services

A Grid needs a starting point where all necessary data can be found to access all other services. In the Venice Service Grid, the *Domain Information Service* provides all that data to users and services. Additional information services provide data about users and services or make local data available to other service domains.

**Domain Information Service (DIS).** This service is responsible for providing all necessary information for using services in a domain. Thus, the service is the starting point for any further service interaction and for example provides meta data about the single sign-on service, the information broker and the service domain itself.

**Who Is Service (WHOIS).** Using this service, it is possible to retrieve data about other users in the local or a remote domain. The data accessible through this service has to be authorized by the affected user in order to respect the user's privacy. This service will be complemented by a presence service in the future.

**Information Broker (IB).** This service is a generic service responsible for brokering service meta data, user meta data and other arbitrary data sets. The service meta data is used to find a suitable service for the user and can contain various kinds of data, e.g. data about the interface, dependability information, user's ratings about usability, etc. The user meta data is ranging from basic status information, e.g. the online status of the user, to more sophisticated data like the personal details of the user. These personal details may include textual information, but also other data is possible, e.g. audio files, images, etc. The implementation of the Information Broker is based on the JXTA P2P technology in order to attain scalability and to overcome some shortcomings of UDDI and other central registries [17].

Some of these shortcomings are *centralization* and *uptodateness*. A directory service like UDDI is typically a centralized service providing data about the registered services. This results in a single point of failure if the service is not available. During operation a single service can be a performance bottleneck and account for a severe performance impact. In Venice, the P2P technology is used to implement a decentralized registry. Additionally, the quality of a directory service is closely related to the quality of the data it is providing. In most directory services the data is entered manually and therefore a regular maintenance is needed to keep the data up-to-date. If the data repository is becoming large, this is only possible with a huge effort and as a result a lot of entries in a directory service might be outdated. In Venice, a service is automatically registered with the P2P network when it is deployed and active. Shutting down a service automatically results in de-registering it from the P2P network.

Therefore, using P2P technology allows to overcome these shortcomings and enhance the Information Broker with various beneficial properties:

- *Availability.* The use of a large number of peers allows for high availability of the meta data sets, because of data replication within the P2P network. While some of the peers are offline other peers can offer the requested meta data and therefore ensure availability.
- *Scalability.* Every peer of the network can be used to retrieve meta data and thus removes the performance bottleneck of a centralized registry.
- *Robustness.* While a centralized registry is a single point of failure, the distribution of meta data and the large number of access points to these meta data enhances the robustness of the Information Broker in contrast to centralized registries.

In order to lower the application requirements to use the Information Broker, the Information Broker provides a Web service interface to the client side. Therefore, the Information Broker is a dedicated service only concerned with data replication and retrieval, while providing transparent access to the P2P network for the requesting party, that does not directly participate in the P2P Network, but via a Web service interface [18]. Thus the clients requesting data from the P2P network do not have to use P2P software directly and don't have to open P2P ports on their operating system.

**Service Information (SInfo).** Every Venice service has to implement a special interface providing information about the service itself (name, purpose, icon, authorization scheme, etc.). Only by implementing this interface, the service can be seamlessly integrated into the framework.

### 2.3 (Horizontal) Application Services

Besides the management and informational services, several other useful services must be provided by a Service Grid in order to create a solid service foundation to build other services and applications upon. The horizontal application services within the Venice Service Grid can be used by any application domain and provide useful functionality of different kinds. Several services have already been developed and will be complemented by some more in the near future.

**Property Storage Service (PSS).** This service securely stores tag/value pairs. As simple as it might sound, its functionality has a large potential, e.g. for storing user preferences, service properties or other arbitrary data. This service is massively used by other services within the framework (e.g. the Notification Service or the Presence Service) in order to attain location transparency.

