Service-Oriented Scene Graph Manipulation

Andreas Schiefer*
René Berndt†
CGV, TU Graz

Torsten Ullrich‡
Volker Settgast§
Fraunhofer Austria

Dieter W. Fellner
Fraunhofer IGD & TU Darmstadt

Abstract

In this paper we present a software architecture for the integration of a RESTful web service interface in OpenSG applications. The proposed architecture can be integrated into any OpenSG application with minimal changes to the sources. Extending a scene graph application with a web service interface offers many new possibilities. Without much effort it is possible to review and control the scene and its components using a web browser. New ways of (browser based) user interactions can be added on all kinds of web enabled devices. As an example we present the integration of "SweetHome3D" into an existing virtual reality setup.

CR Categories: H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Artificial, augmented, and virtual realities; H.5.3 [Information Interfaces and Presentation]: Group and Organization Interfaces—Web-based interaction; I.3.2 [Computer Graphics]: Graphics Systems—Distributed/network graphics

Keywords: web services, scene graph systems

1 Introduction

The intention to combine a web service with a scene graph system is to address the following problems:

Modeling Visualization Cycle Most virtual environments offer only limited modeling capabilities. As "standard" modeling software (Maya, 3D Max, etc.) is seldom adapted to virtual reality (VR), many VR systems are just viewers. In this case, the modeling-visualization-cycle is interrupted; i.e. a human modeler has to switch frequently between modeling environment and VR visualization. Especially during presentations where customers inspect 3D models in VR and want to apply model changes (customization), an interrupted modeling visualization cycle is not feasible.

VR Correlated UX Due to differences in VR systems (High-resolution projection walls, CAVE environments, etc.), the user interaction and user experience (UX) implement different paradigms. The interaction capabilities may vary from desktop-based mouse and keyboard interaction to multi-touch or 3D gestures. These shifting hardware situations result in a substantial adaption and implementation effort.

*e-mail: a.schiefer@cgv.tugraz.at
†e-mail:r.berndt@cgv.tugraz.at
‡e-mail:torsten.ullrich@fraunhofer.at
§e-mail:volker.settgast@fraunhofer.at

Web3D 2010, Los Angeles, California, July 24 – 25, 2010.
© 2010 ACM 978-1-4503-0209-8/10/0007 $10.00

The proposed solution to the previously described problems is a flexible software architecture. It strictly realizes the model-view-controller pattern. As a consequence the software development process can be parallelized. Furthermore it is easily adaptable to new user interfaces and requirements.

2 Web Services

Web services provide an implementation for a service-oriented architecture (SOA) [Josuttis 2007]. The main characteristic of this software design principle is a loose coupling of services. These services communicate by exchanging messages. This provides a great flexibility in term that two applications need only to agree on the message format and are independent from the actual application code.

The following definition of a web service was introduced by the World Wide Web Consortium (W3C) [Booth et al. 2004]:

> A web service is a software system designed to support interoperable machine-to-machine interaction over a network. . . .

Web services offer a number of advantages over other machine-to-machine middleware systems like Open Network Computing Remote Procedure Call (ONC RPC) [Srinivasan 1995], Distributed Component Object Model (DCOM) [Brown and Kindel 1998], Common Object Request Broker Architecture (CORBA) [Object Management Group], Remote Method Invocation (RMI) [Sun Microsystems Inc. 2004], etc:

Platform support Web services use the world wide web for communication. Every platform got already built-in support for the web. Web services can be consumed by web browsers. Since most platforms already include a web server even hosting web services can be done without additional efforts. Other module communication systems are only supported by a small number of platforms (e.g. DCOM is available only for the Windows platforms) or do not ship with the platform and need to be installed separately (e.g. CORBA).

Language support Almost every programming language provides support for internet protocols and XML processing (C++, Java, C#, JavaScript, Python, Ruby, etc.). Although also other middleware systems (e.g. CORBA) claim to be language independent one will find only support for C, C++, and Java.

