Clayworks: A System for Collaborative Real-Time Modeling and High-Performance Simulation

Jens Müller, Martin Alt, Jan Dünnweber, and Sergei Gorlatch
University of Münster, Germany
{jmueller|mnalt|duennweb|gorlatch}@uni-muenster.de

Abstract

Clayworks is a software system which integrates collaborative real-time modeling and distributed computing. It addresses the challenge of developing a collaborative workspace with a seamless access to high-performance servers. Clayworks allows modeling of virtual clay objects and running computation-intensive deformation simulations for objects crashing into each other. To integrate heterogeneous computational resources, we adopted modern Grid middleware and provided the users with an intuitive graphical interface. We parallelized the computation of simulations using a Higher-Order Component (HOC) which abstracts over the Globus Web service resource framework (WSRF) used to interconnect our worksuite to the computation server. Clayworks is a representative of a large class of demanding systems which combine collaborative modeling with performance-critical computations, e.g., crash-tests or simulations for biological population evolution.

1. Introduction

Distributed computing over the Internet has become a broadly used concept in a variety of applications for business, science, engineering, and entertainment. According to John Taylor’s definition [8], eScience “is about global collaboration in key areas of science, and the next generation of infrastructure that will enable it”. Two important approaches in this area are: a) Computer-Supported Collaborative Work (CSCW) environments which allow specialists to work together on a single project from different locations, and b) High-Performance Computing (HPC). We present an infrastructure which combines both approaches, such that usual desktop PCs are interconnected in a CSCW environment and resource-intensive operations are outsourced to specifically configured remote machines.

Our work is motivated by a large class of demanding applications that require distributed worksuites which combine features from both CSCW and HPC areas. While contemporary synchronous CSCW systems provide responsive, soft real-time user interactions and can rely on event- and distributed-objects-based middleware like Java RMI or CORBA, they do not provide adequate mechanisms for running high-performance simulations. In contrast, typical frameworks for distributed HPC and Grid computing [5] like the Globus Toolkit [7] or Unicore [15] provide a transparent access to remote computing resources, but do not provide the means necessary to let several users collaborate and interact in a synchronous way.

We present Clayworks—an application case study that addresses the challenge of integrating the different computation and communication requirements of tightly coupled collaborative work and HPC applications. Existing distributed problem solving environments based on Grids usually do not support synchronous collaborative work; moreover, they often require the end user to be aware of low-level details of the underlying middleware and protocols. An additional goal of Clayworks is, therefore, to hide the complexity of CSCW and the HPC infrastructure from the application users, allowing them to transparently use remote computing resources within a collaborative environment.

We implemented Clayworks as a distributed worksuite allowing to collaboratively model clay objects and execute deformation experiments with them. In the modeling mode, several users concurrently model objects in a shared design workspace using virtual clay. Changes to objects are immediately shown at all user clients, which requires soft real-time communication and computation to provide a high level of responsiveness. In the simulation mode, clay objects are deformed when they crash into each other, which is simulated using a remotely located high-performance server. This way, Clayworks facilitates an integrated and seamless workflow for collaborative modeling and simulation; it exemplifies the main features of a large class of demanding scientific and engineering applications, e.g., CAD, biological evolution, or geophysical simulations.

Section 2 presents Clayworks and its distributed three-tier architecture. Section 3 describes the distributed CSCW
part for modeling clay objects. The deformation algorithm and its parallel implementation on top of the Globus middleware are presented in Section 4. Finally, we discuss related work and conclusions from the development of Clayworks in Section 5.

2. Clayworks

This section gives an overview of our target application and the design of Clayworks. We discuss the three-tier architecture (Client, CSCW-server, parallel computation) designed for the Clayworks implementation. We briefly explain the main operations which can be performed by the users and discuss the integrated workflow of cooperatively modeling clay objects and simulating their deformation in Clayworks.

2.1 Clayworks: Integrating CSCW with HPC

Clayworks is a result of combining techniques from two active research areas: Collaborative environments and HPC. Distributed collaborative applications allow experts from different locations to work together on a single project, while HPC deals with accessing remote computational resources like processing power or storage space in a transparent way. For a large group of applications, the integration of both these concepts into a single worksuite is very promising: CAD engineering, biological simulations or virtual physical experiments require a) an interactive construction part to model the objects of interest and to define and set up the simulation, and b) a resource-intensive simulation part.

