Design and Evaluation of a Collaborative Online Visualization and Steering Framework Implementation for Computational Grids

Morris Riedel [#1], Thomas Eickermann [#], Wolfgang Frings [#], Sonja Dominiczak [#], Daniel Mallmann [#], Thomas Düssel [#], Achim Streit [#], Paul Gibbon [#], Felix Wolf [#] [+], Wolfram Schiffmann [*], Thomas Lippert [#]

[#] *Central Institute for Applied Mathematics, John von Neumann Institute for Computing*
Forschungszentrum Jülich, D-52425, Jülich, Germany
[1] `m.riedel@fz-juelich.de`

[+] *Department of Computer Science*
RWTH Aachen University, D-52056, Aachen, Germany

[*] *Institute of Computer Architecture, Department of Computer Science*
University of Hagen, 58097, Hagen, Germany

Abstract— **Today's large-scale scientific research often relies on the collaborative use of a Grid or e-Science infrastructure (e.g. DEISA, EGEE, TeraGrid, OSG) with computational, storage, or other types of physical resources. One of the goals of these emerging infrastructures is to support the work of scientists with advanced problem-solving tools. Many e-Science applications within these infrastructures aim at simulations of a scientific problem on powerful parallel computing resources. Typically, a researcher first performs a simulation for some fixed amount of time and then analyses results in a separate post-processing step, for instance, by viewing results in visualizations. In earlier work we have described early prototypes of a Collaborative Online Visualization and Steering (COVS) Framework in Grids that performs both - simulation and visualization - at the same time (online) to increase the efficiency of e-Scientists. This paper evaluates the evolved mature reference implementation of the COVS framework design that is ready for production usage within Web service-based Grid and e-Science infrastructures.**

I. Introduction

Grid infrastructures such as DEISA, EGEE, OSG or TeraGrid provide wide varieties of Grid services to enable large-scale resource sharing and access to unprecedented amounts of various types of Grid resources. An important objective for Virtual Organizations (VOs) [1] that result from these sharing across organizational boundaries is to make efficient use of the provisioned computational Grid resources such as supercomputers, clusters, or server farms.

Scientific applications within these VOs and underlying Grid infrastructures aim at simulations of a physical, biological, chemical, or other types of domain-specific processes or unsolved scientific problems. These applications typically rely on *parallel computing techniques* to compute solutions for such scientific problems. Parallel computing simulations use computers with multiple processors that are able to jointly work on one or more specific problems at the same time. The outcome of these simulations are often analyzed in a *separate post-processing step*, for instance by viewing the results in a scientific domain-specific visualization application.

In order to increase the efficiency of e-Scientists and thus their complete VOs, the *collaborative online visualization and steering (COVS)* technique emerged that performs the simulation and visualization at the same time. In this context online visualization refers to e-Scientists that are able to observe the intermediate processing steps during the computation of the simulation. This allows for computational steering [2] to influence the computation of the simulation during its run-time on a supercomputer or cluster. This saves cost-intensive computational time on Grid resources by quickly reacting on potentially misrouted applications with steering their parameters back to correct values or even guide the applications with steering to interesting locations in the model.

The lack of a widely accepted common COVS frameworks within the major Grid middlewares (e.g. UNICORE, gLite, Globus Toolkits) motivates the development of the framework presented here. In earlier work we described the requirements and design issues of early prototypes of the COVS framework [3]. This paper emphasizes on the reference implementation of the COVS framework that is based on the *UNICORE Grid middleware* and the *VISIT communication library* [4]. However, an implementation of a COVS framework will be only accepted in realistic Grid scenarios if it reaches high levels of usability and sophisticated performance. Therefore, the contribution of this work is an evaluation of the framework's architectural design. Thus, to demonstrate that the COVS framework is of practical relevance, the reference implementation is applied to a real-world test case, including performance analysis of data connections and scalability anyalysis of the key component enabling the collaboration.

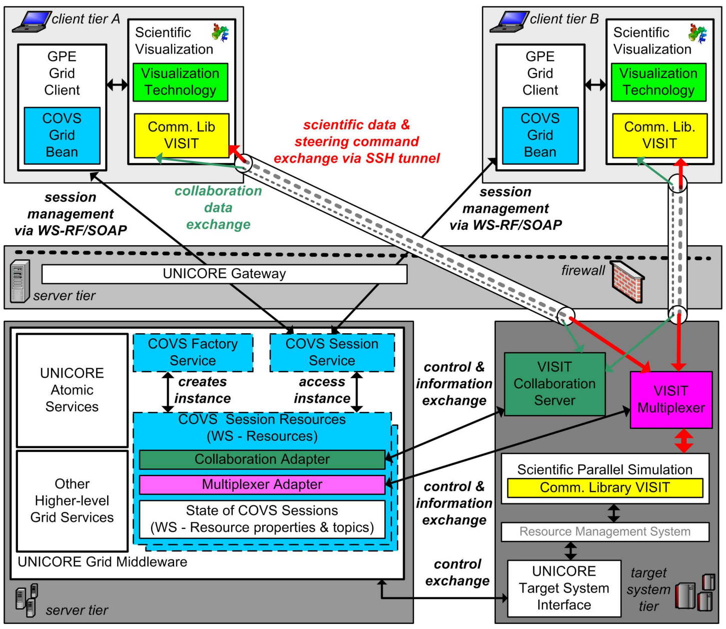


Fig. 1. COVS framework reference implementation that is based on UNICORE as Grid middleware and the VISIT communication library.