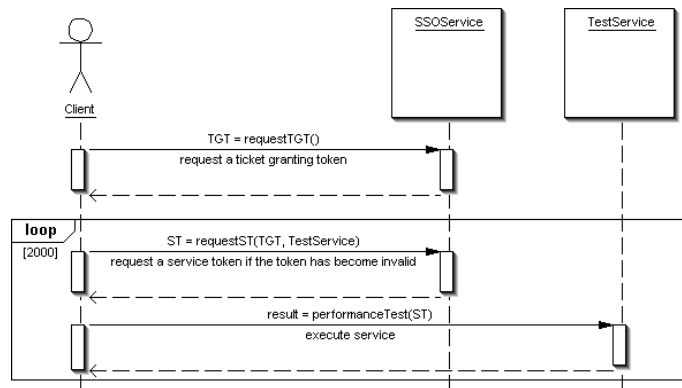
**Time Keeper Service (TKS).** This service can be used by other services or client programs to store measuring points consisting of a time stamp and a context. After measuring a sequence of data points, a time series analysis can be visualized and exported as image or text files. The results shown in section 3 have been measured and visualized using the Time Keeper Service. The overhead produced by the Time Keeper Service for each measurement is around 3500 ns.

**Location Service (LOCS).** When it comes to locating physical resources (printers, scanners, laboratory equipment, etc.) in three dimensional space, this service provides a geometric and symbolic positioning scheme and a useful plugin mechanism for integrating several location sensing techniques (like GPS, RADAR, etc) into clients. The Location Service provides standardized location information about humans or objects based upon the collected data of various different location systems. These location systems implement different location technologies, e.g. satellite based location technologies, cellular network based location technologies, indoor location technologies. Therefore all available data of the different location systems is used to compute the location of a person or object as exact as possible and to provide for the service user a transparent transition between different location systems. This finally allows for finding the nearest suitable resource for a specific task.

**Notification Service (NS).** A publish/subscribe notification service enables applications and services to subscribe for topics of interest and receive notifications when an event of interest appears. Such a service reduces polling and can even be used for deferred message delivery if a client or service is currently not running. Using the notification service it is possible to react to events occurring in the Service Grid by specifying

other services or tools that will be triggered by such an event. Every service implementing a special NotificationSink interface can receive notifications sent by the Notification Service. The services issuing notifications have to implement a NotificationSource interface so that it is possible to extract meta data about the potential notifications sent by the service.

**Presence Service (PS).** In order to strengthen user interaction inside the Venice Service Grid and to demonstrate the possibilities of the Information Broker, a Presence Service according to [19] is currently being developed. A presence service allows users to subscribe to each other and be notified of changes in state. Additionally, it is possible to apply certain rules of notification to groups of users. On top of the Presence Service and in combination with the Notification Service it will be possible to define actions that can be triggered when a certain state within a presence group appears.



**Figure 4.** Performance Evaluation (UML Diagram)

## 2.4 (Vertical) Application Services

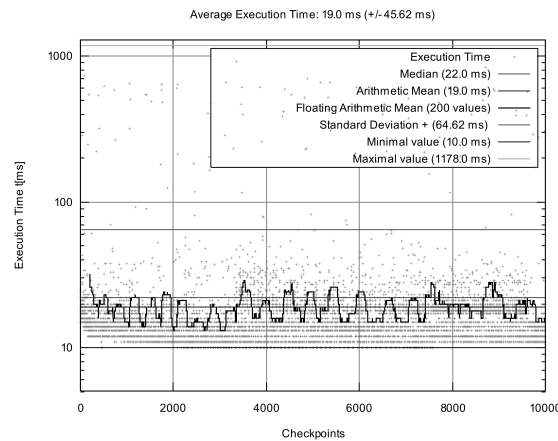
In a first application scenario, the framework has been used as an infrastructure for a Web services based Voice over IP (VoIP) application<sup>1</sup>. Several services dealing with VoIP functionality have been developed and brought into operation [7] (cf. *VoIP* in Fig. 2). A basic VoIP service abstracts from underlying VoIP protocols and gives access to both SIP and H.323 telephony. Additionally, phone calls can be made to the classic ISDN network by using an Asterisk server. Several additional supplementary services like Call Forwarding, Call Completion, a Voice Mailbox and a Call Center etc. make it worthwhile for the customers to use these VoIP services. The main goal was to move as much code as possible from the client to the services and to be able to seamlessly integrate new supplementary services or larger software updates without interruption.

