

Towards E-Learning Grids: Using Grid Computing in Electronic Learning

Victor Pankratius
AIFB Institute
University of Karlsruhe
D-76128 Karlsruhe, Germany
pankratius@aifb.uni-karlsruhe.de

Gottfried Vossen
Dept. of Information Systems
University of Münster
D-48149 Münster, Germany
vossen@uni-muenster.de

Abstract

E-learning has been a topic of increasing interest in recent years, due mainly to the fact that increased scheduling flexibility as well as tool support can now be offered at a widely affordable level. As a result, many e-learning platforms and systems have been developed and commercialized; these are based on client-server, peer-to-peer, or, more recently, Web service architectures, with a major drawback being their limitations in scalability, availability, and distribution of computing power as well as storage capabilities. This paper tries to remedy this situation by proposing the use of grid computing in the context of e-learning. In particular, it is shown what advantages a utilization of grid computing may have to offer and which applications could benefit from it. Moreover, an architecture for an e-learning grid is outlined, and the notion of a grid learning object is introduced.

1 Introduction

Electronic learning (“e-learning”) has been a topic of increasing interest in recent years [1], due mainly to the fact that increased scheduling flexibility as well as tool support can now be offered at a widely affordable level, both with respect to technical prerequisites and pricing. At the same time, learning content can be made available even in remote places and without the need to travel to the site where content is delivered. As a result, many e-learning platforms and systems have been developed and commercialized; these are based on client-server, on peer-to-peer, or, more recently, on Web service architectures [33], with a major drawback being their limitations in scalability, availability, and distribution of computing power as well as storage capabilities. Thus, e-learning is currently deployed mostly in areas where high requirements in any of these aspects are not

mission-critical, which excludes its exploitation, for example, in most natural sciences or medical areas. This paper tries to open a new door for electronic learning, by proposing the use of *grid computing* in the context of e-learning. In particular, it is shown what advantages a utilization of grid computing may have to offer and which applications could benefit from it. Moreover, an architecture for an *e-learning grid* is outlined, and the notion of a *grid learning object* is introduced as a first step towards the realization of this novel concept.

As a typical example where currently e-learning systems reach their limits, consider a medical school where anatomy students examine the human body and prepare for practical exercises. Up to now, it is vastly impossible to compute, say, photo-realistic visualizations of a complex body model in real-time and display the computation result on a remote screen. With the advanced functionality of an e-learning grid, students could be provided with the possibility to grab, deform, and cut model elements (e.g., organs) with the click of a mouse. Basically as before, the e-learning system could support the learner by giving advice on how to cut or give feedback for the actions, but beyond that virtual reality at local machine would become possible and improve the understanding of the subject considerably.

The organization of the paper is as follows: In Section 2 we collect several fundamentals that are considered prerequisites for our work; these stem from the areas of e-learning as well as grid computing. In Section 3 we begin to combine the two, and we outline an architecture that shows in some level of detail how a learning management system can be interfaced with grid middleware in such a way that an e-learning grid capable of handling new types of learning content results; the main ingredients upon which activities inside this grid are based is the grid learning object, a notion that is introduced as well. In Section 4, we indicate how an implementation of a grid application for some photo-realistic e-learning visualization could be obtained, thereby

showing the feasibility of our approach. Finally, we summarize our work and draw some conclusions in Section 5. We mention that this paper is based on [27], in which further details and explanations can be found.

2 Fundamentals

In this section we introduce the fundamentals relevant to our work, which falls into two categories: e-learning and grid computing. We consider each area in turn.

2.1 E-Learning Fundamentals

E-learning has been a topic of increasing interest for a number of years now, and a consolidation w.r.t. several fundamental notions has meanwhile been achieved. A general agreement seems to exist regarding roles played by people in a learning environment as well as regarding the core functionality of modern e-learning platforms; we refer the reader to [1] for a detail account. The main players in these systems are the *learners* and the *authors*; others include trainers and administrators. Authors (which may be teachers or instructional designers) create content, which is stored under the control of a *learning management system* (LMS) and typically in a database [31]. Existing content can be updated, and it can also be exchanged with other systems. A learning management system as shown in Figure 1 is under the control of an administrator, and it interacts with a runtime environment which is addressed by learners, who in turn may be coached by a trainer. Importantly, these three components of an e-learning system can be logically and physically distributed, i.e., installed on distinct machines, and provided by different vendors or content suppliers. In order to make such a distribution feasible, standards (e.g., IMS and SCORM) try to ensure plug-and-play compatibility [22].

E-learning systems often do not address just a special kind of learner, but may rather be implemented in such a way that a customization of features and appearance to a particular learner's needs is supported. Learners vary significantly in their prerequisites, their abilities, their goals for approaching a learning system, their pace of learning, their way of learning, and the time (and money) they are able to spend on learning. Thus, the target group of learners is typically very heterogeneous; a system is ideally able to provide and present content for all (or at least several of) these groups, in order to be suitable, for example, for a student who wants to learn about database concepts or for a company employee who wants to become familiar with company-internal processes and their execution. To fulfill the needs of a flexible system as sketched, a learning platform has to meet a number of requirements, including the integration of a variety of materials, the potential deviation

from predetermined sequences of actions [10], personalization and adaptation, and the verifiability of work and accomplishments [32].

Content consumed by learners and created by authors is commonly handled, stored, and exchanged in units of *learning objects* (LOs). Basically, LOs are units of study, exercise, or practice that can be consumed in a single session, and they represent reusable granules that can be authored independently of the delivery medium and be accessed dynamically, e.g., over the Web [31]. Learning objects can be stored in a database and are typically broken down into a collection of attributes [31]. In a similar way, other information relevant to a learning system (e.g., learner personal data, learner profiles, course maps, LO sequencing or presentation information, general user data, etc.) can be mapped to common database structures. This does not only render interoperability feasible, but also allows for process or even workflow support inside an e-learning system. Indeed, as has been shown, for example, in [32] e-learning consists of a multiplicity of complex activities such as content authoring or learner tracking and administration which interact with resources (including people such as learners and authors), with one another (some activities trigger others), and with the outside world (such as existing software systems) in a predefined way. If these activities are modeled as processes or workflows that operate on and manipulate learner and learning objects, and if they are then attributed to and associated with the various components of a learning platform, a workflow management system can be employed to control these activities. Thus, it becomes possible, for example, to track the work and performance of a learner automatically, or to deliver content or process feedback. This idea can be taken to higher levels as well; for example, one can think of a college degree program that is fully supervised by an electronic system.