Following the introduction the scene is set in Section 2 where we present the design of the COVS framework reference implementation in UNICORE and its core building blocks. Section 3 then evaluates the proposed architectural design with respect to usability for end-users and performance measurements. A survey of related work is described in Section 4, while this paper ends with some concluding remarks.

II. Design and Reference Implementation

This section introduces the *core building blocks* of the COVS framework and its components for collaborative scenarios that are necessary in large-scale Grid infrastructures. The intention of it is to describe how existing components of the visualization and Grid community fit into the designed framework and how several components can be augmented to provide a *full functional COVS reference implementation* for end-users that meet the requirements that appear within Grids.

Figure 1 provides an overview of the reference implementation of the COVS framework's architectural design. The framework determines the architecture of COVS applications and defines the overall structure by addressing the key responsibilities and the interaction between its components. The main motivation of the COVS framework is to support *High Performance Computing (HPC)* applications in the area of e-Science and thus to be used as a *tool for efficiently solving complex scientific problems* such as grand challenge problems. In addition, the COVS framework must be integrated seamlessly into the different Grid environments (e.g. DEISA) by hiding the differences in security policies, systems architectures, access methods and resource representations to reach an overall transparency for end-users.

A. Addressing Collaborative Aspects in COVS

A COVS framework implementation in Grids allows for an easier collaboration between geographically distributed e-Scientists during data analysis. Therefore, the COVS design raise a demand for a *multiplexer entity* (e.g. VISIT Multiplexer) that distributes the scientific data output from one parallel simulation to n bi-directional connections that connect the n scientific visualizations. This multiplexer represents a novel component and can be considered as a key component within the framework and interconnects the simulation with n visualizations as shown in Figure 1. In addition, the design relies on a *collaboration entity* (e.g. the new VISIT Collaboration Server) that transports collaboration data (e.g. turn of viewpoints) from one visualization to all other n-1 visualizations. Hence, the collaboration entity interacts with all visualizations to ensure every participant shares the same view on the data. Both the scientific and collaboration data transfer have to be secured, for instance with an SSH tunnel to avoid firewall problems. The framework provides the *COVS Grid service* that controls the multiplexer and collaboration server entities. That means a participant in the *master role* is able to add and remove participants using the COVS Grid services and is the only one that is able to submit/abort the scientific simulation to/on the Grid. The session management represents a major difference than single user control as well the problem that appear when when one participant steers one parameter to the right while another one steers it to the left. We use an explicit request token mechanism (*setsteerer()*) to ensure that only one participant is able to steer the simulation at the same time. A similar mechanism is used to specify that only one participant at the same time is allowed to change the view (*setcollaborator()*). To sum up, the functionality of one participant differs from the functionality of another.

B. Architectural Design to Support e-Science Applications

The architecture of the COVS framework is specifically designed to support a wide variety of *parallel simulations* and *visualizations* from numerous scientific domains that both represent rather domain-specific core building blocks of the framework. As often within HPC environments, such simulations are typically implemented by using the *Message Passing Interface (MPI)* standard or other parallel computing paradigms. The parallel simulations that are used in conjunction with the COVS framework are submitted to the computational resource using a Grid client (e.g. GPE Grid Client [5]) and the underlying Grid middleware (e.g. UNICORE) of the correspondent Grids (e.g. DEISA). In the context of the online visualization of its outcome, this simulation must provide data in a stepwise fashion to enable the visualization of single computational steps. Hence, if the simulation provides interim results, they can be transferred via a communication library (e.g. VISIT) to the visualization and afterwards turned into *visualization idioms* [6] by a visualization technology (e.g. VTK) to show the result of the actual computation status. In this context, a visualization idiom is any specific sequence of data enrichment and enhancement transformations, visualization mappings, and rendering transformations that produce a display of a scientific dataset within a visualization application. In addition to scientific data, steering commands must be transferred from the visualization to the simulation and this data transfer can be securely accomplished via *bi-directional connections over SSH*, because most firewalls allow access via SSH to the highly protected systems running the simulation. The COVS reference implementation uses a technique for an SSH tunnel establishment in Grids that was shown by Riedel et al. in [7] and relies on an RSA-based authentication while a Grid middleware performs an RSA key exchange.

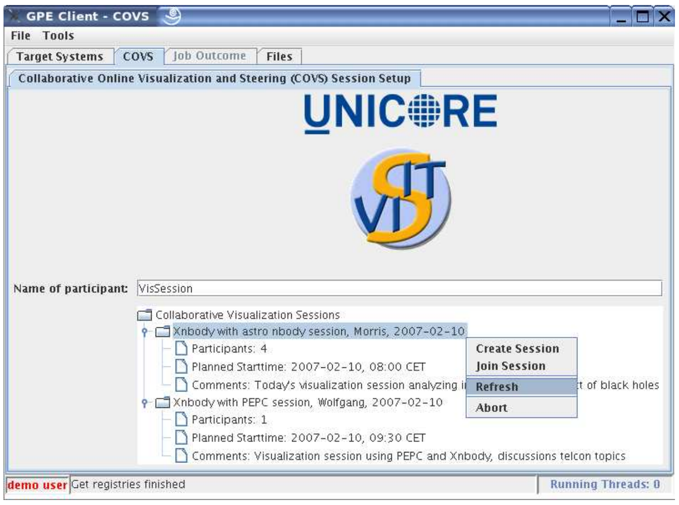
C. Enable Collaboration with COVS Grid Services

