Principles of Service Oriented Operating Systems

Lutz Schubert and Alexander Kipp

HLRS - University of Stuttgart
Intelligent Service Infrastructures
Allmandring 30, 70569 Stuttgart, Germany
schubert@hlrs.de

Abstract. Grid middleware support and the Web Services domain have advanced significantly over recent years, reaching a point where resource exposition and usage over the web has not only become feasible, but an actual commercial reality. Nonetheless, commercial uptake is still slow, and certainly not progressing the same way as other web developments have been taking place - this is mostly due to the fact that usage is still complicated and restrictive. This paper will discuss a new approach towards tackling grid-like networking across organisational boundaries and across different types of resources by moving main parts of the infrastructure to a lower (OS) level. This will allow more intuitive use of Grid infrastructures for current types of users.

Keywords: Service Oriented Architecture, distributed operating systems, Grid, virtualisation, encapsulation.

1 The "Grid"

Over recent years, the internet has become the most important means of communication and hence implicitly an important means for performing business and collaborative tasks. But the internet offers more potential than this: with the ever-increasing band-width, the internet is no longer restricted to pure communication, but is also capable of exchanging huge, complex data-sets within a fraction of the time once needed to fulfill these tasks. Likewise, it has become possible to send requests to remote services and have them execute complex tasks on your own behalf, thus giving the user the possibility to outsource resource intensive tasks.

Recently, web services have found wide interest both in the commercial and the academic domain. They offer the capability to communicate in a standardised fashion across the web to invoke specific functionalities as exposed by the respective resource(s), so that from a pure developer's perspective, the according capabilities can be treated just like a (local) library extension.

In principle, this would allow generation of complex resource sharing networks that build a grid-like infrastructure over which distributed processes can be enacted. However, the concept fails due to multiple reasons:

P. Primet et al. (Eds.): GridNets 2008, LNICST 2, pp. 56–69, 2009.

- non regarding standardised messaging, all providers will expose their own types of interfaces which is hardly ever compatible to another provider's interface even when essentially providing the same functions.
- additional capabilities are necessary in order to support the full scope of business requirements in situtations such as joint ventures, in particular with respect to legal restrictions and requirements.
- exposing capabilities over the web and adapting the own processes to cater for the according requirements, is typically very time and effort consuming
- a high degree of technical capabilities is required from the persons managing services and resources in a Virtual Organisation

Current Grid related research projects tberefore focus strongly on overcoming this deficiencies by realising more flexibile interfaces and more intuitive means of interactions and management. One of the most sophisticated project in this area is certainly BREIN [1] which tries to enhance typical Grid support with a specific focus on simplifying usage for the human behind the system. It therefore introduces extensions from the Agent, Semantic and AI domains, as shall be discussed in more detail below.

Whilst such projects hence enrichen the capabilities of the system, it remains questionable whether this is not a sub-efficient solution given that the underlying infrastructure suffers from the additional messaging overhead and that the actual low-level management issues have not been essentially simplified, but mostly just moved further away from the end-user.

This paper presents an alternative approach to realising a grid-enabled networking infrastructure that reduces the administrative overhead on the individual user significantly. The approach is basing strongly on the concept of distributed operating systems, as already envisaged e.g. by Andrew. S. Tanenbaum [2]. We will demonstrate how modern advances in virtualisation and encapsulation can be taken to a lower (OS) layer, that will not only allow cross-internet resource usage in a more efficient way, but will also enable new forms of more complex applications running on top of it, which exceeds current Grid solutions.

We will therefore examine the potentials of what is commonly called "the grid" (section 2), how a low level, OS-embedded Grid system would look like (section 3) and how it could be employed in collaborative environments (section 4). Basing on this we will the derive requirements towards future network support for such a system (section 5) and conclude with a summary and outlook (section 6).

2 Low level Grids?

All "Grid"-Definitions have in common that remote resources are used as, respectively instead of local ones, so as to enhance or stabilise the execution of complex tasks, see e.g. the EGEE project [3].

What is the advantage on an Operating System layer though?

Currently, more and more services are offered over the internet which are, in essence, at a low hardware level: storage can be rented on the web that grants access from all over the world; computational platforms can be used to reduce

the local CPU usage; printing services expose capabilities to print high quality documents and photos; remote recording service store data, burn it to a disk and send it by mail and so on.