However, the use of nowadays’ HPC is still complicated for application developers, because it requires a lot of specific know-how about how to write jobs and configuration files and how to remotely start computations. Furthermore, current problem solving environments built on top of Grids expose a lot of technical details to the end users, who, besides their own area of interest, have to become experts in the area of Grid computing. The general vision of e-Science, therefore, is to provide the “invisible Grid”. Clayworks is a case study for such an integrated “HPC-powered CSCW” application, which allows non-experts to make full use of a remote high-performance server without caring about details of the underlying network infrastructure.

In the CSCW part of Clayworks, several users, each connected via a graphical client to a shared workspace (Fig. 1), model objects made of virtual clay:

- Users can create, delete, merge or modify the shape of objects, which will be immediately visualized at all connected clients.
- Objects can be grouped or locked for exclusive access, such that users can easily split up the work on complex objects among themselves and collaboratively create large scenarios involving a lot of objects.
- Objects can be saved to a database and reloaded later. Even in other workspaces and projects, which allows to build a library of reusable objects.

In the simulation part, the computation of the deformations of moving clay objects is started on a multiprocessor remote server:

- Users assign a velocity and a direction vector to each object and define other simulation parameters.
• For each object, it can be defined whether the object is solid or should be deformable.

• Users can view the result of the simulation as a movie and freely move the camera and scale and rotate the scene.

This allows to execute deformation simulations among the moving objects, for which we used the deformation algorithm described in [3] as a basis. However, since the computation requires a lot of memory and processing time for larger scenarios, we parallelized the algorithm and embedded it into a software component, which we deployed to a Globus WSRF Grid-container. This setup enables the use of a server with a multiprocessor architecture for running the computations efficiently (see Section 4 for details).

Following the 2x2 classification of CSCW systems described by Ellis et al. [4], the interactions of users in a particular CSCW system can be classified to be local/distributed and asynchronous/synchronous. Although Clayworks allows to work asynchronously as well, it mainly aims at providing a synchronous distributed workspace, as we discuss in Section 2.2 in detail. In order to enable synchronous collaborative work, modeling actions of users have to be visualized immediately at other connected clients. These actions do not only have to be carried out with the maximal possible performance, but they are subject to strong real-time constraints [9]. In fact, Clayworks is not used for controlling any physical machinery, like a hard real-time system where deadline violations lead to serious damage, but any timeout is still fatal, as, thereafter, the system state can no longer be maintained correctly. Thus, Clayworks belongs to the most demanding class of applications regarding processing power and computation bandwidth of a distributed CSCW system.

Several other applications like distributed and collaborative CAD/CAE (Computer Aided Design/Engineering) are similar to the virtual clay modeling offered by Clayworks. The simulation part is very relevant to industrial scenarios as well, where, for example, the time-intensive computation of the aerodynamic resistance for a collaboratively modeled car using a similar HPC infrastructure would increase the efficiency of the car’s development.

2.2 Three-tier Architecture of Clayworks

The main challenge in the development of Clayworks was the integration of the CSCW and HPC part, which in some sense have contradictory requirements: The CSCW part is a soft real-time system which requires timely communication and computation for a high responsiveness of the application. However, the computations for the modeling are not very expensive and can be executed on a standard, modern desktop PC. In contrast, the simulation algorithm, which iteratively moves and deforms the clay objects, needs a computer with a very high computational power, but has no real-time requirements for the communication.

In order to integrate these different requirements of the CSCW modeling and HPC in Clayworks, we developed a three-tier architecture, shown in Fig. 2. The server tier is in the middle, consisting of the Clayworks server and a database put in between the user clients and the HPC tier, thus decoupling the clients from the HPC host. The use of a Web service for interconnecting the Clayworks server and the HPC host enables to flexibly exchange the HPC host according to the application requirements without affecting the client, as indicated by the dashed lines in the figure.