If a process view or even workflow management is accepted as fundamental modeling and enactment paradigm, it is a straightforward task to turn this kind of learning, at least for certain situations and learner groups, into a collection of *Web services* that handle content and course offerings as well as other LMS processes, as illustrated in Figure 3. We briefly introduce the Web service paradigm next.

2.2 Web Services

In essence, Web services are independent software components that use the Internet as a communication and composition infrastructure. They abstract from the view of specific computers and provide a service-oriented view by using a standardized stack of protocols. To specify the operations supported by a Web service, the *Web Services Description Language* (WSDL) is used [9]. The *Simple Object Access Protocol* (SOAP) is used to exchange structured

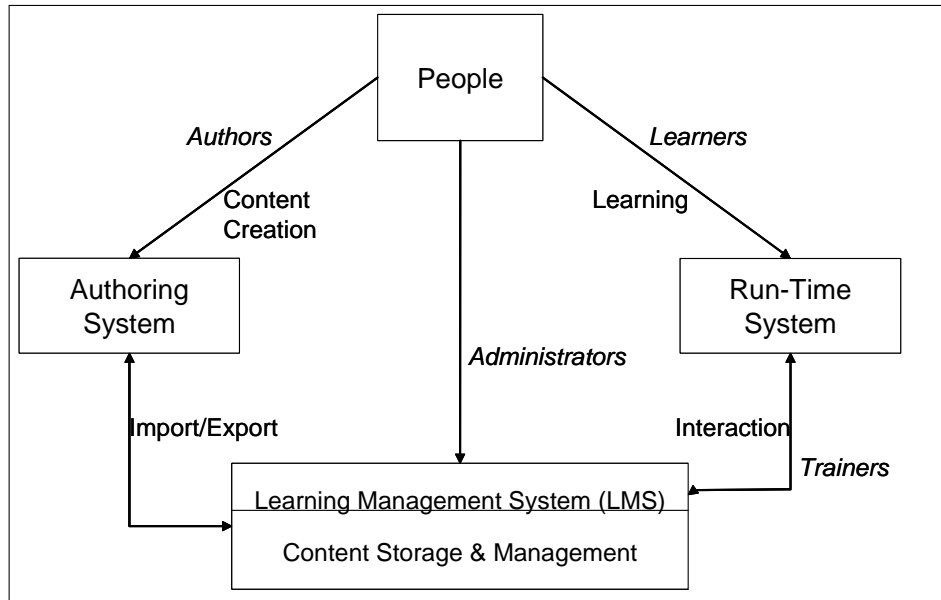


Figure 1. Generic View of a Learning Management System.

data over the Web by building upon the HTTP protocol [7]. To discover new services on the Web, or to publish existing ones, the *Universal Description Discovery and Integration Protocol* (UDDI) is employed [5]. More complex Web services can be composed out of existing services using *BPEL4WS* [2].

In Figure 2, the typical steps of an invocation of a Web service are shown. In a first step, suppose that a client needs to find a Web service which provides a specific functionality. This is done by contacting a UDDI registry (step 1), which returns the name of a server (service provider) where an appropriate Web service is hosted (step 2). Since the client still does not know how to invoke the desired service, a WSDL description is requested which contains the name and the parameters of the operation(s) of the service (steps 3 and 4). The client is now able to invoke the service using the SOAP protocol, which essentially puts the data in an envelope and sends it over the Web by using HTTP. The service provider receives the request and executes the desired operation(s) on behalf of that client. The results are finally sent back to the client by using SOAP over HTTP again (step 6).

The aforementioned view of making e-learning offerings available as Web services has recently been developed in [33]. As described there, the various functionalities of an e-learning system can be decomposed and be made available as Web services individually. The decomposition is based on an appropriate modeling of the processes, objects, and their interactions as they occur in such a platform [32]; fi-

nally, the results need to be mapped to distinct services (and corresponding providers) as desired. E-learning functions result from adequate service compositions.

2.3 Basics of Grid Computing

The *grid computing* paradigm [13, 8] essentially aggregates the view on existing hardware and software resources. The term “grid” alludes to an electrical power grid and emphasizes the perception that access to computational resources should be as easy as the common access to a power grid. In addition, a grid user should not have to care where the computational power he or she is currently using comes from, just like in a power grid where a user does not care by which plant the electricity is produced which is currently used.

A grid unifies many different resources like computers with CPUs and storage, remotely controlled instruments (e.g., electron microscopes, radio telescopes), and visualization environments by using uniform interfaces. Furthermore, it is possible to distribute computations and data across all computers in a grid. This is done in a transparent way so that users of the grid are not aware of it. Thus, available resources can be used more efficiently and the aggregation of idle CPU cycles can even lead to supercomputing capabilities.

Grid computing has its origins in wide-area distributed computing, in particular in meta-computing, where geographically distributed computers are unified in a way that

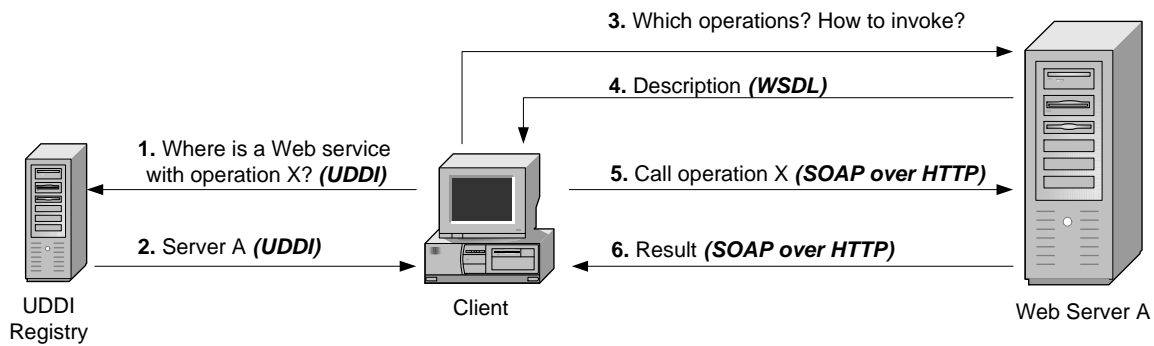


Figure 2. Invocation Steps of a Web Service.