Interoperability & Usability. Using these capabilities currently requires the user to write code in a specific manner, integrating public interfaces, making use of APIs and / or using browser interfaces over which to up- and download data. As opposed to this, local resources are provided to the user via the OS typical interfaces, such as Explorer, which are typically more intuitive to use and in particular do use OS layer drivers to communicate with the devices, rather than leaving it up to the user to make the resources usable.

However, in particular low layer resources, such as the ones mentioned above, could in principal be treated in the same fashion as local resources on cost of delays for communication. We will treat the communication problem in more detail below.

Resource Enhancement. The obvious benefit of integrating remote resources consists in the enhancement to local resources in order to e.g. extend storage space, but also to add additional computing power and extend local storage, as we will discuss below. In particular on the level of printers, disk burners and storage extensions, remote resources are already in use, but with above described drawbacks.

Stability and Reliability. By using remote resources for backup and replication features, overall process execution and data maintenance of the system can increase significantly. Already there are backup solutions duplicating disk storage onto a resource accessible over the web and systems such as the one by Salesforce [4] allows for stable, managed execution of code by using local replication techniques (cf. [5], [6]). However, none of them are intuitive to use and allow for interactive integration.

Tight integration on the OS layer would hence allow making use of remote resources in a simple fashion, without having to care about interoperability issues.

3 Building up the Future Middleware

Web Service and Grid middlewares typically make use of HTTP communication with the SOAP protocol on top of it, which renders interactions rather slow and thus basically useless on a level that wants to make full use of the resources. Classical Grid applications mostly assume decoupled process execution where interaction only takes place to receive input data and return, respectively forward operation results. Even “fast” Grid networks are building more on dynamic relationships than on actual interactivity [7], [8], [9].

There have hence been multiple approaches to improve message throughput, such as SOAP over TCP and XML integration into C++ [10], [11] or standard binary encoding of SOAP messages [12], [13]. None of these approaches however

can overcome the full impact of the high SOAP overhead, thus rendering lower level resource integration nearly impossible.

3.1 A Model for Service Oriented Operating Systems

As has been noted, the main feature of Grid / Web Service oriented middlewares consists in provider transparent resource integration for local usage. Notably higher capabilities are of relevance for distributed process enactment, but from each provider's local perspective the same statement holds true.

On the operating system level, resources can not be treated as transparently as on the application level, as the resources, their location and the means of transaction have to be known in full detail - implicitly taking this burden away from the user. In a classical OS approach, the operating system would *just* provide the means of communication and leave it up to higher layers to take care of identification and maintenance of any remote resource.

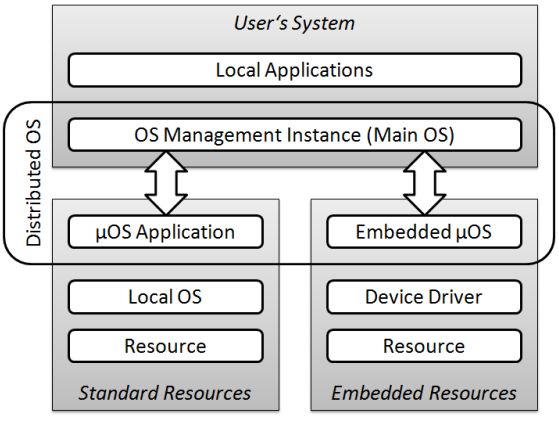

Fig. 1. Schema of a service oriented, distributed operating system

Figure 1 shows a high level view on a Service Oriented OS and its relationships to resources in its environment: from the user's perspective the OS spans all resources, independent of their location etc. and represents them as locally integrated. In order to realise this, the *main* operating instance provides common communication means on basis of an enhanced I/O management system.

We distinguish three types of OS instances, depending on usage and location:

The Main Operating Instance represents the user side - as opposed to all other instances, this one maintains resources in the network and controls their usage. It performs the main OS tasks, such as process maintenance and in particular virtual memory management. Any MicroKernel instance are extensions to the local resources and treated as such. The system therefore does not promote a flat grid hierarchy where all nodes can be treated as extensions to each other.