Internet The major drawback of systems like DCOM and CORBA is that communication across the internet is nearly impossible. These middleware system use binary formats which are blocked by most firewalls. Even in the same department it can be a painful task to configure the network to successfully communicate using DCOM/CORBA.

Currently, two principles to implement web services exist:

SOAP The Simple Object Access Protocol (SOAP) [Gudgin et al. 2003] is a protocol for exchanging structured messages with a web service. The structure of this messages is usually described through the Web Services Description Language

(WSDL) [Christensen et al. 2001]. Both SOAP and WSDL are XML-based language.

REST Representational State Transfer (REST) was first introduced by Dr. Roy Fielding in his Ph.D. thesis [Fielding 2000]. The basic principles behind REST are

1. identification of resources using URI
2. manipulation of resources using standard HTTP operations (GET, POST, PUT, DELETE)
3. self-descriptive messages (e.g. Plain Old XML (POX), JavaScript Object Notation (JSON) [Crockford 2006], etc).

3 Scene Graph Systems

Typically a scene graph is a hierarchical collection of nodes expressing logical and spatial relations of 3D objects. Each node may have arbitrary children but only one parent node. The first node of the hierarchy is called root node from which all of the nodes in the scene can be accessed by traversing through the graph. Nodes are typically parts of the geometry of the scene. Other functionality like transformations and groups can also be expressed by nodes.

Effects on a node also affect the node's children. Spatial relations between objects can be expressed this way. If for example a chair node is added as child of a room node, the transformations applied to the room node will also affect the chair.

With a scene graph it is possible to apply high level optimization like visibility culling. This is hardly possible using a low level API because a typical state machine renderer is not aware of the whole scene. A visibility test for a node is typically performed by intersecting the node with the planes of the view frustum. Often a bounding volume hierarchy is used for fast visibility tests. Bounding volumes of low complexity for example a sphere or a box are fast to compare. The bounding volume of a node is defined by the bounding volumes of the nodes children. If a visibility test for a node fails, also its children will not be visible and the whole sub graph does not need to be send to the graphics hardware.

In many scene graphs it is possible to reuse geometry in multiple locations. This is called instancing. In the same way other resources of the scene may be reused like materials, shaders and transformations. This helps to reduce memory consumption.

3.1 OpenSG

OpenSG[1] is an open source C++ scene graph system for high quality real time 3D graphics [Reiners et al. 2002]. After the Fahrenheit scene graph project of Microsoft and SGI was not continued in 1999 among others OpenSG was born. It was initially designed and implemented by Dirk Reiners, Gerrit Voss and Johannes Behr at IGD Fraunhofer in Darmstadt. OpenSG is based on the cross platform graphics API OpenGL. It is available under the GNU Lesser General Public License (LGPL) and runs on Windows, Linux and Mac OS.

The plan is to develop a scene graph system that serves as a general basis for a wide variety of applications. OpenSG 2.0 is currently the latest version with many improvements in usability and supported features.

Performance From the beginning it was designed to make optimal use of the graphics hardware. To obtain optimal performance OpenSG provides optimization algorithms. For example it provides functions to transform a model consisting of triangle to connected triangle stripes, a material sorting optimizer which minimizes the number of state changes for rendering the scene. Also many high level optimizations are included in the OpenSG renderer like visibility culling. The processing of scene parts which are behind the viewer or occluded by other parts can be skipped by OpenSG to speed up the rendering of large scenes.

Multi-Threading and Clustering OpenSG allows to have multiple threads accessing and manipulating the scene graph in an efficient way [Voß et al. 2002]. Data structures are organized in multi-thread safe buffers called aspects. Each thread can be assigned to its own aspect meaning it can have its own copy of the buffer. To reduce memory overhead the copy is created only on demand. Remote aspects are copies of the buffers which are send over the network. They are used to synchronize the scene graph within a cluster of multiple machines. In this way the rendering of complex geometry can be handled interactively. Also virtual environments consisting of many screens and machines like tiled displays and CAVEs are implemented with remote aspects.