**Figure 2. Three-tier Clayworks’ Architecture**

The clients together with the server form the modeling part which is optimized for immediate communication, while the server together with the HPC host(s) constitutes the high-performance computing facility of Clayworks for the deformation simulations. In the following, the three tiers and their respective functionality are briefly discussed:

**Client Tier:** The clients run at the users’ desktop computers and provide an integrated access to all functionality of Clayworks. Users can connect to a shared workspace on a Clayworks server, model objects and observe and discuss the work of other users in real-time, and can start and view simulations.

**Server Tier:** In this additional middle tier between clients and the HPC host(s), we realized functionality which is not feasible to be run on the HPC host(s). In particular, current Web services-based middleware provides no possibility to implement real-time interactions of users; therefore, all real-time communication of the collaborative modeling is handled by the Clayworks server. The database, residing at the Clayworks server, allows to reuse objects and to recover a workspace in case of a server failure.

For simulations on the HPC host, the server prepares datasets for the remote computation, it starts and monitors
the progress of the computations performed by the remote HPC host and finally returns the result to the clients.

**HPC Tier:** The HPC tier of Clayworks runs a parallel implementation of the clay deformation algorithm [3] used for the simulation. The implementation is realized as a HOC (Higher-Order Component) in Java. Higher-order components, introduced in [10], abstract over the middleware used to connect to the HPC host performing parallel computations, i.e., in our application, the simulated clay deformation. As discussed in Sect. 4.2, HOCs allow the developer to concentrate on implementing a particular algorithm, while the middleware support, required to exchange data in portable formats, is pre-packaged with the HOC and thereby hidden from the developer.

The HOC used in Clayworks is built on top of the Globus WSRF, so that a Web service is used to outsource computations. To start a simulation, the server sends all objects in the workspace and additional simulation information, like the objects’ velocities and movement direction vectors, to this Web service. The deformation algorithm then iteratively moves and deforms the objects, resulting in a sequence of single simulation “screens”. Upon completion of the simulation, i.e., when all objects have spent their kinetic energy for movement and deformation, the server downloads the simulation results from the Web service and sends them to the clients. To reduce the size of the transferred data, the ZLIB-compression utilities from the java.util.zip package are employed. Due to the low number of differences between subsequent pictures in an animation, the size of the result files, which must be transferred over the network, can be compressed to less than a hundredth of their original size.

### 2.3 Collaboration in Clayworks

The Clayworks clients allow the users to form, resize, reposition, rotate, split up, merge or logically group objects. Objects can be built either by using built-in primitives to create base objects for further refinement or by drawing and spinning 2D shapes to directly sculpt complex objects. Clayworks allows its users to collaboratively work synchronously as well as asynchronously.

**Synchronous collaboration** connects several users to a single workspace at the same time. The users can immediately see the changes to objects made by other users, which allows to collaborate in a tightly-coupled manner. In practice, users often split up the work on a complex object among them, each user working on different parts of the object. Clayworks immediately shows the updated object if a user changes the shape of a specific part of the object. Other users can immediately react and adapt the shape of their own parts to the changes or discuss the change with the initiator of the update using the built-in chat functionality. Additionally, Clayworks allows to lock objects, such that no other user can interfere if a specific user wants to execute a complex series of modifications to a specific object.

**Asynchronous collaboration** does not require the users to be connected simultaneously; moreover, they can work on the same workspace or on the same objects one after the other. Clayworks allows this type of collaboration by providing two technical mechanisms: a) arbitrary leaving and joining a session, and b) persistent workspaces and objects in the Clayworks database:

- The arbitrary joining allows users to connect to an already existing workspace. The objects are version-controlled at the server, such that a client joining late can download the latest version data, even if other users proceed with the modeling during that download.

- The Clayworks database allows to collaborate asynchronously due to the possibility to save a complete workspace and load it later on. It is possible to shut down a server and restart the session from the last state some time later. Furthermore, we use the database for a recovery mechanism, based on the frequent storing of backups. Thus, our centralized Clayworks server does not form a single point of failure.

### 3 CSCW: Distributed Real-time Modeling

In this section, we present the technical realization of the collaborative real-time modeling in Clayworks. We explain two main aspects of the modeling part: the data structures used for representing clay objects and the distributed execution of user actions.