Today, Grid services conform to *Open Grid Services Architecture (OGSA)* [8] and typically implemented by using the OASIS *Web Services Resource Framework (WS-RF)* [9] standard. In particular, the WS-RF compliant COVS Grid service represent another core building block of the COVS framework. In the reference implementation, for example, it is implemented as a higher-level service on top of the *UNICORE Atomic Services (UAS)* [10] that provide basic job submission/management and file transfer functionalities. In more detail, the COVS Grid service consists of two WS-RF compliant services as shown in Figure 1, namely the *COVS Factory service* and the *COVS Session service*. The COVS Factory service implements the WS-RF implied factory pattern that is defined as any kind of service that brings a stateful WS-Resource (e.g. COVS session resource) into existence [9]. It can be used to create new *COVS session resources* while the access to these resources is provided by the COVS Session service. The COVS session resource expose the status of the COVS session by using the WS-Resource properties [9] mechanisms. Figure 1 illustrates the COVS Session service that consists of a *MultiplexerAdapter* that controls the multiplexer entity. The COVS Session service provides operations (e.g. *RemoveParticipant()*) that are forwarded to this multiplexer using an XML-based protocol. Similar to the Multiplexer-Adapter, the design comprises a *CollaborationAdapter* that is also integrated as one component within the COVS Session service. It can be used to control the collaboration server via service operations (e.g. *shutdown()*) that forwards actions to the collaboration server using an XML-based protocol. Any information gathered by the collaboration server is forwarded to the service and in turn converted into properties of the COVS session resource.

To sum up, the scope of the WS-RF compliant COVS Session service reaches from dynamic collaboration to authorized session management control. *Authentication* and *authorization* is provided by the Grid middleware. In the reference implementation, end-users of the COVS framework are *authenticated* via their X.509 credentials at the *UNICORE Gateway* [11] and authorized within the *UNICORE User DataBase (UUDB)* [12]. Finally, the next paragraph reveals how these management capabilities for collaborative scenarios are accessed via common open standards such as WS-RF.

D. Open Standards-based COVS Session Management

In general, a major disadvantage of *Service Oriented Architectures (SOAs)* such as modern Grids is that its core technologies (e.g. WS-RF over SOAP [13]) are typically not suitable for a high amount of scientific data that is regularly transferred between the simulation and visualization. Therefore, the COVS framework relies on SSH tunnels for bi-directional connections, but uses the open standard technologies of SOAs for the COVS session management as shown in Figure 1.



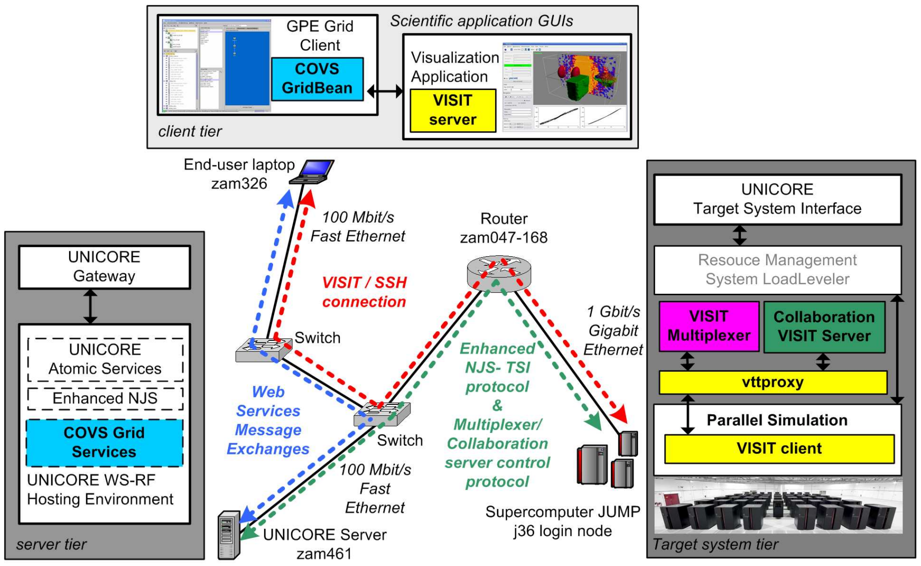
Name of participant: VisSession

Collaborative Visualization Sessions
Xnbody with astro nbody session, Morris, 2007-02-10
Participants: 4
Planned Starttime: 2007-02-10, 08:00 CET
Comments: Today's visualization session analyzing i t of black holes
Xnbody with PEPC session, Wolfgang, 2007-02-10
Participants: 1
Planned Starttime: 2007-02-10, 09:30 CET
Comments: Visualization session using PEPC and Xnbody, discussions telcon topics

demo user Get registries finished Running Threads: 0

Fig. 2. GPE Application client with loaded COVS GridBean.