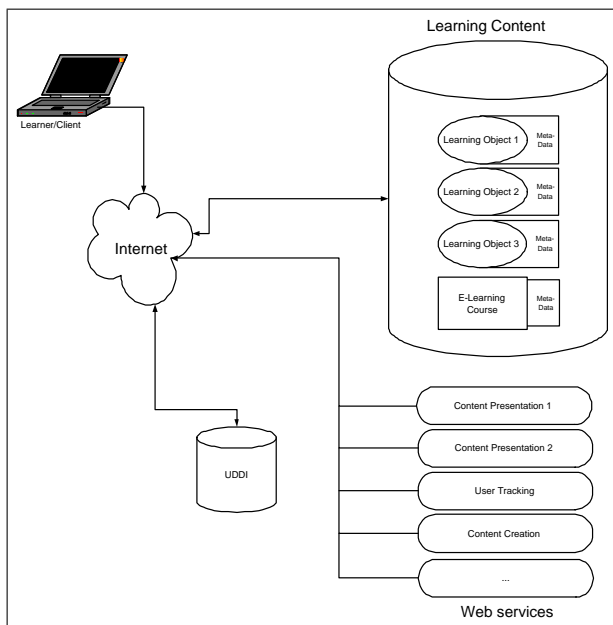


Figure 3. E-learning as a Web Service.

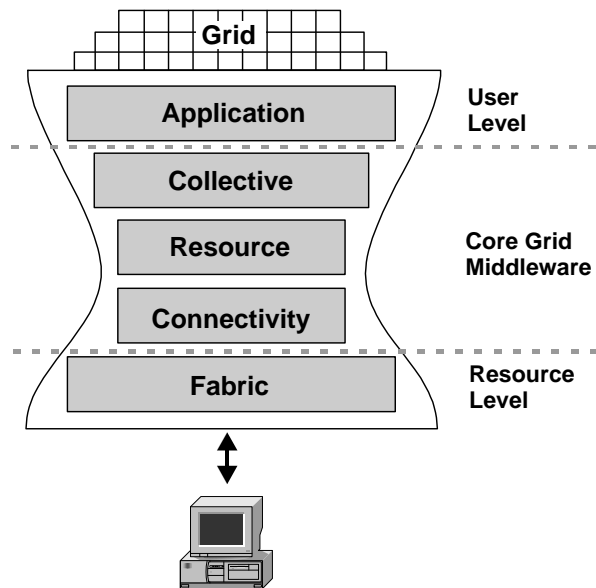


Figure 4. Layers in a Grid Middleware.

they can be perceived as one big powerful computer [12]. Grid computing extends this view to a large-scale “flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions, and resources” [15]. Modern grids also focus on scalability and adaptability and therefore adopt Web technologies and standards such as the Extensible Markup Language (XML) or Web services. During the evolution of grid computing, two basic types of grids have merged: Computational grids and data grids. Computational grids mainly focus on the distribution of computa-

tion among computers in a grid, while data grids are tailored towards data intensive applications which handle petabyte-sized data in a grid [11]. Although these two types of grids have evolved, some authors argue that a convergence between computational grids and data grids can be observed [3, 23].

Grids are typically implemented as a form of middleware which provides all grid-related services and can also use the Internet as a communication infrastructure. In [15]

a general protocol architecture for grids is proposed which is depicted in Figure 4. As can be seen, the middleware contains five layers. The lowest *fabric layer* implements uniform interfaces for access to resources, such as computers, storage media, or instruments in a grid and guarantees interoperability. The usage restrictions of a resource within the grid can be specified in security policy files. The fabric layer should be made available locally on each resource to enable communication with the grid. The next higher *connectivity layer* defines communication and security protocols for the grid. By using cryptographic authentication algorithms a “single-sign-on” can be implemented, so that a user has to authenticate only once in the grid and use any resources later on. The security protocols cooperate with the local security protocols of each resource and do not replace them. Next, the *resource layer* implements access to single resources and monitors their status. This functionality is required in the *collective layer*, which coordinates global interactions among collections of resources. In this layer brokers and schedulers distribute computations or data on the grid. Finally, the *application layer* contains grid applications which build upon the underlying layers of the core grid middleware. This layer can change depending on user programs, while the underlying layers always provide the same functionality. The hourglass-shaped outline of Figure 4 suggests that the number of protocols and services should be minimized in the core grid middleware in order to maximize interoperability. The wide bottom and of the hourglass implies that the fabric layer should support as many resource types as possible. Similarly, many applications should be available at the top.

Grid middleware along the lines just described has already been implemented in various projects, including Globus [14], Legion [19], or Condor-G [17], to name just a few. The *Globus Toolkit* Version 3 with its *Open Grid Services Architecture* (OGSA) [16] currently has the biggest influence on the development of grids. Its philosophy is to provide a bag of services and specifications which can be combined individually to form a grid middleware. The component model of Globus Version 3 is based on grid services which are actually Web services with specific extensions (e.g., interfaces) for use in grids. Grid services can be implemented in nearly any programming language and can use the Internet as an underlying communication infrastructure. In addition, grid services can also create instances of themselves. The idea behind OGSA is to construct each of the grid middleware layers described above by using appropriate grid services.

Since Web services are being employed in the context of grid computing already, and since certain e-learning applications also render the use of Web services suitably, it is near at hand to consider combining the two, which will be the subject of the next section.

3 E-Learning Grids

Complex applications which are computationally intensive and handle large data sets have been systematically ignored in the context of e-learning up to now, due mainly to technical feasibility problems and prohibitively high costs. As will be demonstrated in the remainder of this paper, grid computing can close this gap and enable new types of e-learning applications, such as photo-realistic visualizations or complex real-time simulations. Computations and data could be distributed on a grid as soon as desktop computers of learners cannot handle them anymore. This particularly pertains to university environments where the hardware infrastructure already exists, since there are often hundreds of networked computers, ranging from PCs to supercomputers, which most of the time are not working to capacity or even run idle. We therefore propose the creation of *e-learning grids* in which grid computing functionality is integrated into e-learning systems.