#### 3.1 The Clayworks Client

The Clayworks client makes use of OpenGL via the Java 3D API for the visualization of the workspace. All the different functionalities, like user communication via chat, managing the workspace by locking or saving objects and the modeling of objects, are seamlessly integrated into a single graphical application. Additionally, the parameters for simulation runs, like the velocity and direction of objects’ movement, can be set up. Completed simulations can be viewed in a special window allowing to freely rotate and move the observer position (see Fig. 3 for an example view).

#### 3.2 Asynchronous Command Processing

The Clayworks server process maintains one or several independent collaborative workspaces. All modeling ac-
tions of users are implemented as remote methods residing on the server, which serializes these actions and guarantees consistency of the shared workspace. The server methods remotely called by clients insert a specific command-object (which can, for example, represent a translation, rotation or reshaping of a particular clay object) into a queue and immediately return then. This mechanism makes the originally synchronous Java Remote Method Invocation become asynchronous: The client does not have to wait until the remote computation is finished, but can continue to process user inputs immediately after issuing a command.

At the server, several worker threads execute the commands from the queue in parallel as long as the commands affect different objects. This allows to speed up the command execution for independent commands on multi-processor or multi-core CPU servers, thus increasing the responsiveness of the client, while the correct order of commands affecting the same objects is still guaranteed.

3.3 Command Communication

When the server executes a modeling command from the queue, it transmits that command to all clients, which in turn execute the command locally. This way, instead of a huge polygonal mesh, only a small command has to be transferred to the clients. Conflicts between commands, e.g., a concurrent movement of a single object into multiple directions, are avoided using the locking mechanism, already mentioned in Section 2.3: a locked object can be manipulated exclusively by only one user at a time. With this mechanism, actions never need to be reset and we avoid the bouncing of objects and other unnatural movements due to command synchronization. Transitions between subsequent actions are always performed smoothly, since the placement of the command queue on the Clayworks server prevents effects of jitter or network latencies.

Due to the fact that each client executes all modeling operations of all users, clients should be run on well-performing desktop systems with a 3D-accelerated graphics card for the Java 3D display. The clients perform well on standard Intel P4 2.6 GHz desktop computers with GeForce 4 graphics cards, so that the hardware requirements of the client can be considered reasonable.

3.4 Two Optimized Representations for Clay Objects

Besides communicating user commands between clients, the server has to hold the central copy of all objects, which have two different representations: polygonal or voxel-based in a three-dimensional grid. Fig. 4 shows a sphere-shaped object in the two different formats: the polygonal version is shown left and the coarser version on the right illustrates the voxel-based version. In the lateral cut-away view of the voxel-based version, it can be seen that only the visible external part of the object is covered by voxels while its inside is unfilled, thus reducing the data size of this format.

The polygonal representation is used in the modeling part and allows to build the objects in an intuitive way by defining vertices and faces. The voxel-based representation is required by our parallel simulation algorithm.
Our solution of transforming data representations at an intermediate application server is feasible for many other distributed applications. In general, this allows clients to always operate on a data structure which is most suitable for visualization and editing, while the simulation can operate on a different data structure optimized for high performance or a specific algorithm.

4. Simulating Clay Deformations in Parallel

This section presents the concept and implementation of the HPC server tier of Clayworks. We briefly describe the basic algorithm used for the simulation and discuss how multiprocessor servers can be used to speed up the required computation. For the implementation, we developed a new Higher-Order Component [10] (HOC), called Deformation-HOC. HOCs are software components, pre-packaged with a reusable implementation of an algorithm plus middleware support. The algorithm provided by the Deformation-HOC computes a simulation of a material deformation and is an adaptation of the algorithm from [3].

By packaging the component code together with all necessary configuration files, these files, which are typically coded in multiple XML-based formats, become invisible to HOC programmers. The programmer does not need to instruct the middleware on how to handle the communication among heterogeneous resources, but he only writes Java code and is freed from dealing with any XML at all.