Dynamically Extensible Not only by making the source code available but also by using highly dynamic and flexible structures OpenSG can easily be extended or adapted to specific needs. Custom render nodes can be added to extend the render capabilities. Reflective interfaces allow to integrate arbitrary types of custom data to the whole system. New application-specific data which is not known by the system at compile time can however be used in the internal loaders and writers and synchronized to a cluster.

4 Web Services and Scene Graphs

Integrating a web service into an OpenSG application allows to access and modify the contents of the scene graph during runtime without recompilation of the application. These features offer many new possibilities in interacting with the OpenSG application and the scene graph, some of them are discussed in detail later in this paper. A system overview is sketched in Figure 1.

The main features of our web service are querying the contents of the scene graph, adding and deleting of nodes and changing properties of nodes. Referencing of node data – to use it in multiple nodes – is also supported by the web service API. It is also possible to download the whole scene or sub graphs of it as a single file from the web service in various file formats.

4.1 RESTful web service API

The API to access the web service is designed in terms of the Representational State Transfer (REST) architectural style as described in [Fielding 2000]. Reasons for choosing the REST style for the web service API include the simplicity of REST and that the hierarchical nature of a scene graph fits nicely to the resource oriented URL style of RESTful web services. Also, the four HTTP methods GET, POST, PUT, DELETE used to access the RESTful web service map very well to the most common operations on the scene graph: reading, creating, updating and deleting of nodes and values.

For implementing the web service, the GNU library libmicrohttpd[2] is used. As the data interchange and resource representation format the JavaScript Object Notation (JSON)[3] is used by our web service.