As we have mentioned in the Introduction, there are many conceivable applications for *e-learning grids*. Medicine students could use photo-realistic visualizations of a complex model of the human body to prepare for practical exercises. Such visualizations, computed in real-time, could improve the understanding of the three-dimensional locations of bones, muscles, or organs. Students should be able to rotate and zoom into the model and get additional information by clicking on each element of the model. With more advanced functionality such as virtual surgery, students could be provided with the possibility to grab, deform, and cut model elements (e.g., organs) with the click of a mouse. In biology courses the ability of grids to integrate heterogeneous resources could be used to integrate an electron microscope into the grid. We mention that the technical feasibility of this approach has already been demonstrated in the TeleScience project [30]. However, this project could be widely extended to integrate the controls and output of the electron microscope into a learning environment so that students can be assigned tasks or read subject-related texts while operating the microscope. Similarly, in engineering courses complex simulations, e.g., in a wind channel, can be made accessible for each student by using grids.

We next outline an architecture for *e-learning grids*. To demonstrate the technical feasibility, the architecture will be kept as simple as possible. It contains a Learning Management System (LMS) as well as grid middleware, which are both based on Web services and grid services, respectively (cf. Figure 5). In this figure, grid and Web services are depicted as rectangles containing a name as well as the most important operations. Note that grid services (with grey name fields) can easily be distinguished from Web services. The LMS interacts transparently with the grid middleware so that a learner is not aware of the grid. Furthermore, the

architecture is designed in such a way that a learner only needs a Java-enabled Internet browser to use both the LMS and the grid. We explain the architecture shown in Figure 5 as well as most of the operations listed in this figure in more detail in the following subsections.; we begin with the upper half (Core Grid Middleware) and then turn to the lower one (LMS).

3.1 Core Grid Middleware

The grid middleware of an *e-learning grid* implements the various layers shown above in Figure 4. The *fabric layer* is implemented as a Java applet, which provides uniform interfaces to all resources in the grid. For the time being, we assume for simplicity that there are only computers and no specialized instruments like electron microscopes in the grid. We will explain possible extensions later. Furthermore, locally available policy files specify usage restrictions for computers in the grid (e.g., maximal CPU load, usage hours, etc.); finally, meta-data describe each resource type. The fabric layer applet has to be started on each computer that participates in sharing computational capacity and storage space in the grid. This can be done while a user accesses a Web page with his or her Web browser to authenticate in the grid.

The *connectivity layer* consists of a *grid login service*. This service needs to have access to a database in which user information and access rights are stored together with a hash value of the password. The service can create new users with a `newGridUser` operation and delete users with corresponding `delGridUser` operation. When a grid user wants to enter the grid, an operation `Grid_login` is called, which checks the user's login name and password against the values stored in the database. If the login was successful, the computer is registered with `registerClient` in a `ClientsRegistry`, the latter of which contains a list of all resources currently available in the grid. Furthermore, a *broker* is assigned which can handle requests to distribute computation or data across other computers in the grid. When a client logs out, the `ClientsRegistry` is updated through a `deregisterClient` operation. The login service uses an authentication mechanism based, for example, on Kerberos v. 5 [24]. It uses a symmetric cryptographic algorithm and requires that each entity in the grid (i.e., grid service or computer) must have a specific key which must be known to the grid login service. For all computers of a grid, the hash-value of the user password could be used as a key, while grid services have a key generated and registered at the login service by the respective developers. The idea of the authentication algorithm, whose details are beyond the scope of this paper, is that a computer first requests a ticket-granting ticket (TGT). If a TGT is received, the authentication was

successful and further session tickets can be requested for the communication with any other grid service. Before a computer can access the operation of a grid service, a valid session ticket must be presented, which is checked by the respective service. This procedure is executed every time an operation of a grid service is accessed, so that an additional graphical representation is omitted in Fig. 5.

The *resource layer* contains an *information service* which is aware of the status and type of all resources in the grid. By accessing the clients registry it firstly determines which computers are available. It then requests from each computer some status information (e.g., CPU load, unused storage space) with a `getStatus` operation, the resource type with `getResType` and the usage restrictions with `getPolicyFile`. The list with the collected information can be accessed with `showAllStatus` and is used to schedule computations or distribute data in the collective layer.

In the *collective layer* there is a *broker, replica management and replica selection service*. For simplicity, we assume that there is only one copy of each service. The broker implements a grid scheduling algorithm [21] and is responsible for distributing computations and data across the grid. The broker has to register in a `BrokerRegistry`, so that the grid login service and grid applications in the application layer can find it. For the distribution of data it uses the replica management and replica selection service, which implement the functionality that is typical of data grids. We assume here that the data to be replicated is read-only. The replica management service can create replicas of data on computers with the operation `copyData`. When data is replicated, an entry is added to the `ReplicaRegistry` with the exact information which parts of data were replicated on which machines. When replicated data is deleted with `deleteData` the corresponding entry in the registry has to be erased with `de-registerReplica`. With the operation `findReplica` of the replica selection service existing replicas can be found when data is accessed.

All assumptions made so far can be relaxed, which leads to a loss of simplicity of the architecture. The fabric layer applet could be extended to support instruments, like electron microscopes. In this case, the information and broker service would have to be adapted since instruments cannot execute computations or store data. Furthermore, there could be more than one service of each type. For example, several grid login services could be responsible for disjoint authentication domains. Clients could look up in a `LoginRegistry` which login service is responsible for the domain they are located in. A problem arises when clients need to access resources of different authentication domains, which requires cooperation between grid login services. It is also possible to have information, broker, and replica services for each authentication domain to increase