4.1 The Multilayer Deformation Process

Once the collaborative construction of the clay objects is finished, the users can specify a direction and a velocity for each object. When a user starts the computation of a simulation, the scene description containing the object coordinates plus the movement information is sent to any available HPC host which runs the simulation algorithm. The simulation algorithm used to compute such scenes is implemented as a multithreaded Java program and, therefore, SMP-servers that map Java threads to different processors are the most appropriate choice of architecture for computing simulations. While multiple simulations can be computed by multiple servers simultaneously, the single simulations are always computed independently, using one dedicated server. All our experiments were conducted using a SunFire 880 computer with 8 UltraSparc-III processors, each running at 1200 Mhz.

Fig. 5 shows the average runtime for computing a deformation of two clay cubes (see Fig. 3). As can be seen, there is quite a good speedup when the number of threads is increased. For 8 threads, e.g., we measured a speedup of 7.2, for a good case, i.e., for a scene wherein the moving clay objects were evenly distributed.

Inside the simulation algorithm, the voxel-based representation of the objects is held in a 3-dimensional array, the cubic voxel universe. Each element of this array is an instance of the Voxel-class, holding the two attributes density and velocity. For each voxel, the density attribute stores the density of the clay at the voxel. All voxels have the same constant volume, which depends on the grain of the cubic voxel universe, which must be specified before the simulation starts. Before the start and after the completion of the algorithm, the clay density of all voxels ranges between 0 and 1, but during the computation, values above 1 are possible. Voxels with a density value above 1 are called congested. The impact of congested cells on the computation is discussed below. The second attribute of the Voxel-class, velocity, is a vector describing the movements taking place in the universe. These movements are always linear shifts, triggered by the Clayworks users.

Basically, we run the sequential algorithm from [3] in parallel on multiple partitions of the cubic voxel universe. In contrast to [3], our implementation does not require the presence of a solid tool object, i.e., all objects in the virtual universe can be built of clay. We also extended the algorithm by introducing energy reduction, which terminates the deformation process in a natural way without requiring the user to stop it. As suggested in [3], our implementation performs a 3-layer traversal, but additional layers can be added using parameters.

In each single traversal, all voxels are visited once and new density values are computed for them. During the first layer traversal, the shifting of voxels is computed. As long as the position of a voxel after a shift does not collide with the position of another voxel, there is no clash and the direction of the velocity vector belonging to this voxel is preserved. The magnitude of the vector is altered by multiplying it with an energy reduction factor \( \alpha \) in each traversal. To find a good assignment for the energy reduction \( \alpha \), we experimented with multiple different factors, including sim-
ple linear ones. Empirically, we verified that the reduction of energy during a shift is simulated very realistically when the exponential function is used to compute the energy reduction factor $\alpha$, as follows:

$$\alpha := \exp\left(-1.0 \cdot \frac{\rho_1}{\rho_2}\right) \quad (1)$$

where $\rho_1$ and $\rho_2$ are the density values of the clashing voxels. When voxels clash into each other, a new velocity vector is computed using the formula

$$\delta := \frac{1 - \alpha}{2} \delta_1 + \frac{1 + \alpha}{2} \delta_2 \quad (2)$$

where $\delta_1$ and $\delta_2$ are the original velocity vectors of the clashing voxels. $\alpha$ adheres to definition (1) reflecting the intensity of the clash, which is reduced proportionally to the reduction of energy.

4.2 HOCs: Parallel HPC Components

The simulation algorithm described in Section 4.1 can be used for the parallel implementation of a broad class of spacial simulations. It is suitable to compute a realistic presentation of any scene, wherein an arbitrary material is deformed, e.g., the diffusion of gases or the freezing of fluids. It is a recurrent program structure and therefore a candidate for a reusable implementation as a Higher-Order Component (HOC)[10].