<sup>1</sup> This research has been funded by Siemens AG, Munich.

In a current research project called DASIS<sup>2</sup> [20] (cf. *Virtual Office* in Fig. 2), the framework is used to virtualize the devices, tools and workflows of an ordinary office. Here, the Location Service will be responsible for locating physical resources like printers and scanners and associate it with other services or workflows.

### 3 Performance Evaluation

In order to evaluate the performance of the Venice Service Grid, the Time Keeper Service can be used to measure the time it takes to execute a specific task. The current time is calculated by a native (Windows or Linux) dynamic link library before and after a task. In this section the evaluation of an authorized Web service call as shown in Fig. 4 will be discussed. An evaluation of the SSO service itself has already been published in [12]. This measurement is not intended to compare the Venice Service Grid to other Grid technology; such a comparison would be unfair due to the different approaches of the diverse Grid solutions. It merely shows that the overhead produced by the SSO service (programmed in Java 2 5.0) is acceptable.

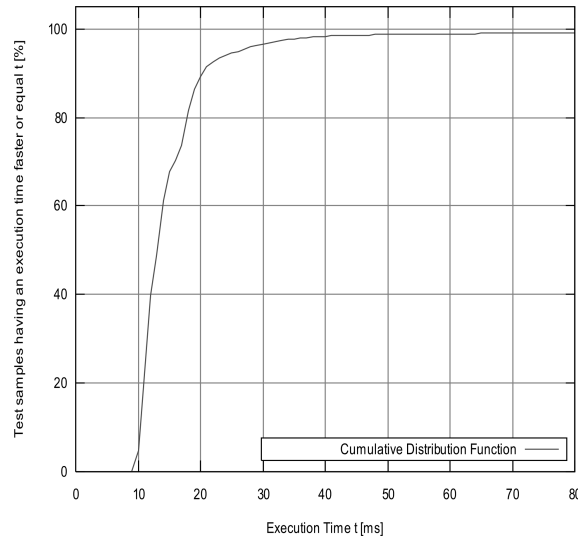


**Figure 5.** Performance Evaluation (Results)

The client has to request a TGT, request a ST and then subsequently call a simple Web service that checks the authorization of the client and returns the extracted username. This means, that at least some decryption of tokens has to be made by the service in order to extract the user name. Whenever the ST has become invalid, the client requests a new one from the SSO Service. In order to simulate a real service usage, the clients waits some time between the calls (randomly between 0 and 1 second).

<sup>2</sup> This research is part of the Cluster of Excellence "Dependable adaptive Systems and Mathematical Modeling" funded by the Program "Wissen schafft Zukunft" of the Ministry of Science, Education, Research and Culture of Rhineland-Palatinate, AZ.: 15212-52 309-2/40 (30)

In the evaluation scenario, five Java clients have been in operation simultaneously on five different hosts running under Windows and Linux. The Venice services have been deployed on a single Tomcat server (running under Linux with two Pentium 4 processors at 3 GHz). Server and clients have been connected by a 100 Mbit/s LAN. On the clients side, submitting the data to the Time Keeper Service adds an overhead of approx.  $7 \mu\text{s}$  to each measurement.



**Figure 6.** Performance Evaluation (Results)

Fig. 5 shows the time needed during 10,000 successful Web service calls. The arithmetic mean of a call is 19 ms. The standard deviation of the normally distributed value set is 64.62 ms. Additionally Fig. 6 shows the cumulative distribution of all test samples. Approximately 90 per cent of all request can be handled within 20 ms, 95 per cent within 25 ms and nearly all requests can be handled within 40 ms. This leads to the conclusion that there are only very few requests that consume a huge amount of time. A reason might be fluctuations of the network performance or within the Java virtual machine, e.g. garbage collection.

