A Component- and Multi Agent-based Visualization System Architecture

Hans Hagen[1], Henning Barthel[1], Achim Ebert[1], Andreas Divivier[1], and Michael Bender[2]
[1] Intelligent Visualization and Simulation Systems, DFKI GmbH, Germany
[2] University of Kaiserslautern, Kaiserslautern, Germany

Keywords

Visualization system, multi agent technology, component technology

Abstract

Today, most visualization systems available on the market lack of flexibility regarding their visualization process with respect to either their interaction possibilities or their rendering quality. In order to satisfy individual user demands and to adapt to changing system loads and different hardware configurations, multi agent technology is combined with the component-based implementation of a visualization system.

1 Introduction

Nowadays, most visualization systems available on the market are designed for special purpose and therefore lack of flexibility regarding their visualization process. This means they provide either a high rendering quality with limited interaction possibilities, or real-time visualization going along with a reduced rendering quality. In order to satisfy individual user demands and to adapt to changing system loads and different hardware configurations, an appropriate visualization system architecture should comprise an intelligent control unit supervising and tuning all system components during runtime.

Up to now the user manually has to adjust the balance between frame rate and rendering quality by modifying the related control parameters or by discovering and (ex)changing the causative system components. In the course of our work we provide a solution for these problems by combining multi agent technology and a component-based implementation of a visualization system.

The system architecture follows proved principles of modern software development resulting in small, platform independent software components. Therefore it is applicable even for less powerful machines, for example mobile computers.

2 General system architecture

Since a huge monolithic visualization system does not sufficiently support the adjustment or replacement of the implementation of a single functionality, the proposed design consists of several modules grouped into three layers according to their functionality (see figure 1). Each module is controlled by a small number of parameters enabling the management of its functional behavior.

The kernel layer covers basic visualization aspects including the management of scene definition, geometry objects, lighting etc. and is responsible for the optimization of the visualization process. Furthermore generic interfaces provide the link to the extension layer, which adds auxiliary functionality to the system. Examples for such extension modules are import/export, geometry generation, visualization methods and general I/O.

In order to preserve platform independency all render routines are encapsulated in hardware specific render managers which form the main parts of the Hardware Abstraction Layer (HAL).

Therefore, the design of our visualization system architecture follows proved principles of modern software development like a small and flexible kernel, modularity, expandability, as well as application and platform independence. Therefore it is applicable even for less powerful machines, for example mobile computers.