Also, Figure 1 illustrates that a end-user of the COVS framework uses two applications at the client tier that is the scientific visualization and a Grid client (e.g. GPE Grid client). The Grid client is used to submit the scientific parallel simulation to the Grid middleware (e.g. UNICORE) but is also used for the SSH key exchange that is necessary to establish an



Scientific application GUIs
GPE Grid Client
COVS GridBean
Visualization Application
VISIT server
client tier
End-user laptop zam326
100 Mbit/s Fast Ethernet
VISIT / SSH connection
Router zam047-168
1 Gbit/s Gigabit Ethernet
UNICORE Target System Interface
Resouce Management System LoadLeveler
VISIT Multiplexer
Collaboration VISIT Server
vttproxy
Parallel Simulation
VISIT client
Target system tier
UNICORE Gateway
UNICORE Atomic Services
Enhanced NJS
COVS Grid Services
UNICORE WS-RF Hosting Environment
server tier
Switch
Switch
Web Services Message Exchanges
Enhanced NJS- TSI protocol & Multiplexer/ Collaboration server control protocol
100 Mbit/s Fast Ethernet
UNICORE Server zam461
Supercomputer JUMP j36 login node

Fig. 3. Network infrastructure of the Grid testbed for the evaluations of the COVS framework implementation. The performance of the VISIT/SSH connection (red) is of major interest since this connection transfers the scientifi c data from the parallel simulation (VISIT client) to the scientifi c visualization (VISIT server). Furthermore this connection is responsible to transfer steering commands from the scientifi c visualization in nearly real-time to the parallel simulation.

SSH connection between client tier and target system tier. In more detail, a dedicated COVS framework-specific client plug-in (e.g. COVS GridBean) is responsible for that following the mechanisms as described in [7].

The main goal of the COVS client plug-in is a sophisticated GUI that provides the functionality to monitor and control a COVS session using Web service message exchanges. To provide an example, Figure 2 shows the COVS GridBean of the reference implementation and its GUI in the context of joining available COVS sessions that are exposed by the COVS Grid service implementation within the Grid middleware. Of course, the GUI also provides functionalities such as connect/disconnect participants during the session run-time, pause simulation/continue simulation or abort a session just to list some. This functionality is conveniently provided via pop-up menus to end-users of the framework.

III. Design Evaluation

This section evaluates the proposed architecture of the COVS framework for Grid and e-Science infrastructures with respect to different usability metrics, focussing on particular on performance measurements of key components/protocols, because this has a very high impact on the overall acceptance of the framework by end-users.

A. Experimental Setup

This paragraph describes an experimental configuration that lays the foundation to examine the feasibility of the COVS framework reference implementation. The experimental set up and later evaluations depend on the particular deployment of the framework to allow for performance evaluations. Figure 3 is part of the JuNet network infrastructure within Forschungszentrum Jülich and illustrates a particular deployment of the framework. It emphasizes on the network interconnection between the machines running COVS components.

On the OSI-layer 3, JuNet is composed of various IP-subnets that are interconnected by a *central router (zam047-168)*. Client- and server-machines are typically connected to switches with 100 or 1000 Mbit/s Ethernet, depending on their communication requirements. The *end-user laptop (zam326)* and the *UNICORE server (zam461)* are located in the same subnet and are attached to the infrastructure with 100 Mbit/s interfaces. Therefore their communication does not traverse the router. The login-node of the supercomputer JUMP, is connected to JuNet via two channeled 1 Gbit/s interfaces. Since JUMP is located in a separate IP-subnet, the laptop and the UNICORE server communicate with JUMP via the router.

The testbed will be used for performance measurements. The performance of the illustrated Web service message exchanges can be disregarded since they are only used to transport small XML documents via SOAP that are not data-intensive. Also, the NJS-TSI protocol as well as the collaboration and multiplexer server control protocol only transport small pieces of text and XML over the wire. Hence, the only data-intensive connection that is of major interest for performance evaluations is the *VISIT/SSH connection* between the end-user laptop and the supercomputer JUMP.

B. Performance of Bi-directional Online Connection

One of the key considerations within Grids is the *secure transport of information and data* between users and resources. In particular, it is one crucial point of the design since *interactive steering of simulations raises a high demand for low latency* to reach real-time behavior. Hence, in order to provide sophisticated steering capabilities of parallel simulations in a timely manner, the protocols used in the COVS framework design must achieve low latency over secure bi-directional connections that can be realized via SSH tunnels. In this paragraph, we provide performance measurements to indicate that the usage of the communication library within Grid environments via SSH (VISIT/SSH) is still feasible instead of using the plain VISIT protocol unsecured over TCP (VISIT/TCP). In this context it is important to understand that the performance or usability of the system relies on what kind of low-level transport is used for data exchange. Because Web services and XML provides flexibility at the cost of performance, the COVS framework uses Web services only for session management but also provides a mechanism whereby a high-performance connection can be used. Until now, TCP was feasible in not distributed environment and the usage of SSH within Grids is feasible since it provides firewall-friendly secure connections. Hence, this distinction between management information transfer (via Web services) and scientific data transfer (via SSH) is a fundamental approach of the COVS framework. If all is done via Web services, the latency is very noticable to the human users and the bandwidth is significantly reduced due to the verbosity of XML or UNICORE internal protocols as shown in an earlier approach [14].

In Figure 3, the VISIT communication library uses a *vttproxy* component [4] that acts as a proxy for the visualization located in the same security domain as the simulation. The measurement presented in Figure 4 were performed with a generic ping-pong program that is part of the VISIT distribution. This program sends messages of varying length from the *VISIT client (i.e. parallel simulation)* to the *VISIT server (i.e. visualization)* and back to the client several times and provides statistics for the latency. In particular, Figure 4 shows the latencies that are measured as half of the message round-trip time.