The Embedded MicroKernel is similar to the classical extension of a networked OS: it is actually a (network) interface to the device controller - depending on the type of resource, such control may range from a simple serial / parallel interface for e.g. printers and storage, up to a standalone job- and resourcemanagement e.g. if the resource is a computer itself that can host jobs.

The Standalone MicroKernel is an extension to the Embedded MicroKernel and allows to run the same functions *on top of a local operating system*, i.e. it acts as an application rather than a kernel. As such it is principally identical to the current Grid middleware extensions which run as normal processes.

Leaving performance aside[1], local and remote resources could principally be treated in the same fashion, i.e. using common interfaces and I/O modules. We will use this simplification in order to provide a sketch of the individual operating system instance and its main features *as they differ from classical* OS structures.

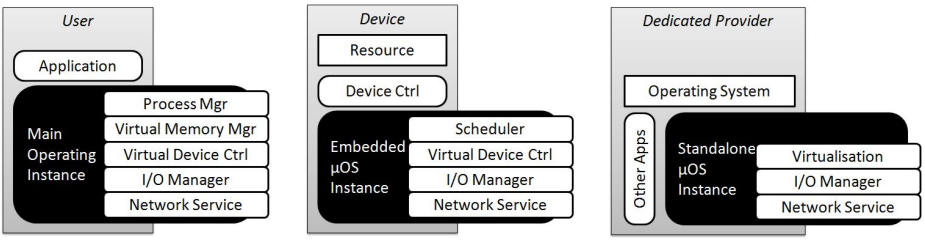

Fig. 2. Sketch of the individual kernel models

Main issue with respect to building a grid like Service Oriented OS consists in identifying the required connectivity between building blocks and resources, and thus candidates for outsourcing and remote usage. It is obvious that actual process management, execution, virtual memory access and all executable code needs to be accessible via (a) local cache, so that according capabilities can not be outsourced without, as a consequence, the operating system failing completely.

The key to all remoting consists in the capability of virtualising devices and resources on a low, operating-system level. Virtualisation techniques imply an enhanced interface that exposes the functionalities of the respective resource in a common structure [14] understandable to the operating system. Accordingly, such interfaces act as an intermediary between the device controller and the respective communication means and as such hide the controller logic from the main operating instance. Using pre-defined descriptions for common devices, this technique not only increases the "resource space", but also reduces the negative impact that device drivers typically inflict upon the operating system. Instead, maintenance of the resource is completely separated from operating system and user context.

[1] In fact this would impact on performance in such a way that hardly any reasonable usage of the system would be possible anymore.

In the following sections, we will describe in more detail, how in particular storage and computational resources will be integrated into and used by a Service Oriented Operating System:

3.2 Memory and Storage Resources

We may distinguish three types of storage, depending on speed and availability:

- local RAM: the fastest and most available type of storage for the computer. It is typically limited in size and is reserved for the executable code and its direct execution environment
- local storage, such as the computer's hard drive is already used as an extension to the local RAM for less frequent access or as a swapping medium for whole processes. It also serves as pure file storage.
- networked storage typically extends the local file storage and only grants slow access, sometimes only with limited reliability, respectively availability.

All these resources *together* form the storage space available to the user. Classical operating systems distinguish these types on an application layer, thus leaving selection and usage to the user. With the Service Oriented Operating System, the whole storage is virtualised as one virtual memory which the OS uses according to processes' requirements, such as speed and size. In addtion, the user will only be aware of the disk space distributed across local and remote resources.

This implies that there is no *direct* distinction between local RAM and remote, networked storage. Process' variables and even parts of the code itself may be stored to remote resources, if the right conditions apply (see computation resources below).

By making use of a "double-blind" virtualisation technique, it is ensured that a provider can locally maintain his / her resources to his / her own discretion (i.e. most likely to the operating system's requirements) without being impacted by the remote user. In other words, remote access to local resources is treated identically to local storage usage, with the according swapping and fragmenting techniques and without the remote user's OS (the main instance) having to deal with changes in allocation etc.

In case of virtual memory access, "double-blindness" works as follows (cf. Figure 3: the calling process maintains its own local memory address range (VMem_{proc}) which on the global level is mapped to a global unique identififer (VMem_{global}). Each actual providing resource must convert this global identifier to a local address space that reflects the actual physical storage in case of embedded devices, or the reserved storage in case of OS driven devices (Mem_{local}).