[1] http://www.opensg.org

[2] http://www.gnu.org/software/libmicrohttpd/

[3] http://www.json.org

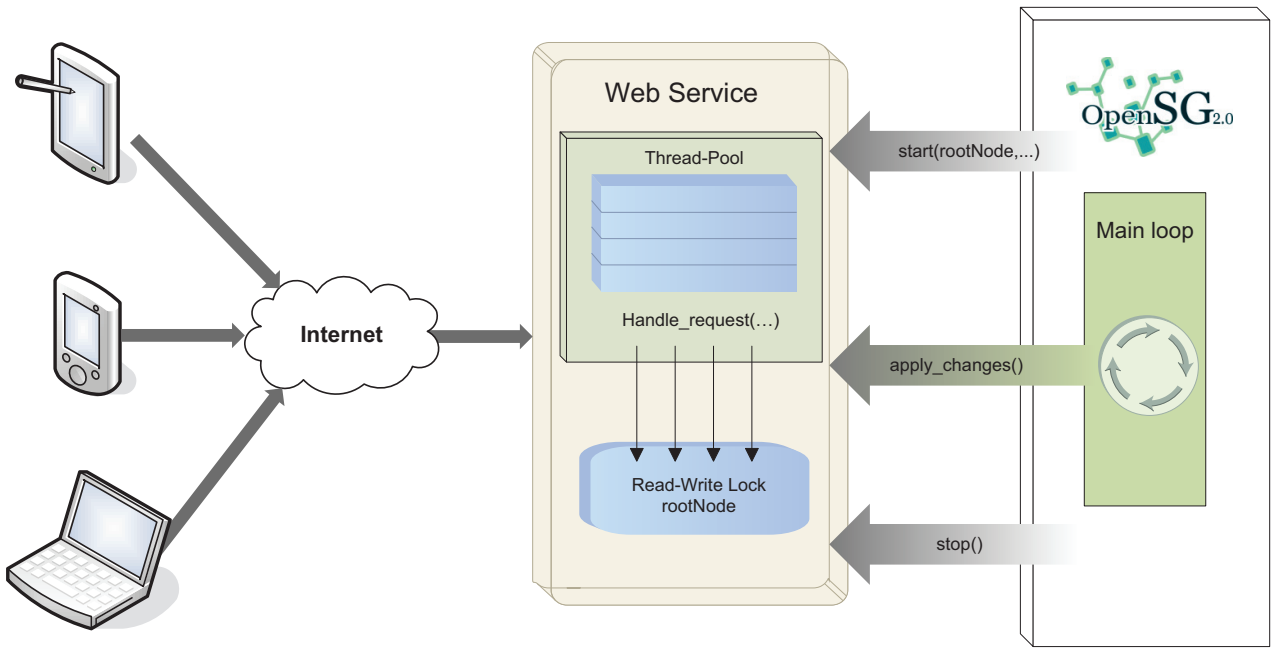

Figure 1: *The integration of a service-oriented architecture via a web service in the scene graph system OpenSG offers many possibilities. Three method calls are sufficient to export a scene graph of an arbitrary OpenSG application. This minimally invasive change of the application transforms closed visualization into an open service-oriented platform.*

It is a lightweight, human readable and easy to parse data interchange format based on a subset of the JavaScript programming language.

There are three types of representations that are returned by the web service: `FieldContainer`, `List` and `Data`.

Data `Data` resources contain the value of primitive OpenSG data-types like integers, floats, vectors or matrices as a string. Additionally the data-type itself is also specified as the name of its OpenSG class. In the following example the field `travMask` from the `FieldContainer` example below is represented:

```
{
  "content": "Data",
  "type": "UInt32",
  "value": "4294967295"
}
```

List `Lists` are ordered collections of resources that are accessible by index. The `List` representation contains the type of its items and a list of locations to the item resources. The following example shows the JSON representation of the `children` field, again from the `FieldContainer` example below:

```
{
  "content": "List",
  "type": "NodePtr",
  "items": [
    "/scene/root/children/0",
    "/scene/root/children/1"
  ]
}
```

FieldContainer `FieldContainers` are resources that itself contain locations of other resources which are accessible by name. Every `FieldContainer` has an unique id and a type, which is the name of its OpenSG class. All sub-resources of the `FieldContainer` are accessible through the `fields` attribute of the JSON object, which contains a mapping from the field name to the location of the corresponding resource. Here is an example of a `FieldContainer` that represents an OpenSG `Node` object located at the root of the scene graph – it contains locations to both the `travMask` and the `children` resources of the above examples:

```
{
  "content": "FieldContainer",
  "id": 292,
  "type": "Node",
  "fields": {
    "attachments": "/scene/root/attachments",
    "travMask": "/scene/root/travMask",
    "core": "/scene/root/core",
    "children": "/scene/root/children"
  }
}
```

All OpenSG data-structures accessible in the scene graph can be represented with this three basic types. For accessing the resources, the four HTTP methods GET, POST, PUT and DELETE are used. To support AJAX web applications on different domains to access the web service, the method OPTIONS is also partially supported. Web browsers use this method to authenticate cross site requests and web services must respond to OPTIONS requests as described by the W3C Cross-Origin Resource Sharing working draft[4] to allow the cross site requests.

[4] http://www.w3.org/TR/cors/

The semantics of the different HTTP methods when accessing resources of the web service are now described in detail.

4.1.1 GET

GET requests are used to get the representation of a resource. All valid URLs of the web service can be called with the GET method and return one of the three representation types presented above. A GET request does not change any data in the scene graph.

If a GET request is made with just the URL to the resource, one of the JSON representations is returned depending on the type of the specified resource. Additionally, a `filetype` URL query with a file extension can be specified to download the resource - including sub-resources - in the specified file format. This applies only to `FieldContainer` resources with the type `Node`, for all other resources the query is ignored.