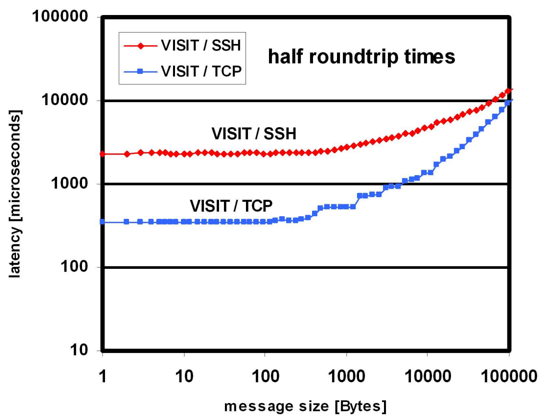


Fig. 4. Ping-Pong latency measured as half of message round-trip time using VISIT/SSH and VISIT/TCP within the Grid testbed.

The Performance measurements between a Linux-Visualization Client (1.7 GHz Pentium) and the login node of the Jülich Supercomputer JUMP over the Grid testbed (see Figure 3) result in an SSH connection startup time of 1.5 seconds, a ping-pong latency of 2299 microseconds and a bandwidth of 84 MBit/s (for a message size of 1 MByte) compared to 345 microseconds / 86 MBit/s for a direct unencrypted VISIT connection via TCP between the same systems. In this test scenario, the bandwidth degradation from using the SSH-tunnel is only about 2.3%. While an increase of latency by 2 milliseconds is significant (a factor of more than 6 in the testbed), it is less relevant in wide-area networks, where the signal propagation delay is about 1 millisecond per 200 km, not including additional delays in network-components as switches or routers.

UNICORE uses Web service message exchanges to collect information for the establishment of the SSH connection or to control collaborative sessions. Therefore, UNICORE only affects the time needed to initialize SSH connections but not the scientific data transfer over SSH itself. To conclude, the COVS framework reference implementation establishes a secure way of providing a bi-directional connection between the simulation and visualization *without critical loss of latency*.

C. Collaborative versus Single User Control (Multiplexer)

The VISIT multiplexer is the essential new component introduced into the data flow to enable collaborative visualizations. To measure its influence on the performance, we have compared the throughput of messages of different sizes with and without multiplexer and with varying number of visualization clients attached in a testbed with gigabit ethernet and without SSH. The results are illustrated in Figures 5 and 6. While the throughput per participant descreases with growing number of prticipants the aggregate throughput (sum of throughput of all participants) even increases due to better utilization of the network.

multiplexer setup	**throughput** [Mbit/s]		
	1 Byte	1 MByte	**aggregated** 1 MByte
w/o multiplexer (direct)	0,025	608	---
w multiplexer, 1 participant	0,021	362	---
w multiplexer, 5 participants	0,013	123	615

Fig. 5. Throughput dependence on message size and multiplexer setup.

D. Analysis of Improved Usability Dimensions/Metrics

The usage of the COVS framework with typical HPC parallel applications in Grid infrastructures provides e-Scientists with an improved usability that is extremely hard to measure precisely. Therefore, this paragraph provides a closer look on an example experiment that describes the PEPC simulation [15] and Xnbody [16] visualization highlighting different dimensions and metrics used in the analysis.

The first dimension focusses on the *domain/technical knowledge of the end-users*. Figure 7 presents a snapshot of the VISIT-enabled Xnbody visualization used with the VISIT-

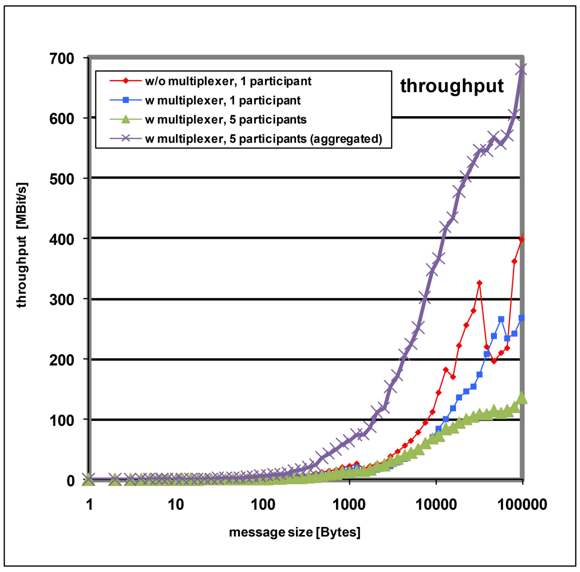
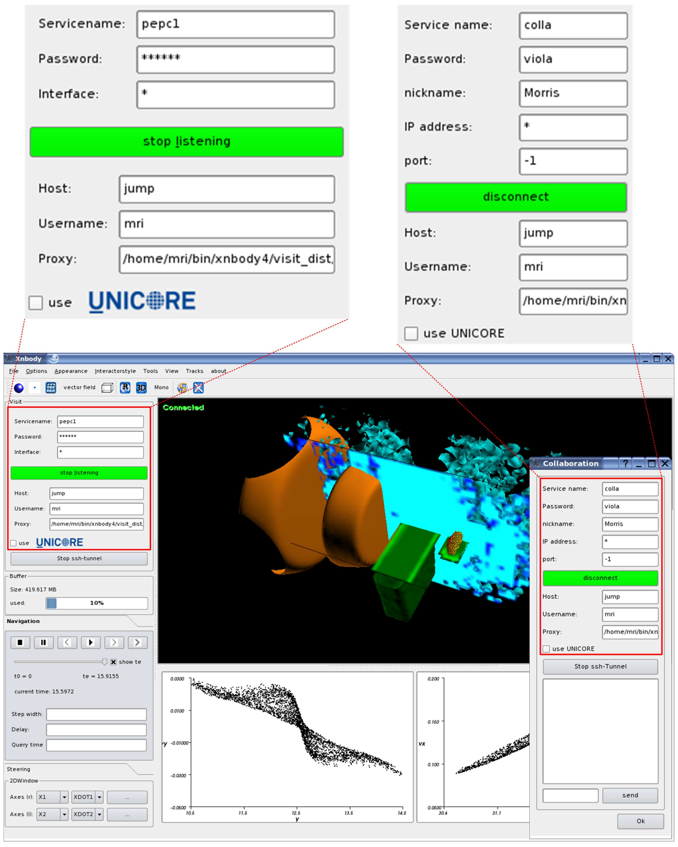