Measuring Memory Usage. The Service Oriented Operating System monitors memory usage of individual processes in two ways: depending on the actual usage in the code (the usage context) and according to the access frequency of specific memory parts. Accordingly, performance and memory usage will improve over time, as the process behaviour is continuously analysed.

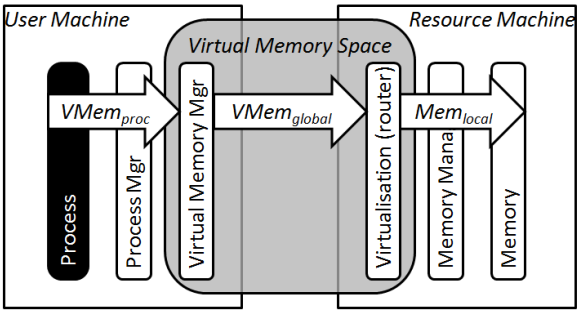

Fig. 3. Double blind memory virtualisation

Whilst the drawback of this consists in sub-optimal behaviour in the beginning, the advantage is that coders and users will not have to consider aspects related to remote resources. Notably, preferences per process may be stated, even though the operating system takes the final decision on resource distribution.

3.3 Computational Resources

Computational resources are in principal computers that allow remote users to deploy and run processes. Depending on the type of resource (cf. above), these processes are either under the control of the Service Oriented Operating System or of the hosting OS.

Computing devices do not directly extend the pool of available resources in the way storage resource do: as has often be shown by parallelisation efforts, and by multi-core CPUs, duplication of the number of CPUs does not imply duplication of speed. More precisely, a CPU can typically take over individual processes, but not jointly participate in executing a single process.

Parallel execution in current single core CPUs is realised through process-swapping, i.e. through freezing individual processes and switching to other scheduled jobs. The Service Oriented Operating System is capable of swapping across computational resources, thus allowing semi-shared process execution. By building up a shared process memory space, parts of the process can be distributed across multiple computational resources and still executed in a linear fashion as the virtual storage space builds a shared memory.

Measuring Process Requirements. The restrictions of storage usage apply in even stronger form to process execution: each process requires a direct hosting environment including process status, passing which is time critical. Whilst the hosting environment itself is part of the embedded, respectively standalone kernels, the actual process status (cache) is not. Therefore the Service Oriented Operating System analyses the relationship between process parts and memory usage in order to identify distributable bits.

Typical examples consist in applications with a high user interaction on the one hand, but a set of computing intensive operations in the background upon

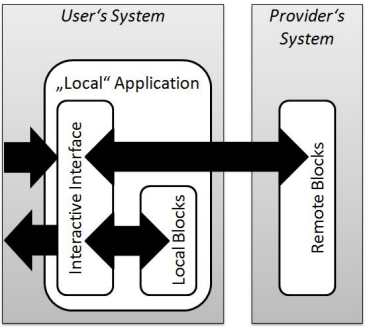

Fig. 4. Invoking remote code blocks from a local interactive interface

request, such as CAD and render software which requires high availability and interactivity during modelling, but is decoupled from the actual rendering process.

3.4 Other Resources

Other devices, such as printers, DVD writers or complex process enactors all communicate with the main operating instance over the local OS instance (embedded / standalone) using a common I/O interface.

Main control over the resource is retained by the provider, thus allowing for complete provider side maintenance: not only can local policies be enforced (such as usage restrictions and prioritisation), but also the actual low-level resource control is maintained, i.e. the device drivers, thus decoupling it from the user-side maintenance tasks.

This obviously comes at the cost of less control from the user side and accordingly less means to influence the details of individual resources, as all are treated identical. As an example, a printer allowing holes to be punched into the printouts will not be able to expose these capabilities to the main operating instance, as it is not a commonly known feature - notably, the protocol could be extended to cater for device specific capabilities too.

However, it is the strong opinion of the authors, that such behaviour is not always desirable. As with the original Grid, resources should be made available according to need, i.e. printers upon a print-out request, CPUs upon executing a high amount of computing intensive processes, storage upon shortage etc. Typically in these cases, no special requirements are put forward to these resources so that simple context-driven retrieval and integration is sufficient. We can assume without loss of generality that all additional requirements are specified on an application level.