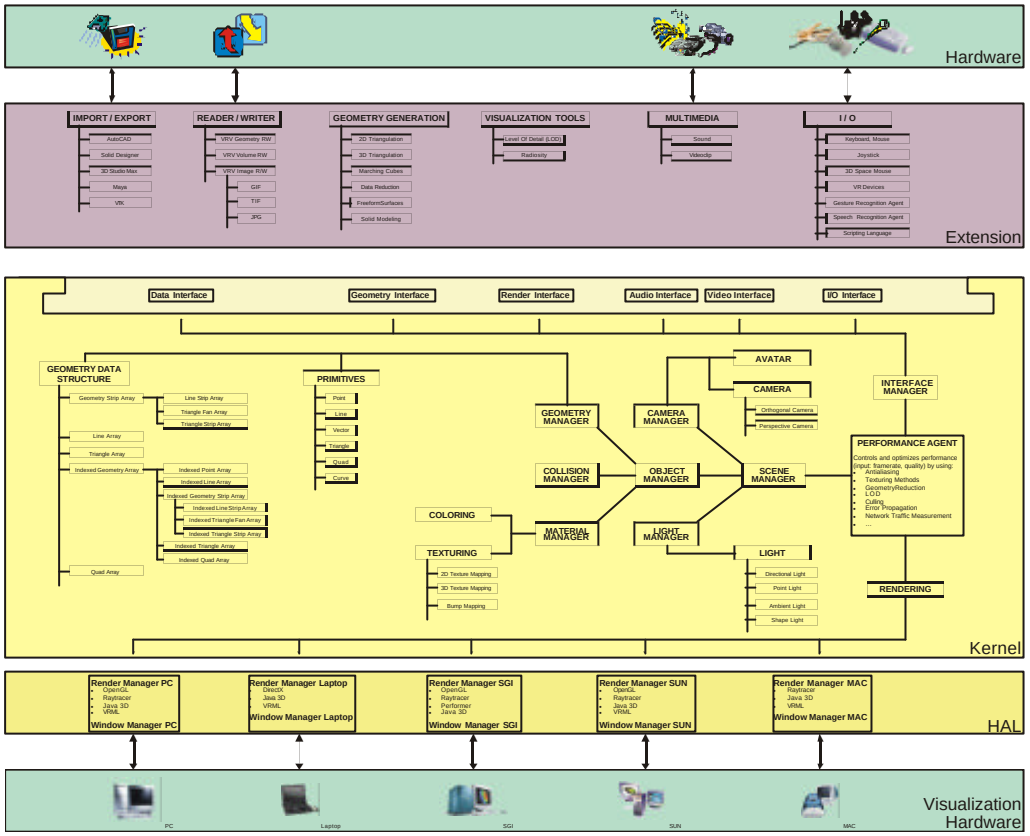

Figure 1: System architecture.

3 Agent-based visualization control

The monitoring and tuning of a visualization application is a complex problem. Due to continually changing conditions and the variety of parameters even experts with their competent knowledge are not able to solve such a problem in general. In order to improve this unsatisfying situation, we are extending the design of our visualization system by incorporating multi agent technology.

3.1 Agents and multi agent systems

In our point of view, an agent is a piece of software that detects and reacts on changes in its environment. An agent is able to operate autonomously, whereby its goals are explicitly given. A multi agent system is a software system build by a number of agents which communicate in order to solve more complex problems.

Apart from those definitions potential applications must meet three criteria for applying multi agent technologies:

- natural distribution,
- dynamic world, and
- complex interaction.

In our application scenarios we are dealing with visualization pipelines, whereby the complete process is devided into distinct parts. Also – especially in the case of a 3-dimensional environment – we have an extreme ly dynamic world with multiple user interaction possibilities. Therefore it is obvious that a visualization task fulfils all three criteria.

The main advantages of applying a multi agent system over using one single agents include that small interacting agents are simpler to handle and incorrect parts can be more easily identified and exchanged. Also this model is a basis for distributed / parallel computing.

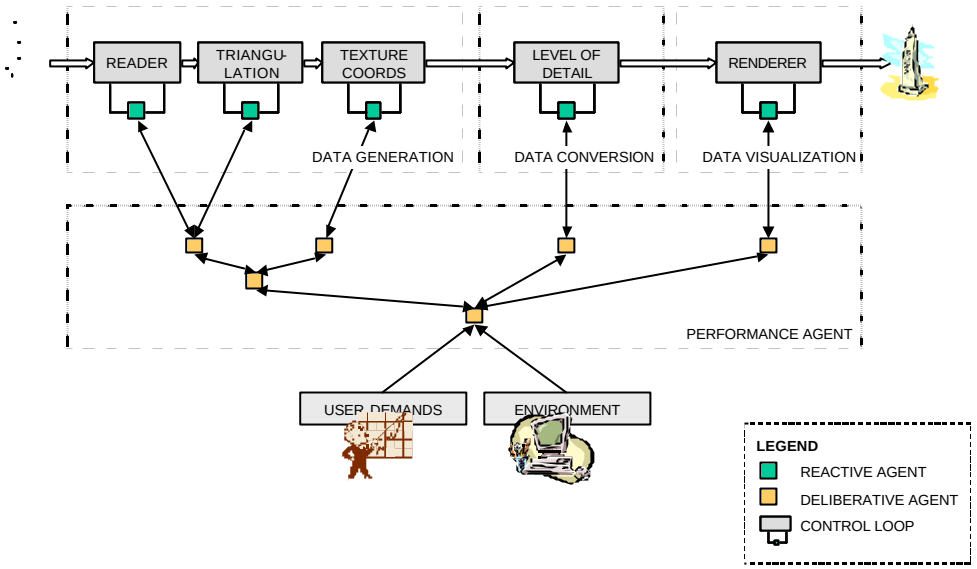

Figure 2: Visualization using multi agent technologies.