Fig. 6. Throughput showing the overhead/scalability of the multiplexer.



Servicename:	pepc1
Password:	******
Interface:	*
	stop listening
Host:	jump
Username:	mri
Proxy:	/home/mri/bin/xnbody4/visit_dist
☐ use UNICORE	

Service name:	colla
Password:	viola
nickname:	Morris
IP address:	*
port:	-1
	disconnect
Host:	jump
Username:	mri
Proxy:	/home/mri/bin/xn
☐ use UNICORE	

Fig. 7. Using the Xnbody scientifi c visualization without the COVS framework implementation implies knowledge of technical details. A end-user must manually provide all necessary information (zoomed red boxes). Instead, when using the 'use UNICORE' checkbox, all pieces of information will be automatically provided by UNICORE.

enabled parallel simulation PEPC, but without using the benefits a COVS framework implementation. Therefore, an end-user has to manually provide all necessary details for the connection establishment via VISIT (seappassword, seapservice [4]) and the startup of the *vttproxy* to enable the transfer via the SSH tunnel. That means an end-user must provide a *Servicename (seapservice)* and a *Password (seappassword)* for the identification of the visualization at the remote site. Purely optional is the definition of the *Interface* to choose from different network interfaces that may be available (* for default). In addition, the end-user must provide a full qualified *Host* and a *Username* on the remote machine. Most notably, the end-user must know the exact path to the *Proxy* (vttproxy) on the remote machine. This could be particularly difficult since the installation of the VISIT library on a remote machine such as a supercomputer is usually undertaken by the administrators of this machine and not by an individual end-user. All in all, scientists that represent an end-user must know a lot of technical details before they can connect to an ongoing parallel simulation with this VISIT-based application. The provisioning of the above described necessary pieces of technical information can be much more *automaticely managed* by using the COVS framework implementation presented here. When using the PEPC parallel simulation and Xnbody visualization with the COVS framework in collaborative scenarios all these issues become transparent to the end-users by using the *'use UNICORE'* checkbox. In other words, all the described pieces of information do not have to be provided by the end-users anymore, instead UNICORE provides all these details to the scientific visualization via named pipes.

The next usability dimension demonstrates the *improvements in handling the complexity of collaborative scenarios*. As shown in Figure 7, typically more issues arise when performing *collaborative visualization scenarios* and much more connection details have to provided. First and foremost, all participants of a COVS session are identified with the VISIT *seapservice:seappassword* combination. Hence, without using the COVS framework, one participant must manually configure the VISIT multiplexer by providing all the different *seapservice* and *seappassword* combinations of the geographically dispersed participants. This information must be exchanged using out-of-band mechanisms such as EMail, Telephone or Skype. In addition, this user must have the knowledge which participants are allowed to participate in a COVS session and thus must provide *manual authorization and authentication of participants*. Hence, this can be particularly difficult when the number of participants is significantly increased. In addition, all end-users have to know the contact information of the collaboration server as shown in Figure 7, and if communicated over insecure networks, also the *Host* and *Username* to establish an SSH connection to the host of the collaboration server must be manually provided by the end-users within the Xnbody GUI. COVS solves this using Grid technologies so that all participants can conveniently request for participation via the COVS GridBean and all the necessary pieces of information are automaticly transferred, which includes the automatic configuration of the VISIT Multiplexer and VISIT Collaboration Server.

Another usability metric is the *simpler session management interface*. The leader of the session conveniently uses the COVS GridBean GUI to manage the session and the GUI in turn forwards operations (e.g. connect/disconnect participant) to the underlying UNICORE Grid middleware using Web service message exchanges. All this is hidden from all the participants that just use the GPE Grid client with the COVS GridBean. Even the use of the Grid client itself provides improved usability by providing a convenient way to submit jobs to remote resources.

Another metric for improved usability is the *simpler interface for authentication*. The authorization and authentication of end-users is done automatically by the Grid middleware and retains the important single sign-on feature of Grid environments. Thus, instead of providing several passwords for remote hosts, only once the keystore of the Grid middleware must be unlocked via one password to gain full access.