Our Deformation-HOC provides the simulation algorithm via a customizable Web service. The setup, which enables, e.g., asynchronous communication between the Clayworks server and the Deformation-HOC, is included in the HOC. The application-specific parts are sent to the HOC using Web service parameters. In the case of the Deformation-HOC, only the procedures during density shifts in the first layer and the procedures to correct overfull voxels in the remaining layers are application-specific. Therefore, the Deformation-HOC takes one customizing parameter which carries a serialized Java class. This class must be accessible using the following interface:

```java
public interface DeformationParameter {
    public double densTrans(int layerNr, double dens);
    public double powTrans(double nrg);
    public double powTrans(int layerNr, double dens);
}
```

The methods in this interface are as follows: `isSolid` returns, for a given objectID, whether the respective object is built of a solid material and therefore not affected by the deformation process.

`getMaxDensity` determines when a voxel is overfull. When a value above 1 is specified, less corrections must be performed, speeding up the higher-layer computations, but the accuracy of the simulation is reduced this way.

`powTrans` converts kinetic energy into deformation energy, as it happens during the deformation process.

`densTrans` is used to distribute density among neighboring voxels in a layer $> 1$, i.e., within a correction.

Users can customize the Deformation-HOC by sending their own implementations of the DeformationParameter. If no such parameter is specified, the HOC uses the DefaultDeformation class, wherein the above methods are implemented according to the mathematical definitions (1) and (2) in Section 4.1: i.e., as long as no different DeformationParameter is specified, the Deformation-HOC computes the deformation of clay objects. Writing a specific DeformationParameter for a different application requires some knowledge about the physical properties of the material which is simulated. When, e.g., a steel brick is simulated, the required DeformationParameter is derived from
the DefaultDeformation-parameter and the methods isSolid and powTrans are overridden, such that isSolid returns true and powTrans returns a constant, depending on the weight of the brick.

Our Deformation-HOC facilitates code reuse and customization. Its full implementation comprises approximately 600 Lines of Java code plus 300 Lines of XML code for the WSRF support. When a new DeformationParameter is derived from the DefaultDeformation-parameter for different simulations, a programmer using the Deformation-HOC has to write no more than 20–30 lines of Java code.

5. Related Work and Conclusion

The main contribution of the Clayworks worksuite is that it tightly integrates collaborative modeling with a HPC infrastructure, allowing end users to collaborate and easily access high-performance servers in a transparent way. Clay deformation is an example application for a large class of problems in industry and science like crash tests, physical simulations or virtual biological experiments.

When compared to other distributed problem solving environments like CUMULVS [6] or Netsolve/GridSolve [14], Clayworks’ novel feature is the support for tightly-coupled, synchronous collaboration with soft real-time deadlines. The three-tier architecture of Clayworks satisfies the different requirements of collaborative modeling and HPC. The real-time requirements are combined with the HPC infrastructure in a transparent way for the end users. Furthermore, the transformation of the clay objects from polygonal to voxel-based representation and vice versa allows to use the representation best suitable for visualization and computation, respectively.

The contribution of Clayworks in the area of parallel and distributed programming is the development and use of the Deformation-HOC. For the parallel implementation of the deformation simulation, the Deformation-HOC proved to be a very suitable way of programming a Globus WSRF-based service without dealing with the details of middleware technologies. Clayworks confirms the advantages of a Grid middleware and a component-based approach to Grid programming, exemplified by systems like GAT [1], ASSIST [13] and Fractal [2]. Changing the underlying technology of our parallel implementation in a way that multiple servers (which can also operate/communicate using non-Java methods, e.g., MPI) process parts of our cubic voxel universe simultaneously, will not affect the CSCW part of our system, due to the loose coupling between the clayworks server and the HPC host(s) via a Web service.

The concept of seamless integration of remote HPC servers into application software can be expanded into novel areas beyond traditional industry and science: Video editing software could outsource transcoding processes, or online computer game players could start new game sessions on a remote server from the game client. In order to make e-Science suitable for the mass market and non-experts in the area of computing, the access to HPC resources has to be made as transparent as possible. Clayworks is a step in this direction, providing an easy to operate application suite.

6. Acknowledgments

We thank the following students for their work on Clayworks: P. Bennour, S. Dabek, M. Hiersche, R. Kappus, T. Lohe, P.-L. Lott, P. Lüdeking, M. Rapp, P. Rhiem, C. Sassenberg, and K. Walenta.

Sincere thanks also go to the anonymous referees of the draft version of this paper for their constructive comments.

References