3.2 Visualization control

As described in section 3.1, conditions that change dynamically are the ideal premise for the use of multi agent systems. In order to brake down the complex adjustment process into tasks which are easier to solve, our approach makes use of two types of agents (deliberative and reactive agents). A deliberative agent has the ability to handle complex problems, but in general it is not able to process such jobs in real time on his own because of the applied sophisticated calculation methods. Reactive agents on the other hand only perform simpler tasks but they can be easily integrated into dynamic environments. Concerning the architecture of a visualization system, a module or a group of modules always performs one fixed task, this means that terms of regulation (e.g. frame rate and rendering quality) can be described by straightforward static rules. Thus, a reactive agent is always assigned to either a module or a group of modules and it controls the modules' parameters on demand. Since a reactive agent only needs the knowledge in the module specific context it acts similar to a control loop. The supervision of the reactive agents, as well as the the involved visualization analysis of information generated by them, is done by the Performance Agent, which consists of a hierarchy of deliberative agents. Based on the overall knowledge about components and with respect to the user specific demands it automatically responds to changes in the environment (e.g. the system load) by modifying the desired values of the subordinated reactive agents.

3.3 Example

Figure 2 shows an example of a visualization pipeline controlled by agents. In the data generation process an unstructured point set is imported and triangulated. After generating and assigning texture coordinates to the resulting object the data conversion process creates different, predefined levels of detail for the corresponding object representation. Within the data visualization process the selected representation is rendered by mapping it onto the underlying hardware abstraction layer.

To achieve the user defined rendering quality and performance reactive agents are assigned to each component of the pipeline and are supervised by the Performance Agent. Exemplarily we will briefly describe the interplay of the Data Conversion and Data Visualization modules and the Performance Manager in the present case: The reactive agent of the Data Conversion module controls the level of detail component by altering its parameters affecting the level of detail algorithm, the reduction level and the accuracy demand. The output of the Data

Conversion module is the effective accuracy level and the number of generated triangles. The reactive agent of the Data Visualization module controls the render component by varying the representation model (wire frame, solid), the shading model, the rendering algorithm, the resolution and the lighting and texturing parameters. The output of the Data Visualization module is the current frame rate or the render time respectively. Both modules receive their permitted parameter ranges from the Performance Agent which in its turn is biased by user demands and the system environment.

4 Component-based implementation

In order to simplify the development of visualization applications, a more pragmatic methodology must defined. For this reason we distinguish two types of visualization software engineers:

- Component developers: engineers that are developing the visualization modules.
- Component users: people that are building applications by assembling existing modules.

By providing a visualization development environment, the component user only needs the information about the existing modules and their functionality. Therefore, an extensive programming knowledge is no longer necessary in this case.

Furthermore, component technology offers an ideal platform to achieve the uncoupling of visualization modules and visualization systems.

4.1 Components and repository

In our context a component is an encapsulated piece of software with a standardized, contractually specified interface giving access to a corresponding functionality (service). The definition of interfaces as well as its administration are regulated by a component system (repository). Therefore a calling process (client) only gets the information about the logical structure of the service according to the interface definition and experiences the behavior through invocation. Consequently the strict separation between implementation and provided functionality supports the development of independent and reusable software components, which leads to the following advantages:

- Dynamic linking: Components can be replaced, added to or deleted from an application at runtime.
- Faster application development: The effort necessary to build / assemble new applications can be reduced in case suitable components are available.
- Small costs: If an application can be assembled out of already existing components, smaller costs arise due to the shorter development times.
- Reliability: Components are permanently extended and improved leading to a higher reliability of the respective applications.
- Flexible system architectures: Easy exchange and customization of components leads to an improved extendibility and maintainability of applications.

4.2 Implementation details

In our implementation we decided to use Sun's Java Beans component technology to realize the modules described in section 2. A Java Bean is a reusable software component that can be manipulated visually in a builder tool. The Java Beans specification is based on classes and only defines some rules (design patterns), which let a class become a component. The interface description of a Java Bean component is defined by the BeanInfo class, which includes information about the methods, properties and events of each Bean. The component developer can model the interfaces in three different ways: automatic description, manual description or IDL description. By using IDL, a Java Bean can function as a CORBA server, i.e. it can provide its functionality to other CORBA components, as well as a client invoking CORBA services.

4.3 Visual Prototyping

With the use of a visual prototyping system new applications can be plugged together at runtime. The control of visualization parameters can be automatically handled by attaching appropriate reactive and deliberative agents to the components. Here the integration of agents directly fits into the concepts of the desired component technology, that is reactive and deliberative agents are software components themselves.