To sum up, the transparency of Grids is the overall goal of using the COVS framework with scientific parallel applications. Its improved usability legitimates the use of the framework by e-Scientists in real application use cases within production Grids, because the efforts of scientists for connecting to a remote simulation is significantly reduced as described by the different usability metrics. Furthemore, several Grid infrastructures, for instance several sites within the EGEE Grid, map certificate identities to pool accounts which leads to the fact that a username and hostname can not be known beforehand for static manually configured SSH tunnels. In such scenarios, the COVS framework provides capabilities to establish an SSH tunnel to the remote site and thus also to use applications in highly dynamic environments.

E. Support for End-users of the COVS Framework

The fundamental idea of the COVS framework is to provide e-Scientists with a tool that is easy to deploy and use by *avoiding work that is not related to their own scientific area or application code*. Therefore, this paragraph evaluates what end-users actually have to do when they want to use a COVS framework implementation. Thus, it is clarified which components must be already provided by a COVS framework implementation and which components the end-users have to provide in which form. Needless to say, it is important to evaluate if the work that end-users have to invest to use the COVS framework is feasible and can be expected to be accepted in production Grids today.

For the clarification of these questions it is worth looking at real production Grids such as D-Grid [17] or DEISA. Both infrastructures have already plans to move to Web service-based Grid middlewares (e.g. UNICORE 6) in the near future. Besides the core Grid services (e.g. job submission and management, file transfer, and storage), additional higher-level services such as COVS Grid services can also be deployed within the Grid middlewares. Hence, the core building blocks Grid middleware and thus COVS Grid services will be automatically provided by the e-Science infrastructure. The deployment of the COVS Grid services implies the configuration and installation of the dedicated communication library it is based on (e.g. VISIT), including its multiplexer and collaboration entities. Furthermore, the Grid client will also be provided to gain access to the infrastructures.

To conclude, a deployed implementation of the COVS framework design provides the most core building blocks for end-users in a ready-to-use form, *only the scientific area-specific parallel simulation and visualization must be provided by the end-user*. This implies, that both components must be instrumented with communication library-specific calls (e.g. VISIT server and client calls) in order to enable the usage of the COVS framework. Many parallel e-Science applications already have visualizations that are based on post-processing techniques. Hence, the *real work that end-users have to do is to instrument their own code with communication library calls* to enable the data and steering command exchange.

Finally, end-users of a COVS framework implementation must request a personal X.509 certificate at the corresponding *Certificate Authority (CA)*. Using such a certificate allows end-users to gain access to the infrastructure and its resources via the Grid client (e.g. GPE Client) and also to the COVS Grid services when loading the client-specific plug-in (e.g. COVS GridBean) into this client. However, this is a general demand for end-users that want to use resources within Grid infrastructures and not a COVS framework specific issue.

IV. COVS Framework User Communities

The COVS framework implementation is used by the ASTRO-Grid D community Grid within D-Grid in the context of the nbody parallel simulation code. Furthermore, it is used with the PEPC parallel application at the *John von Neumann Institute for Computing (NIC)* in Jülich in conjunction with the Xnbody visualization (see Fig. 7). This demonstrates the adoption by user communities that use the framework to significantly increase their analysis of scientific data provided by nbody or PEPc through collaborative sessions with participants ot the whole VO.

V. Related Work

There is a quite a lot of related work in the area of visualization and steering technologies in Grid infrastructures. The UK RealityGrid project provides a steering library that enable calls which can be embedded into its three components that are simulation, visualization, and a steering client. Recently, prototypes of this library are renewed to be conform to OGSA. In comparison to the work presented here, the COVS framework is loosely coupled to the Grid middleware while the recent efforts around the RealityGrid steering library are focusing on its tighter integration into the *Imperial College e-Science Networked Infrastructure (ICENI)* [18].

Another well-known system is developed within the Austrian Grid initiative. The *Grid Enabled Visualization Pipeline (GVID)* [19] provides high quality visualizations of scientific datasets on thin clients. In more detail, the data of the scientific simulations are efficiently encoded with the H262 code into a video stream and transferred to the thin client afterwards. The client, in turn decodes the video stream for visualization of the scientific data. The system also offers steering capabilities similar to the approach within this paper, but realized via so-called Event-Encoders that run on the thin clients and sent steering commands to the simulation. However, the major difference to our approach is that it is not seamlessly integrated as one higher-level service into a common Grid middleware.

NAREGI provided an API that consists of a visualization library and a Grid visualization service API [20]. The visualization library can be used to connect simulation applications by the support of multiple visualization functionalities. The visualization service API wraps this library to provide Grid service functionality that are a set of WS-RF compliant services, for instance Coupled Simulation Services, Post-Processing Services, or Molecular Visualization Services. Even if this approach is using the WS-RF standard similar as our approach, the internal architecture is rather different. To provide an example, the scientific data as well as its rendering is completely computed within the Grid that finally is represented by a compressed image. The COVS framework, on the other hand, sends the scientific data in an online connection to the client for rendering and to allow for accurate steering of result parameters.

Finally, Brodlie et al. describes in [21] a well known rather high-level framework for collaborative visualization in distributed environments, while our contribution is much more oriented and closer to production Grid scenarios today.