If the called URL is not found in the scene graph, an error message with the HTTP status code `404 - Location not found` is returned by the web service, otherwise the response has the status code `200 - OK`.

Examples for GET requests:

GET /scene/root/core Get the JSON representation of the core of the root node

GET /scene/root?filetype=osb Download the root node (and all its children) as OpenSG-Binary file

4.1.2 POST

The POST method is used to either appending new nodes to resources of type `List` or to update the value of a `Data` resource. According to the HTTP/1.1 specification [Fielding et al. 1999], the POST method is the only non-idempotent method. In other words, only for the POST method it is allowed that multiple identical requests lead to different results (as when appending to a list).

Updating of `Data` resources is also handled with the POST method because POST can handle big amounts of upload data which may occur when updating textures or vertex arrays.

For appending a new node to a `List` resource, the type of the new node must be specified with the `type` URL query. If the specified type is a OpenSG node, a `Group` core is automatically added to prevent nodes without a core. On the other hand if the specified type is a OpenSG node-core, then an empty node is created and a core of the specified type is added. The response to such a request contains a JSON string with the location of the new node and additionally a HTTP header field `Location` with the full URL to the new node.

When updating a `Data` resource, the new value of the resource must be specified in the entity body of the HTTP request as a JSON string. After a successful update, the JSON representation of the updated resource is returned - the same as a GET to the resource, but with the updated value.

In case a new node is successfully added to a `List` resource, the response has the status code `201 - Created`. If a `Data` resource is successfully changed, the status code of the response is `200 - OK`. There are multiple possible errors that can occur using this method. A `400 - Bad request` status code is returned if the `type` URL query for appending a new node is missing or the given type is unknown, or if the entity body of the request is not a valid JSON string when updating a `Data` resource. For unknown resources – when the URL is not found in the scene graph – the status code `404 - Location not found` is returned. Finally, if the called URL does not point to a `List` or `Data` resource, the response has the status code `405 - Method not allowed`.

Examples for POST requests:

POST /scene/root/children?type=Geometry Appends a new node with a `Geometry` core to the children of the root node.

POST /scene/root/children/1/core/matrix With `"1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1"` as the entity body, sets the matrix of the second child's core to the identity matrix.

4.1.3 PUT

Similar to POST, a PUT request can be used to create new resources. But instead of appending a resource to a list PUT replaces an already existing resource or inserts the new resource at a specific index in the list.

To replace an existing `FieldContainer` resource with a new one, the PUT request either needs a `type` URL query with the type of the new `FieldContainer` to create - similar to POST - or a `location` URL query with the location of another `FieldContainer`. In the latter case, the specified location gets referenced from the location of the PUT request. This allows for example to instance geometry or to share a single material between multiple geometries.

If the request is successfully processed, the response has the status code `200 - OK`. As with the POST method, there are multiple error cases for which different error codes are returned. The status code `400 - Bad request` is used if the required `type` respectively `location` URL query is either not existing or the specified value is not valid. Also the `400 - Bad request` status code is returned if the index specified in the URL is not valid when inserting a new resource in a `List` resource. Like the other methods, a PUT request returns the `404 - Location not found` status code if the called URL is not found in the scene graph. If the called URL does not point to a `FieldContainer` resource or an item of a `List` resource, the status code `405 - Method not allowed` is returned by the web service. Some OpenSG containers do not support inserting items at a specific index, in that case a response with the status code `500 - Internal Server Error` is returned.

Examples for PUT requests:

PUT /scene/root/core?type=Geometry Replaces the core of the root node with a new `Geometry` core.

PUT /scene/root/children/0/core?location=/scene/root/core Sets the core of the first child to the root's core. Both nodes share the same core after this request.

PUT /scene/root/core/properties/8?type=GeoVec2fProperty Creates a new `GeoVec2fProperty` and assigns it to the ninth item in the root's core properties.