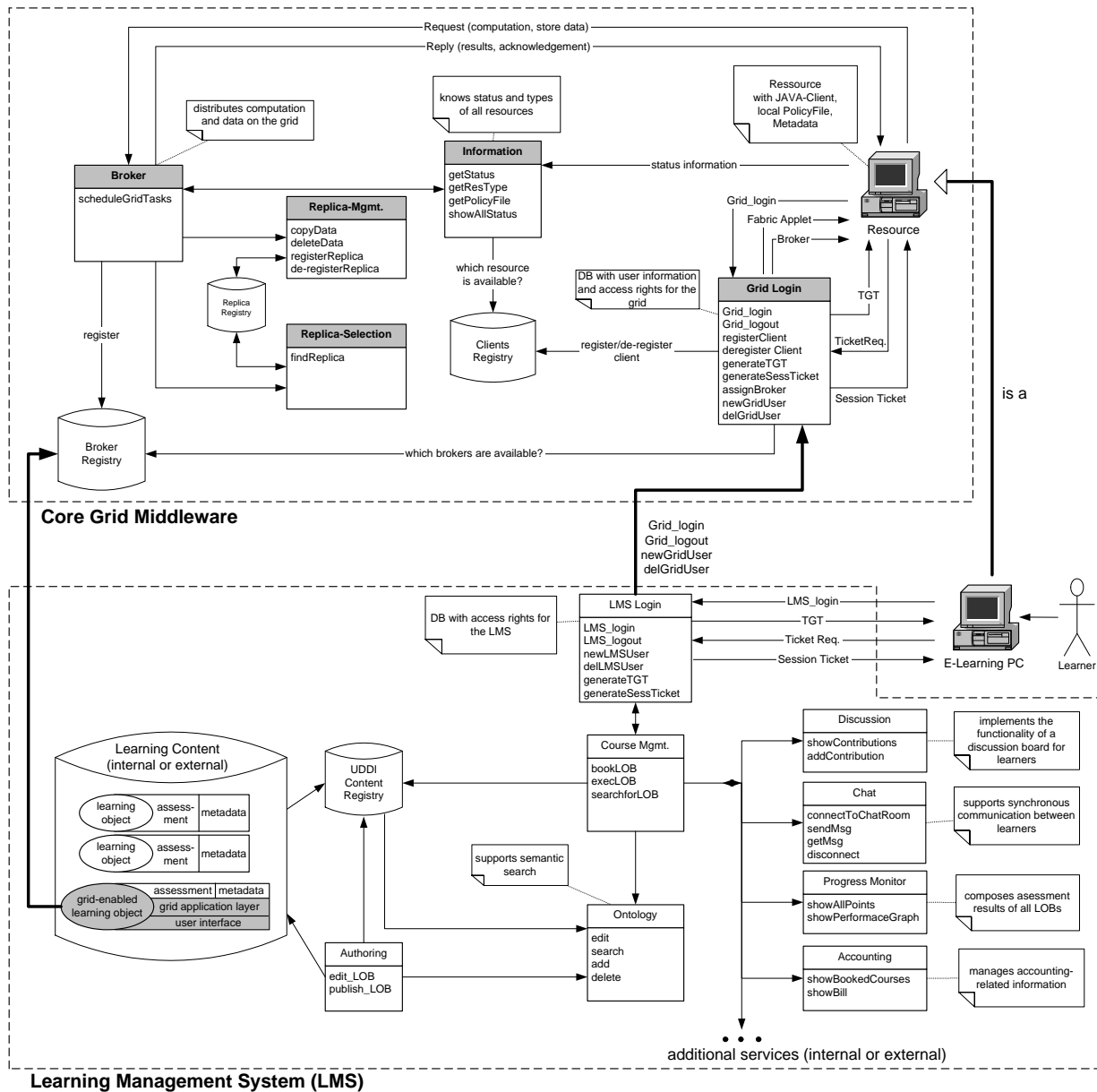


Figure 5. Architecture of an E-Learning Grid.

the efficiency in large grids. Similarly to the problems encountered at the login services, a cooperation between all information, broker, and replica services of each domain could be necessary.

3.2 Learning Management System (LMS)

An LMS generally coordinates all learning-related activities. We assume that the entire LMS functionality including the learning contents are implemented as Web services. A learner who typically uses a PC for a learning session interacts directly only with the LMS and is not aware of a grid. The LMS offers both content which makes use of the grid as well as content that does not need grid functionality. The integration between LMS and grid will be described in the next subsection. All Web services of the LMS are accessed via Web pages, so that the learner only needs a Web browser to utilize the LMS.

In a first step the learner has to authenticate in the LMS, which is done by an *LMS login service*. This service is similar to the *grid login service*, i.e., it draws on a database with access rights and uses the same authentication mechanism for the LMS. When the learner is logged in and authenticated, he or she can access a Web page for course management, the functionality of which is implemented in a *course management service*. The learner can look for suitable courses with a *searchLOB* operation, which searches for learning objects in a *ContentRegistry*. The *bookLOB* operation is called to enroll for a course. A class can be attended by calling the *execLOB* operation.

An *ontology service* supports the semantic search for courses. It basically contains an ontology defining semantic relationships between Web services that provide learning objects. Thus, for a search term like "programming" it possible to obtain results like "C++," "Java," or "Prolog." The ontology service provides operations to add, delete, edit, or search for entries in the ontology. Next, the *authoring service* provides an environment to create, edit, and publish e-learning content in the *ContentRegistry*, so that they can be found by the LMS. In addition, entries can be added to the ontology for the semantic retrieval of content.

The Web services which provide *e-learning content* consist of three fundamental components. The first part is a learning object, which is typically a lesson (e.g., in HTML or XML format) or a course consisting of several learning objects. The assessment part defines online-tests so that students or teachers can check whether a lesson is well-understood and the material mastered. The last part contains meta-data for search engines that describes the content in a standardized way. The functionality of the grid is used with *grid learning objects*, which also integrate a grid application layer and a user interface and will be described in the next subsection.

The LMS also comprises other services. In *discussion boards* or *chat rooms* learners can interact with instructors or other learners and ask questions. A *progress monitor* composes the assessment results from all lessons into a general overview; this can also be used to create certificates. An *accounting service* manages all processes which are related to financial aspects. It shows, for example, all booked courses or the bill that has to be paid by an individual. Finally, it should be mentioned that the functionality of the LMS can be extended by other Web services, which can either be provided internally (i.e., in a local network) or externally from other suppliers over the Web. The flexibility of Web services also allows to distribute services of the LMS on different machines in a local network or over the Web.

3.3 Integration of Grid Middleware and LMS

After having discussed the functionalities of grid middleware and LMS in the previous subsection, resp., we will explain their integration next.

The *e-learning PC* is used by learners to access the LMS. At the same time, such a PC can also be used as a resource for the grid. This has been modeled by the "is a" relationship in Figure 5 which also illustrates that at the same time not every resource of the grid needs to have access to the LMS.

The *LMS login service* makes it possible for the *e-learning PC* to become a resource in the grid. When the learner authenticates himself on the Web page that is connected to the login service of the LMS, the fabric layer applet of the grid can be transferred as *mobile code* and be started locally on the *e-learning PC*. This enables communication with the grid. The *LMS login service* transparently calls the *Grid_login* operation of the *grid login service* with the data and the PC of the user as parameters. This completes the authentication process of the *e-learning PC* in the grid. If a user of the LMS is not registered at the grid login service, the LMS login service could be given the authority to create new users in the grid login service, based on a trust relationship between the two services. These steps keep the login procedure of the grid completely in the background, so that the learner is not aware of it.