4 Putting Service Oriented Operating Systems to Use

The benefits of such a system may not be obvious straight away, though their applications are many-fold: with remote resources being available on a low

operating system layer, applications and data can be agnostic to its distribution across the internet (leaving issues of speed aside) - this leads to a variety of use cases, which shall be detailed in this section:

4.1 Mobile Office

With code and data being principally distributed across the internet, execution of applications and their usage context becomes mostly independent of the runtime environment of the main system and its resources. Instead, the same application could be hosted and maintained remotely, whilst the actual user accesses the full environment from a local low-level computer with the right network connection, as only direct interfaces and interactive code parts may have to be hosted locally.

Accordingly, an employee could principally use the full office working environment from any location with according web-access (cf. also section 5) with any device of his or her choice. By using the main machine to maintain the user profile and additional information such as for identification, the user can also recreate the typical working environment's look and feel non-regarding his or her current location. This will allow better integration of mobile workers in a company, thus realising the "mobile office".

4.2 Collaborative Workspaces

The IST-project CoSpaces [15] is examining means to share applications across the internet so as to allow concurrent access to the same data via this application, i.e. to collaboratively work on a dataset, such as a CAD design. One of the specific use cases of CoSpaces consists in a mechanical engineer trying to repair a defect aircraft without knowing the full design: through CoSpaces he will not only get access to the design data, but also may invite experts who have better knowledge about the problem and the design than he is. Using a common application, the experts can *together* discuss the problem and its solution and directly communicate the work to be done in order to fix it. This implies that all people involved in the collaboration can see the same actions, changes etc. but also concurrently work on the data.

With a Service Oriented Operating System, application sharing becomes far more simple: as the actual code may be distributed across the web, local execution will require only the minimal interfaces for interaction whilst the actual main code may be hosted remotely. With the right means to resolve conflicting actions upon both code and data and with network-based replication support (cf. section 5), it will be possible to grant concurrent access to common *code and parameter environment parts*, thus effectively using the same application across the network in a collaborative fashion.

CoSpaces is currently taking a web service based approach, which does not provide the data throughput necessary for such a scenario but is vital to elaborate the detailed low-level protocols and the high-level management means which would have to be incorporated into the OS to meet all ends.

4.3 Distributed Intelligent Virtual Organisations

A classical use case for distributed systems consists in so-calles "Virtual Organisations" (VO) where an end user brings together different service providers so that they collectively can realise a task none of them would have been able to fulfil individually. Typically, this involves execution of a cross-organisational workflow that distributes and collects data, respectively products from the individual participants, thus generating a larger product or result.

Currently, VO support middlewares and frameworks are still difficult to use and typically require a lot of technical expertise from the respective user, ranging from deployment issues over policy descriptions to specifications integration and application / service adaptation. And hardly ever the overall business policies and goals of the user are respected or even taken into consideration.

The IST project BREIN [1] is addressing this issue from a user centric point of view, i.e. trying primarily to make the use of VO middleware systems easier for the user by adding self-management and intelligent adaptation features as kind of an additional management layer on top of the base support system. This implies enhancements to the underlying infrastructure layer that is responsible for actual resource provisioning and maintenance.

With a Service Oriented Operating System, remote resources can be linked on an OS level as if locally available - accordingly, execution of distributed workflows shifts the main issue related to discovery, binding and maintenance down to the lower layer. This allows the user to generate execution processes over resources in the same way as executing simple batch programs: by accessing and using local resources and not caring about their details. User profiles may be exploited to define discovery and integration details, which again leads to the problem of technical expertise to fully exploit the respective capabilities.

Future version of the SOS may also consider more complex services as part of the low-level resource infrastructure. The BREIN project will deliver results with respect to how the according low level protocols need to be extended and in particular how the high level user interface of the OS needs to be enhanced to allow optimal representation of the user's requirements and goals.