4.1.4 DELETE

The last method used by our web service, DELETE, is used to delete items from a `List` resource. If the DELETE request is made to the URL of the `List` resource itself, all items of that `List` are deleted. But if the request is made to one of the items in a `List` resource, only that item is removed from the `List`.

If the item is successfully deleted the response has the status code `200 - OK`. In case the called URL is not found in the scene graph a response with the `404 - Location not found` status code is returned. Because DELETE is only allowed for `List` resources

or for items of a `List` resource, the status code `405 - Method not allowed` is returned if the URL points to any other type of resource.

Examples for DELETE requests:

DELETE /scene/root/children/0 Removes the first child of the root node.

DELETE /scene/root/children Deletes all children from the root node of the scene graph.

4.2 Integration overview

Our web service is designed to be easily integrable into every OpenSG application with only a few changes to the source code. The public interface of the `OSGWebservice` class consists only of three methods, which is everything needed to get the web service running in an existing OpenSG application. The three methods are now presented in detail.

```
bool start(FieldContainerMTRecPtr root,
           UInt32 aspect,
           UInt32 port);
```

This method is used to start the web service. The parameter `root` specifies which resources are exported through the web service. The `FieldContainer` specified as the `root` parameter is accessible through the web service at the location `/scene`. Typically this is the root node of the scene graph, but it may also be an OpenSG `Viewport` or any other `FieldContainer`. In case it is an OpenSG `Viewport`, additional properties of the OpenSG application are accessible through the web service besides the scene graph, like the camera of the specified `Viewport`.

OpenSG provides so called aspects for multi-threaded operations on the scene graph. Threads can operate independently on different aspects of the scene graph without interfering each other. At some point, the aspects can be synchronized and every thread then sees the changes of the other threads. The `aspect` parameter defines the aspect on which the web service operates. This parameter should be set to some unused aspect which is then exclusively used for the changes through the web service.

Finally the `port` parameter specifies the port on which the web service listens for incoming HTTP connections. The default value for this parameter is `8080`.

```
void apply_changes();
```

This method should be called at regular intervals to sync the changes made through the web service into the application's main thread. Changes are also synced in the other direction, from the main thread into the web service, by this method. So it is important to call this method regularly to keep the application and the web service in sync. Typically this method will be called during the render function of the application every frame.

```
void stop();
```

Finally, this method can be used to stop the web service.

4.3 Threading issues

Depending on an user defined setting, the web service may use multiple threads to handle incoming HTTP requests. All of the threads work on a single OpenSG aspect (specified when starting the web service), so they need to synchronize their access to this aspect. To synchronize the threads, access to the aspect of the web service is protected by a Read-Write lock. Multiple threads can simultaneous read, but if one thread writes to the aspect no other thread can access the aspect. Every subsequent access to the aspect waits until the current write operation finishes. But even with this synchronization in place it is possible that threads overwrite each others changes to the aspect before the changes get synchronized to the main application. Therefore, to ensure reliability, currently only one writing change is allowed for every call to `apply_changes()`. Requests that only read data are not affected by this limitation.

4.4 Extensibility

Currently the web service provides access to an OpenSG application's scene graph. But users of the web service, the application programmers, may also want to provide additional functionality for their application through the web service. To satisfy their needs, an additional API is designed, which allows application programmers to map function calls of their application to user defined URLs accessible through the web service. The application programmer just needs to register a callback function at the web service with a specific user defined URL. The web service stores the function pointer to the provided callback function and the URL, and every time the URL gets accessed through the web service, the stored function gets called.

For example the application programmer can map his function `string buildHouse(string input)` to the URL `/actions/build_house`. Whenever the URL `/actions/build_house` is accessed through the web service, the function `string buildHouse(string input)` gets called by the web service. All input from the HTTP request is passed to the function as a string and the return value from the function is used as the response to the HTTP request.

Using this mechanism application programmers can easily extend the functionality of the web service specifically for their application's needs.