3.4 Grid Learning Objects (GLOBs)

We propose the notion of a *grid learning objects* (GLOB) for using the grid in e-learning applications. A GLOB extends the functionality of a "traditional" learning object (in the sense of [4, 31]) by adding grid functionality consisting of a specific *grid application layer* (see Figure 4) and a *user interface*. The structure of a GLOB is depicted in Figure 6 and is designed in such a way that it can contain both conventional e-learning content and content that uses

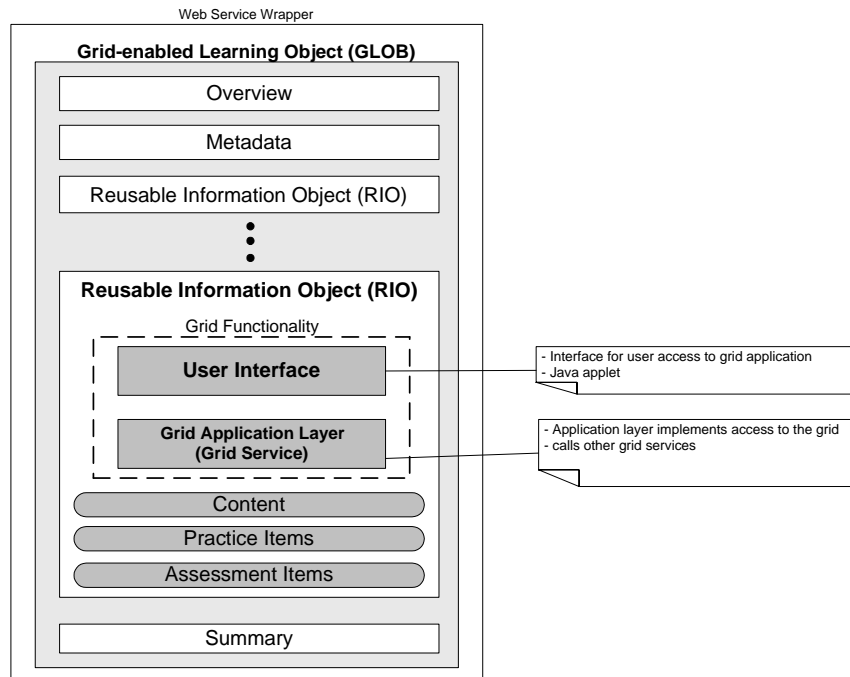


Figure 6. Structure of a Grid Learning Object (GLOB).

grid functionality. Basically, a GLOB is wrapped by a Web service which makes it possible to easily integrate it into the LMS (see above). In addition, the Web service provides operations to access individual elements of the GLOB, to transform content (e.g., from XML to HTML), or to generate online tests.

The design of a GLOB is based on [4] and consists of several parts: An *overview* of the lesson, *metadata*, which is used to find GLOBs, several *reusable information objects (RIOs)*, and a *summary*. A RIO contains a *content* part with the actual content of a unit of study. The content can either be stored in a format such as XML, HTML or, as already mentioned in Section 2, in a database. *Practice items* can be used to generate online exercises for the learners. *Assessment items* are used to generate online tests for final exams.

An RIO may additionally contain grid functionality which is implemented as a grid service of the *grid application layer* and accessed via a *user interface*. The grid application layer implements grid applications for e-learning and uses the underlying layers of the core grid middleware. While the layers of the core grid middleware remain stable, the application layer may change depending on the application. To be executed, the grid service of the application layer first has to be extracted from an RIO by using a Web service operation. It then has to be started on a computer that provides the hosting environment of the

Globus Toolkit v. 3, which also supports the creation of instances of grid services. We will call such a computer a *grid application server*. During an initialization procedure of the grid service, a grid broker has to be found via the *BrokerRegistry*, so that it can be assigned tasks such as the distribution of computations or data on the grid. In addition, the grid service of the application layer is responsible for the composition of the results returned by the broker.

The *user interface* can be implemented as a Java applet that transforms user input (e.g., mouse clicks) into tasks for the grid service in the application layer (e.g., a request to recalculate a 3D model). If the learning content is available in HTML format, the interface applet can be placed anywhere inside the content. When a user accesses such a lesson, the applet is automatically loaded onto his local *e-learning PC*. In this way no other administrative actions have to be taken, and a learner is not aware that he or she is using the grid. If the learning content is available in other formats (e.g., XML), Web service operations can be provided to transform it into HTML and embed the interface applet. Furthermore, Web service operations can be used to integrate the user interface applet into online tests generated from practice and assessment items.

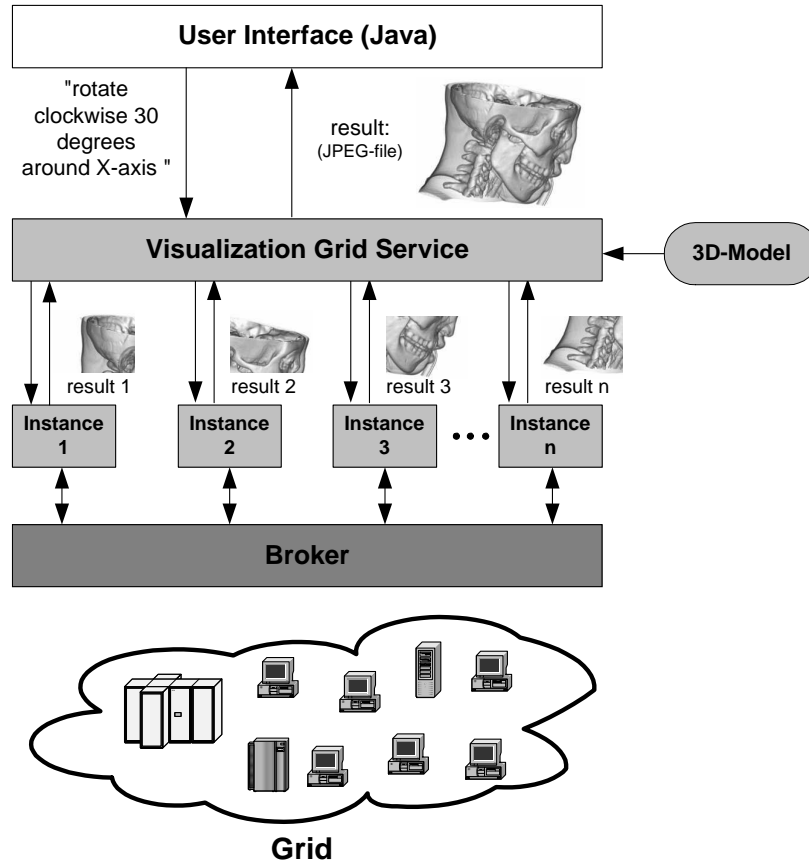


Figure 7. Layers of a Visualization Application for an E-Learning Grid.