## 4 Conclusion and Future Work

This paper described the Venice Service Grid focussing on a lightweight approach for an easy to deploy, easy to maintain and easy to use Service Grid. The basic framework provides an open, dependable and flexible framework for deploying services to end-users. The architecture consists of Web services belonging to three different categories: Management Services, Information Services, and Application Services. The Management Services are necessary for the common usage and maintenance of Grid entities.

The Information Services collect, manage, and distribute information needed by users or services. As horizontal services the Application Services are typical services that can be used by any application that is build on top of this framework.

In the future the integration of the Web Services Resource Framework (WSRF) will allow for separating resources from services and – on the long run – allow for compatibility with other WSRF-enabled products like GT4. A connection to GT4, Unicore, gLite will transparently make available computing power and large data stores to all services and users inside Venice. The goal is to treat e.g. a GT4 node like a resource in Venice and thus enable all Venice services to acquire additional computing power or storage.

## References

1. Foster, I.: What is the Grid? A Three Point Checklist. Grid Today, 2002
2. Foster, I., Kesselman, C. (Ed.): The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, 1998
3. UNICORE: <http://unicore.sourceforge.net>
4. gLite: Ligthweight Middleware for Grid Computing <http://glite.web.cern.ch/glite/>
5. Globus Toolkit 4. <http://www.globus.org>
6. The DataGrid project. <http://eu-datagrid.web.cern.ch/eu-datagrid>
7. Hillenbrand, M.; Götze, J.; Müller, J. and Müller, P.: Voice over IP – Considerations for a Next Generation Architecture. Proceedings of the 31th Euromico Conference (Porto, Portugal, 2005)
8. Apache Tomcat: <http://tomcat.apache.org/>
9. Apache Axis: <http://ws.apache.org/axis/>
10. De Clercq, J.: Single sign-on architectures Proceedings of Infrastructure Security, International Conference, InfraSec (Bristol, UK, 2002)
11. Opplinger R.: Authentication Systems for Secure Networks. Artech House (1996)
12. Hillenbrand, M., Götze, J., Müller, J. and Müller, P.: A Single Sign-On Framework for Web Services-based Distributed Applications. Proceedings of the 8th International Conference on Telecommunications ConTEL (Zagreb, Croatia, 2005)
13. Machiraju, V., Sahai, A., Moorsel A. v.: Web Services Management Network. Proceedings of the 8th IFIP/IEEE International Symposium on Integrated Network Management (Colorado Springs, US, 2003)
14. Zhang, G., Müller, J., Müller, P.: Designing Web Service Accounting Systems. Proceedings of the Workshop on Web Services: Business, Financial and Legal Aspects (Geneva, Switzerland, 2005)
15. Liebau N.-C., Darlagiannis, V., Mauthe, A., Steinmetz, R.: Token-based Accounting for P2P-Systems. Proceedings of Kommunikation in Verteilten Systemen KiVS (Kaiserslautern, Germany, 2005)
16. Hillenbrand, M., Müller, P. and Mihajloski, K.: A Software Deployment Service for Autonomous Computing Environments. Proceedings of the International Conference on Intelligent Agents, Web Technology and Internet Commerce IAWTIC (Gold Coast, Australia, 2004)
17. Forster, F., de Meer, H.: Discovery of Web Services with a P2P Network Proceedings of International Conference on Computational Science (2004)
18. Hillenbrand, M., Müller, P.: Web Services and Peer-to-Peer in: Peer-to-Peer Systems and Applications (Springer Verlag, Berlin, 2005)
19. Day, M., Rosenberg, J., Sugano H.: RFC 2778: A Model for Presence and Instant Messaging.
20. Project DASIS: <http://www.dasmod.de>