5 Advantages of Web-Based Linking

The presented approach offers a variety of possibilities. First of all, the approach realizes a simple integration of the scene graph system OpenSG into already existing applications. While it is still possible to realize an integration via dynamic or static linking, a web-based linking via web services enables a simple integration across different languages. As HTTP requests are the common denominator of web-services, they are understood by almost all programming languages. Therefore, the integration of an OpenSG-based visualization into e.g. a Java application is no problem anymore. It offers the possibility to use the right tool respectively programming language of the job.

Furthermore, the web-based integration of OpenSG offers new and simple interaction possibilities. As the camera, a part of the via web-service exported scene graph, can be edited and modified via web browsers new human-computer interactions are possible. For example, a CAVE environment can be controlled and explored via a smart-phone. With adopted, specialized applications this has been possible before, but with our approach it can be realized much easier: a few lines of code start the web-server, the web server takes the root node of the scene graph and automatically exports it.

6 An Example Application

To demonstrate and test our web service we have implemented a plug-in to the open-source interior design application Sweet-

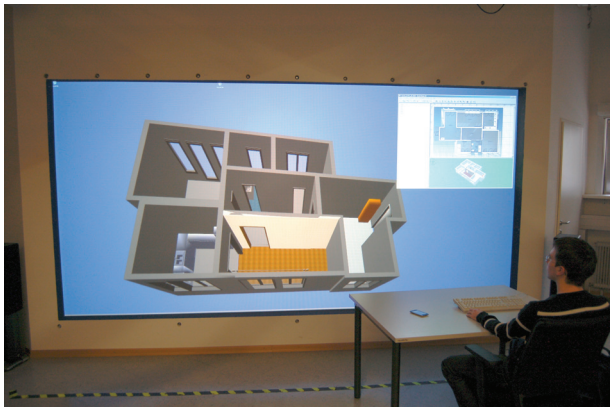

Figure 2: *A plug-in to the interior design application SweetHome3D transfers all changes made to the 2D house plan through our web service to an OpenSG application which displays a 3D representation of the house plan on a big tiled display.*

Home3D[5] (see Figure 3). The Java application SweetHome3D runs on a variety of platforms. It's main features are the ability to draw a 2D house plan and to place furniture on that plan. 3D geometry is generated from those plans and there is a 3D preview of the house plan included in the user interface. A plug-in API is available to access all data that defines the house plan like position and size of walls, materials, details of placed furniture, etc. From this data the geometry for a 3D representation is calculated by the plug-in – which is very similar to what the renderer included in SweetHome3D does for displaying the 3D preview.

With a menu entry our plug-in can be connected to an OpenSG application through our web service. Once connected, our plug-in transfers all changes made in SweetHome3D in real time to the OpenSG application, where it adds the geometry of the 3D representation of the 2D house plan to the scene graph (see Fig. 2, 4).

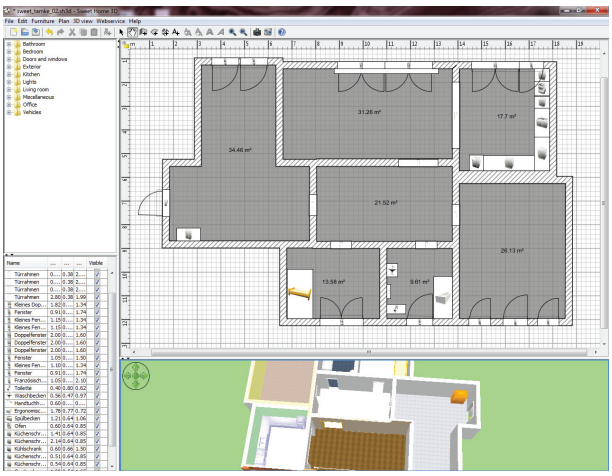

Figure 3: *The Sweet Home 3D software connected to the OpenSG application via the web service plugin.*

Except for starting the web service and exporting a root node in which the SweetHome3D plug-in creates the house geometry through the web service the OpenSG application does not need any additional code for the integration to work.