4 Towards an Implementation of an Application Layer for Visualization

To show the feasibility of the concepts developed in the previous section, and to illustrate the interaction between grid middleware, grid application layer and user interface, we will now outline a possible implementation of a grid application for photo-realistic visualization that can be used for e-learning courses in medicine. To this end, we assume that learners are to be provided with an interactive component visualizing in a photo-realistic way complex models of the human body with elements like bones, muscles, or organs. By clicking with the mouse, learners can rotate or zoom into the model or get explanatory information on certain elements. Furthermore, learners can suppress the rendering of different layers of the model (e.g. skin, blood vessels, muscles, etc.).

The relevant layers for an implementation of this scenario are shown in Figure 7. The user interface is imple-

mented as a Java applet and runs inside an Internet browser of a learner. The visualization component is implemented as a grid service, runs on a grid application server and represents the grid application layer. The broker provides access to the grid, which consists of many networked resources. The user interface captures mouse clicks and movements and transforms them into commands for the visualization service (e.g., rotate model clockwise 30° around X-axis and display it again). After performing the necessary calculations, the visualization service returns an image file (e.g., in JPEG format) that can be displayed by the user interface.

The visualization service implements a computer graphics algorithm (e.g., ray tracing [18]) to render model data in a photo-realistic way. It uses the commands of the user interface, a model file and the coordinates of the upper left and lower right corners of a rendering window to compute a graphics file as output. The model data is found in a file which may contain polygon and texture descriptions or finite-element models of the human body. Before the computation of the output file begins, the area inside the render-

ing window is divided into several disjoint regions. For each region, an instance of the visualization service is created. This is basically a copy of the visualization service where the parameters of the rendering window are adjusted to the respective region. An instance has to calculate the color for every pixel of its region depending on parameters like view angle, texture, lighting, etc. The calculations for each pixel represent a task that is submitted to the grid broker. The broker distributes the computations on the grid and returns the results (i.e. color of a pixel) back to the instance which submitted the task. When the calculations of all regions are finished, the results are returned to the visualization service which composes them to the final result. Although this application is computationally intensive, it can be observed that network traffic is low.

For the implementation of the visualization service, available libraries like `vtk` [29] or `VisAd` [20], which offer a wide range of graphics algorithms, could be used. In the 3D model file different categories such as data for skin, muscles or blood vessels need to be distinguished. To preserve interoperability, the format could be based on XML (see also X3D [34]). This would render the use of recent query languages such as XQuery [6] possible, which can be used to select only those model categories for rendering which have not been suppressed by the user. In addition, XML offers the possibility to easily add explanatory texts associated to model categories or elements. If a learner requests more information by clicking on a certain group of polygons (e.g., a bone), the user interface applet and the visualization service can identify the model element and extract the textual information from the model file by using a query. Since a model file may become huge, it could be stored on a server and identified with a unique URL. The advantage is that model data does not have to be duplicated for each instance that is created by the visualization service. Instead, the visualization service only passes the URL reference of the model file.

Finally, it should be mentioned that anatomically correct models can be obtained by using the computed tomography technique [25, 28]. The functionality of the visualization service can even be extended to support virtual surgeries, which would be in line with recent developments in the intersection of tele-medicine and Web services. To give learners the possibility to grab, deform, or cut model elements, a simulation component and physical models have to be added. These models specify the physical properties of model elements, e.g., how skin or organs can be deformed when touched, permissible positions, etc. More details for the implementation of virtual surgeries can be found in [25] as well as in [26].

5 Conclusions

In this paper, we have argued that e-learning and learning management systems on the one hand and grid computing on the other, which have been considered and developed separately in the past, can fruitfully be brought together, in particular for applications or learning scenarios where either high computational power is needed or the tool sets on which learning should be done are too expensive to be given out to each and every learner. Beyond making a case for its feasibility and more importantly, we have outlined in detail an architecture for an e-learning grid which integrates core grid middleware and LMS functionality appropriately. Finally, we have indicated how an e-learning grid could be realized on the basis of suitably designed grid learning objects.

Clearly, what has been described in this paper is intended mainly as an attempt to draw up a research agenda for an exploitation of grid computing in e-learning, and many details remain to be filled in. Yet it appears feasible to pursue the area, as there is considerable hope for being able to extend the achievements of electronic learning beyond the limits of individual computers. Future issues to be studied include, for example, transactional guarantees for service executions over a grid such as atomicity, or recovery protocols that help restore an operational state after a grid failure (a problem not to be neglected, as learned from the recent northeastern US power grid failure).

Acknowledgements. Most of this work was done while the first author was with the Department of Information Systems, University of Münster in Germany and the second author was with the Department of Management Systems, University of Waikato in Hamilton, New Zealand; we thank both institutions for their support and hospitality.

References

- [1] Adelsberger, H.H., B. Collis, J.M. Pawlowski, eds. *Handbook on Information Technologies for Education and Training*. Springer-Verlag, Berlin, 2002.
- [2] Andrews, T., F. Curbera, H. Dholakia, et al. "Specification: Business Process Execution Language for Web Services Version 1.1," *IBM developerWorks*, 05 May 2003 <http://www.ibm.com/developerworks/library/ws-bpel/>
- [3] Arnold, D.C., S.S. Vadhiyar, J. Dongarra. "On the Convergence of Computational and Data Grids," *Parallel Processing Letters* 11, 2001, pp. 187-202.
- [4] Barrit, C. "CISCO Systems Reusable Learning Object Strategy - Designing Information and Learning Objects Through Concept, Fact Procedure, Process, and Principle Templates," *Version 4.0. White Paper*, CISCO Systems, Inc., November 2001.