5 Future Network Requirements

Like most web-based applications, the Service Oriented Operating System too profits most from an increase in speed, reliability and bandwidth of the network. The actual detailed requirements are defined more by the respective use case than by the overall system as such - however, one will notice that there is a set of base requirements and assumptions that are recurring in recent network research areas, which in fact can be approached on different levels and with different technologies. Most prominent of these are obviously virtualisation technologies which hide the actual context and details of the resources and endpoint from the overlaying system (cf. [1], [16], [17]). This does not only allow for easier communication management, but in particular for higher dynamicity and

mobility, as the full messaging details do not need to be communicated to all the endpoints and protocols do not need to be renegotiated every time.

Other issues along the same line involve secure authentication, encryption and in particular unique identification under highly dynamic circumstances, such as in mobile environments. Intermediary servers and routers may be used for caching and / or as resources themselves, but should not be overloaded with additional tasks of name and location resolution, in particular if the latter are subject to change. With the current instability of network servers, the overhead for (re)direction reliability just diminishes the efficiency of the system.

Resources can be treated both as consumers and providers in a SOS network, meaning that hosting (serving) and consumption of messages need to be treated at the same level, i.e. consumers need to be directly accessible without intermediary services. The IPv6 protocol [18] offers most of the base capabilities for stable direct consumer access under dynamic conditions.

What is more, to increase the performance of the system, context information related to connectivity and reliability of the respective location need to be taken into consideration to ensure stable execution on an Operating System level. Protocols such as SIP allow communication of location sensitive information [19] but there is no approach as yet to exploit network-traffic related quality information in the context of traffic control and automatic protocol adapatation via location specific data which is heuristically sufficient to indicate average performance issues.

In the following we will elaborate the specifics in the context of the individual use cases in some more detail:

Mobile Offices do not put forward requirements with respect to concurrent access (such as the other scenarios) but are particularly sensitive to performance and reliability issues as the main goal tends towards distributed application execution with little to no performance loss. We need to furthermore distinguish between access to local (intranet) and remote resources. Working towards uniformity in both usage and production, both connectivity types should make use of the same physical network ideally over a self-adaptive protocol layer. This would reduce the need of dedicated docking stations.

As we will see, the security requirements in this scenario are of less priority than they appear to be: even though the corporate network may grant secure access to the local data structures, authentication and encryption do not require the same dynamicity support as with the other scenarios.

Collaborative Workspaces share access to common applications and need to distinguish between two types of data even though identical from the application point of view: user specific and shared information. Though access is performed identically, the network may not grant cross-user access to private information whilst shared data must be subject to atomic transactions to reduce potential conflicts. Note that the degree of transaction freedom may be subject to individual roles' privileges.

Accordingly, the network needs to provide and maintain role and profile specific information that can be used for secure authentication, access restriction and in particular for differentiation between individual code-data-user-relationships.

Further to this, replication can be supported through low-level multiplexing according to context-specific information, thus reducing the management overhead for the operating system. In other words extended control mechanisms of the higher layer may be exploited to control replication on the network layer. By making use of unique dynamic identifiers as discussed above, this would allow usage-independent access to data even in a replicated state and with according dynamic shifting of this replicated data.

Distributed VO Operations do maintain access restriction according to dynamic end-user profiles, but typically do not share applications. Accordingly, most relevant for this scenario is data maintenance across the network, i.e. conflict anagement and atomic transactions to reduce the impact of potential conflicts.

6 Summary and Implications

We introduced in this paper a next generation operating system that realises grid on a low OS level thus allowing for full integration and usage of remote resources in a similar fashion to local ones. With such an operating system, management and monitoring of resources, workflow execution over remote resources etc. is simplified significantly, leading to more efficient grid systems in particular in the areas of distributed computing and database management.

Obviously, such an operating system simplifies resource integration at the cost of flexibility and standardised communication on this low layer. In other words, all low level control is passed to the OS and influence by the user is restricted. For the average user, this is a big advantage, but it increases the effort of Kernel maintenance in case of required updates from the standardisation side.

As it stands so far, the Service Oriented Operating System does not cater for realiability issues, i.e. it treats resource according to their *current* availabilty - this means that the Operating System will shift jobs and files to remote resources independently of potential reliability issues. Accordingly, code and data blocks may become invisible during execution and even fail, leading to unwanted behaviour and system crashes.