VI. Conclusions

This paper introduced the reference implementation of the COVS framework design by using the UNICORE 6 Grid middleware and the VISIT communication library. The evaluations have shown that the choice of the communication library is a crucial step for the implementation of a COVS framework since this core building block has dependencies with all others. Most notably, the implementation of the COVS framework can be used by all visualizations and simulations that rely on the selected communication library. Nevertheless, the communication library can also be replaced by another library that provides similar capabilities such as bi-directional online connections, scalable data multiplexers and collaboration servers.

Also, the Grid middleware is an important cornerstone and the proof of concept implementation with UNICORE 6 indicates that the COVS framework could be, in principle, implemented in any WS-RF compliant Grid middleware. This open source implementation of the COVS framework was successfully demonstrated at the EuroPar 2006 conference, at the Fujitsu UNICORE booth at OGF18 in Washington, at the Supercomputing 2006 conference in Tampa and in a visualization and steering session at OGF19 in Chapel Hill. Furthermore, it was demonstrated to end-users at a DEISA Training. More recently, Intel flyers use the COVS framework implementation presented within this paper for marketing of their open source GPE client suite.

Nevertheless, deploying the proposed COVS architecture is an important next step to broadly incorporate implementations of the COVS framework into production Grid environments. The reference implementation described here relies on the WS-based UNICORE 6 middleware. UNICORE 6 will be soon evaluated by the DEISA and D-Grid Grid infrastructures for production usage. When these production Grids shift their access methods from UNICORE 5 to UNICORE 6, the reference implementation of the COVS framework can be also deployed as one higher-level service for production usage. In general, efficient usage of computational resources by using COVS and thus beneficial steering technologies must be improved with the goal to incorporate such steering tools into the usual workflows of e-Scientists. *Once an implementation of the COVS framework is deployed within production Grids such as DEISA or D-Grid, an important tool for a efficient use of Grid and e-Science infrastructures is accomplished.*

References

[1] I. Foster *et al.*, *The Anatomy of the Grid - Enable Scalable Virtual Organizations.* John Wiley and Sons Ltd., 2003.

[2] R. Marshall *et al.*, “Visualization methods and simulation steering for a 3D turbulence model for Lake Erie,” *ACM CIGGRAPH Computer Graphics*, vol. 24(2), pp. 89–97, 1990.

[3] M.Riedel *et al.*, “VISIT/GS: Higher Level Grid Services for Scientifi c Collaborative Online Visualization and Steering in UNICORE Grids,” in *To appear in Proc. of Int. Symposium on Parallel and Distributed Computing, Linz*, 2007.

[4] VISIT. [Online]. Available: http://www.fz-juelich.de/zam/visit

[5] R. Ratering *et al.*, “GridBeans: Supporting e-Science and Grid Applications,” in *Proc. of 2nd IEEE e-Science, Amsterdam*, 2006.

[6] R. Haber *et al.*, “Visualization Idioms: A conceptual model for scientifi c visualization systems,” *Vis. in Scientifi c Computing*, pp. 74–93.

[7] M. Riedel *et al.*, “Enhancing Scientifi c Workfbws with Secure Shell Functionality in UNICORE Grids,” in *Proc. of 1st IEEE e-Science, Melbourne*, 2005.

[8] I. Foster *et al.*, *The Open Grid Services Architecture V.1.5.* OGF (GFD80), 2006.

[9] WSRF-Technical Committee. [Online]. Available: http://www.oasis-open.org/committees/wsrf/

[10] M. Riedel *et al.*, “Standardization Processes of the UNICORE Grid System,” in *Proceedings of 1st Austrian Grid Symposium, Linz*, 2005, pp. 191–203.

[11] R. Menday, “The Web Services Architecture and the UNICORE Gateway,” in *Proc. of the Int. Conf. on Internet and Web Applications and Services*, 2006.

[12] A. Streit *et al.*, “UNICORE - From Project Results to Production Grids,” *Grid Computing: The New Frontiers of High Performance Processing, Advances in Parallel Computing*, vol. 14, pp. 357–376, 2005.

[13] M. Gudgin *et al.*, *SOAP Version 1.2 Part 1: Messaging Framework.* W3C Recommendation, 2003.

[14] T. Eickermann *et al.*, “Steering UNICORE Applications with VISIT,” *Phil. Transactions of the Royal Society*, vol. 363, pp. 1855–1865, 2005.

[15] S. Pfalzner and P. Gibbon, *Many-Body Tree Methods in Physics.* Cambridge University Press, 1996, ISBN-10: 0521019168.

[16] Xnbody. [Online]. Available: http://www.fz-juelich.de/zam/xnbody

[17] D-Grid. [Online]. Available: http://www.d-grid.de/

[18] J. Cohen *et al.*, “RealityGrid: An Integrated Approach to Middleware through ICENI,” *Phil. Transactions of the Royal Society*, vol. 363, pp. 1817–1827, 2005.

[19] T. Koeckerbauer *et al.*, “GVid - Video Coding and Encryption for Advanced Grid Visualization,” in *Proc. of 1st Austrian Grid Symposium, Linz*, 2005, pp. 204–218.

[20] P. Kleijer *et al.*, “API for Grid Based Visualization Systems,” *GGF 12 Workshop on Grid Application Programming Interfaces*, 2004.

[21] K. Brodlie *et al.*, “Distributed and Collaborative Visualization,” *Computer Graphics Forum*, vol. 23, 2004.