- [5] Bellwood, T., L. Clément, D. Ehnebuske, et al. "UDDI Version 3.0," *UDDI Spec Technical Committee Specification*, 19 July 2002 <http://uddi.org/pubs/uddi-v3.00-published-20020719.htm>
- [6] Boag, S., Chamberlin, D., Fernandez, M.F, Florescu, D., Robie, J., J. Siméon, eds. "XQuery 1.0: An XML Query Language," *W3C Working Draft* 02 May 2003, <http://www.w3.org/TR/2003/WD-xquery-20030502/>
- [7] Box, D., D. Ehnebuske, G. Kakivaya, et al. "Simple Object Access Protocol (SOAP) 1.1," *W3C Note* 08 May 2000 <http://www.w3.org/TR/2000/NOTE-SOAP-20000508>
- [8] Berman, F., Fox, G., Hey, T., eds. *Grid Computing: Making the Global Infrastructure a Reality*, John Wiley and Sons, Inc., New York, 2003.
- [9] Christensen, E., F. Curbera, G. Meredith, S. Weerawarana. "Web Services Description Language (WSDL) 1.1," *W3C Note* 15 March 2001 <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>
- [10] Casati, F., U. Dayal, eds. "Special Issue on Web Services." *IEEE Bulletin of the Technical Committee on Data Engineering*, 25 (4) 2002.
- [11] Chervenak, A., I. Foster, C. Kesselman, C. Salisbury, S. Tuecke. "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets," *J. Network and Computer Applications* 23, 2001, pp. 187–200.
- [12] De Roure, D., M.A. Baker, N.R. Jennings, N.R. Shadbolt. "The Evolution of the Grid," in [8], 2003, pp. 65-100.
- [13] Foster, I., Kesselman, C. *The Grid: Blueprint for a New Computing Infrastructure*, Morgan-Kaufmann Publishers, San Francisco, CA, 1998.
- [14] Foster, I., C. Kesselman. Globus: "A Metacomputing Infrastructure Toolkit," in [13], 1998.
- [15] Foster, I., C. Kesselman, S. Tuecke. "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *Int. J. of Supercomputer Applications and High Performance Computing* 15 (3) 2001, pp. 200-222.
- [16] Foster, I., C. Kesselman, J. Nick, S. Tuecke. "The Physiology of the Grid: Open Grid Services Architecture for Distributed Systems Integration," *Proc. 4th Global Grid Forum (GGF4) Workshop* 2002, <http://www.globus.org/research/papers/ogsa.pdf>
- [17] Frey, J., T. Tannenbaum, I. Foster, M. Livny, S. Tuecke. "Condor-G: a computation management agent for multi-institutional grids," *Proc. 10th IEEE Int. Symp. on High Performance Distributed Computing*, San Francisco, CA, 2001, pp. 55-67.
- [18] Foley, J.D., van Dam, A., Feiner, S.K., J.F. Hughes. *Computer Graphics. Principles and Practice*, 2nd ed., Addison-Wesley, Reading, MA, 1995.
- [19] Grimshaw, A.S., W.A. Wulf. "The Legion Vision of a Worldwide Virtual Computer," *Communications of the ACM* 40 (1) 1997, pp. 39-45.
- [20] Hibbard, W. "Building 3-D User Interface Components Using a Visualization Library," *IEEE Computer Graphics* 36 (1) 2002, pp. 4-7.
- [21] Hamscher V., U. Schwiegelshohn, A. Streit, R. Yahyapour. "Evaluation of Job-Scheduling Strategies for Grid Computing," in R. Buyya and M. Baker (Eds.): *GRID 2000*, LNCS 1971, Springer-Verlag, Berlin, 2000, pp. 191-202.
- [22] IMS Global Learning Consortium, Inc. "IMS Content Packaging Best Practice Guide," Version 1.1.2., 2001.
- [23] Jagatheesan, A., A. Rajasekar. "Data Grid Management Systems," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, San Diego, CA, 2003.
- [24] Kohl, J., B.C. Neuman. "The Kerberos Network Authentication Service (Version 5)," *Internet Engineering Task Force, Request for Comments RFC-1510*, 1993.
- [25] Montgomery, K., Bruyns, C., S. Wildermuth. "A Virtual Environment for Simulated Rat Dissection: A Case Study of Visualization for Astronaut Training," *12th IEEE Visualization Conference*, San Diego, CA, 2001, pp. 509-512.
- [26] Montgomery, K., Heinrichs, L., Bruyns, C., et al. "Surgical Simulator for Hysteroscopy: A Case Study of Visualization in Surgical Training," *12th IEEE Visualization Conference*, San Diego, CA, 2001, pp. 449-452.
- [27] Pankratius, V. *E-Learning Grids: Exploitation of Grid Computing in Electronic Learning*, Masters' Thesis, Dept. of Information Systems, University of Muenster, Germany, 2003.
- [28] Suzuki, N., A. Hattori. "Quantitative Visualization of Human Structure Using Segmented Volume Data Obtained by MRI," *IEEE Journal of Visualization*, 3 (3) 2000, p. 209.
- [29] Schroeder, W., Martin, K., B. Lorensen. *The Visualization Toolkit. An Object-Oriented Approach To 3D Graphics*, 3rd ed., Kitware Inc., New York, 2003.
- [30] Peltier S. Alpha Project: "Telescience for Advanced Tomography Applications," *National Center for Microscopy and Imaging Research*, UC San Diego 2003, https://gridport.npaci.edu/Telescience/general_telescience.pdf
- [31] Vossen, G., P. Jaeschke. "Towards a Uniform and Flexible Data Model for Learning Objects," in Proc. 30th Annual Conf. of the Int. Bus. School Computing Assoc. (IBSCA), Savannah, Georgia, 2002, pp. 99–129.
- [32] Vossen, G., Jaeschke, P., Oberweis, A. "Flexible Workflow Management as a Central E-Learning Support Paradigm," *Proc. 1st European Conference on E-Learning (ECEL)*, Uxbridge, UK, 2002, pp. 253–267.
- [33] Vossen, G., Westerkamp P. "E-Learning as a Web Service (Extended Abstract)," *Proc. 7th Int. Conf. on Database Engineering and Applications (IDEAS)*, Hong Kong, China, 2003, IEEE Computer Society Press.
- [34] Information technology - Computer graphics and image processing - Extensible 3D (X3D), *ISO/IEC 19775:200x*, Edition 1, Stage 40.20, January 6, 2003 http://www.web3d.org/technicalinfo/specifications/ISO_IEC_19775/index.html