Different approaches towards increasing reliability exist, ranging from simple replication to device behaviour analysis. The approach will differ on a case-by-case-basis as it will depend on the requirements put towards the respective processes and / or data. Notably the system can principally extend local execution reliability, when replicating critical processes.

Along that line it is not yet fully specified, how these requirements should be defined and on what level - it is currently assumed that the user or coder specifies these directly per process or file. However, in particular for commonly used devices (printers, storage etc.) it may be sensible to defind and use user profiles that represent the most typical user requirements under specific circumstances and could thus serve as the base setup for most resources.

As the system is still in its initial stage, no performance measurements exist as yet that could quantify the discussions above - in particular since reliability issues have not been covered so far. This would also involve measurements involving different network conditions, i.e. bandwidth, reliability etc. in order to enable more precise defintion of the requirements towards future networks.

Acknowledgments

This work has been supported by the BREIN (http://www. gridsforbusiness.eu) and the CoSpaces project (http://www.cospaces.org) and has been partly funded by the European Commission's IST activity of the 6th Framework Programme under contract number 034556 and 034245. This paper expresses the opinions of the authors and not necessarily those of the European Commission. The European Commission is not liable for any use that may be made of the information contained in this paper.

References

1. BREIN: Business objective driven reliable and intelligent grid for real business, `http://www.eu-brein.com`
2. Tanenbaum, A.S.: Modern Operating Systems. Prentice Hall PTR, Upper Saddle River (2001)
3. EGEE: Enabling grids for e-science, `http://www.eu-egee.org`
4. SalesForce: force.com - platform as a service, `http://www.salesforce.com`
5. Mitchell, D.: Defining platform-as-a-service, or PaaS (2008), `http://bungeeconnect.wordpress.com/2008/02/18/defining-platform-as-a-service-or-paas`
6. Hinchcliffe, D.: The next evolution in web apps: Platform-as-a-service, PaaS (2008), `http://bungee-media.s3.amazonaws.com/whitepapers/hinchcliffe/hinchcliffe0408.pdf`
7. Wesner, S., Schubert, L., Dimitrakos, T.: Dynamic virtual organisations in engineering. In: Computational Science and High Performance Computing II. Notes on Numerical Fluid Mechanics and Multidisciplinary Design, vol. 91, pp. 289–302. Springer, Berlin (2006)
8. Wilson, M., Schubert, L., Arenas, A.: The trustcom framework v4. Technical report (2007), `http://epubs.cclrc.ac.uk/work-details?w=37589`
9. Jun, T., Ji-chang, S., Hui, W.: Research on the evolution model for the virtual organization of cooperative information storage. In: International Conference on Management Science and Engineering, 2007. ICMSE 2007, pp. 52–57 (2007)
10. van Engelen, R.A., Gallivan, K.: The gsoap toolkit for web services and peer-to-peer computing networks. In: Proceedings of the 2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2002), pp. 128–135 (2002)
11. Head, M.R., Govindaraju, M., Slominski, A., Liu, P., Abu-Ghazaleh, N., van Engelen, R., Chiu, K., Lewis, M.J.: The gsi plug-in for gsoap: Enhanced security, performance, and reliability. In: ITCC conference 2005, pp. 304–309 (2005)

12. Ng, A., Greenfield, P., Chen, S.: A study of the impact of compression and binary encoding on SOAP performance. In: Schneider, J.G. (ed.) The Sixth Australasian-Workshop on Software and System Architectures: Proceedings, pp. 46–56. AWSA (2005)
13. Smith, J.: Inside Windows® Communication Foundation. Microsoft Press (2007)
14. Nash, A.: Service virtualization – key to managing change in soa. Service Virtualization – Key to Managing Change in SOA (2006)
15. CoSpaces: Cospaces project website, `http://www.cospaces.org`
16. 4WARD: The 4ward project, `http://www.4ward-project.eu`
17. IRMOS: Interactive realtime multimedia applications on service oriented infrastructures, `http://www.irmosproject.eu/`
18. Hinden, R., Deering, S.: Ip version 6 addressing architecture (fc 2373). Technical report, Network Working Group (1998)
19. Polk, J., Brian Rosenberg, J.: Session initiation protocol location conveyance. Technical report, SIP Working Group (2007)