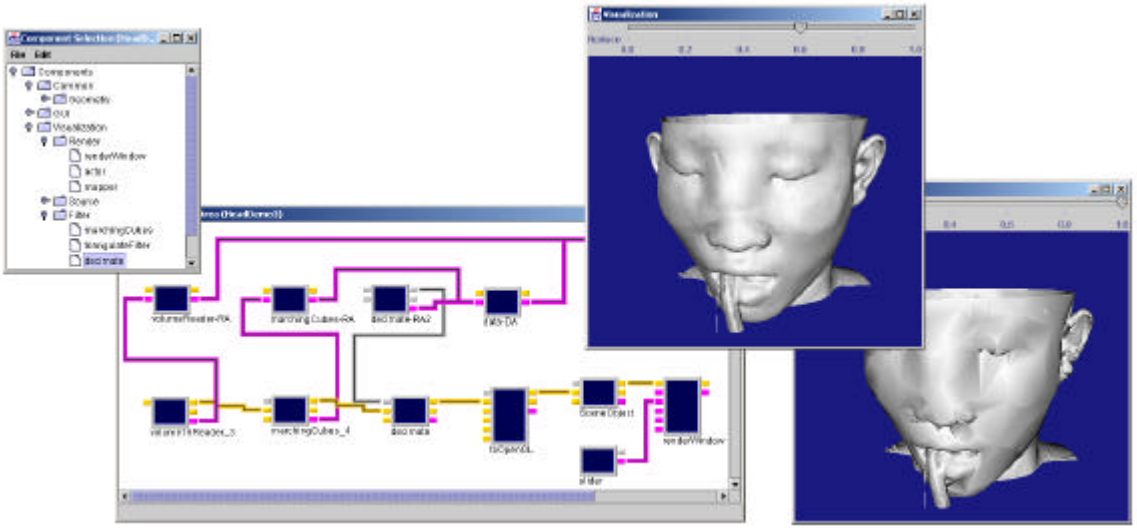

Figure 3: Snapshot of our visual prototyping system.

4.4 Example

Figure 3 shows a snapshot of our visual prototyping system while assembling a special volume data visualization application. The intelligent visualization system response which depends on the users interaction demands and the system environment is illustrated by the two images showing different levels of detail.

5 Ongoing work

In this paper, we have presented the design of a visualization system that is based on component and multi-agent technologies. With this approach we are able to uncouple visualization modules and visualization systems and automatically monitor and tune the visualization application.

Up to now, some applications in the field of scientific visualization are taken as experimental environments to evaluate and benchmark the implemented features in realistic settings. It is yet unknown if a single deliberative agent will be able to control an entire visualization system or if a hierarchy of deliberative agents is needed for this task.

Another task to do will be the more detailed evaluation of our architecture in the field of mobile computing.

References

[1] Achim Ebert, Andreas Divivier, Michael Bender, and Henning Barthel: *A Visualization System using Multi Agent Technologies*, Visualization 2000 Conference, USA, 2000

[2] Henning Barthel, Michael Bender, Andreas Divivier, Achim Ebert, and Hans Hagen: *A Component based Visualization Architecture*, CoData 2000, Italy, 2000

[3] Bender, M., Klein, R., Disch, A., Ebert, *A.: A Functional Framework for Web-Based Information Visualization Systems*, IEEE Transactions on Visualization and Computer Graphics, Volume 6, No. 1, 2000

[4] Bender, M.: *A Functional Framework for Efficient Web-based Scientific Visualization Systems*, Dissertation, University of Kaiserslautern, 2000

[5] Barthel, H.: *Komponentenbasiertes visuelles Prototyping von Visualisierungsanwendungen*, Dissertation, University of Kaiserslautern, 2000

[6] Sun Microsystems: *Java Beans*, Sun Microsystems Inc., Specification 1997

[7] Sun Microsystems Inc.: *JavaBeans for Java Studio: Architecture and API*, Sun Microsystems, Inc., 1997

[8] Ranze, Müller: *Über den Einsatz von Agenten in Umweltanwendungen*, 11. Symposium "Informatik im Umweltschutz", 1997

[9] Genesereth, Ketchpel: *Software Agents*, Communications of the ACM, 37, 1994

[10] Nwana: *Software Agents: An Overview*, Knowledge Engineering Review, 11(3), 1996

[11] Wooldridge, Jennings: *Intelligent Agents: Theory and Practice*, Knowledge Engineering Review, 10(2), 1995