[5]http://www.sweethome3d.eu

All of the features of SweetHome3D are transferred correctly: The geometry, the colors and the materials with textures as well as the imported geometry representing the interior are displayed in the same way as they are planed in SweetHome3D. Additionally the graphical representation can be much improved over the simple 3D preview of SweetHome3D. The 3D preview does not have advanced features like shadows, etc. which can easily be included in the OpenSG application displaying the 3D representation of the house plan.

The advantage over the included 3D preview is the flexibility of OpenSG. The OpenSG application can display the 3D representation of the house plan in a virtual environment like a CAVE or on a big tiled display. At the same time the model can still be modified in its semantically enriched form of the SweetHome3D editor.

7 Comparison & Conclusion

Web-based Architectures Currently there is an ongoing discussion "REST vs. SOAP" in the SOA world. A comprehensive comparison between RESTful vs. "Big" Web services can be found in [Pautasso et al. 2008].

As described in [Weerawarana et al. 2005], REST services and SOAP services should not be treated as different implementations alternatives for the same solution:

> ...As a rule of thumb, REST is preferable in problem domains that are query intense or that require exchange of large grain chunks of data. SOA in general and Web service technology ... in particular is preferable in areas that require asynchrony and various qualities of services ...

Distributed Graphics Bacu et al. describe a render cluster using the Chromium[6] software [Bacu et al. 2008]. A scene graph is rendered on multiple clients by transferring the low level OpenGL commands over the network. For transferring the render results back to the server a video streaming format is used. Using the Chromium software it is possible to distribute OpenGL based applications without code modifications but in general the amount of data transfered over the network is much greater than using OpenSG. And especially for a CAVE setup a customized application is essential.

Integrated Approaches Zhang and Gračanin propose a framework using web services to combine multiple input providers into one 3D portal application [Zhang and Gracanin 2008].

Web services in combination with a 3D city visualization are described by Hagedorn and Döllner [Hagedorn and Döllner 2007]. The service provides high quality rendered images independent of the client devices graphics capacities. Their frame work is specialized on using high-level geoinformation services without cluster support.

A similar combination of web service and scene graph is described by Behr et al. [Behr et al. 2004] for the instant reality framework[7]. It also uses OpenSG as rendering back end and X3D for the scene description. A difference to our approach is the use of SOAP instead of REST. The flexible instant reality frame work offers a great render performance and can also be used in a CAVE environment. But it is not possible to integrate it into existing applications on the C++ level.

[6]http://chromium.sourceforge.net
[7]http://www.instant-reality.org

Conclusion In this paper we present an integrated approach of a web-based scene graph application interface. In contrast to previous work our solution is

multi-threaded All components of our system are multi-threaded. The scene graph system OpenSG supports multiple threads and the web service can handle requests in parallel.

multi-user capable The implemented synchronization mechanism ensures a consistent scene graph, even if multiple users send several HTTP requests at once.

platform independent OpenSG as well as the web service implementation (libmicrohttpd) are platform independent; i.e. all major platforms (MS Windows, Linux, Mac OSX) are supported.

minimally invasive Any OpenSG application can be upgraded to a web-based service-oriented application by adding a few lines of code. Due to the clean design of OpenSG only the web server's start-up method and its synchronization routine have to be inserted into the existing application. Therefore, the number of changes to an existing application normally involves less than ten lines of code. This minimally invasive modification transforms an OpenSG application into a web server.

adaptable / accessible On the client side only minimal adaptions are needed. The effort required to build a client to a RESTful service – the technique used to transform OpenSG into a service-oriented architecture – is very small as developers can begin testing such services from an ordinary web browser. Due to web support on all platforms and in all languages (Java, C/C++, ...), the integration of an OpenSG-based visualization is no problem anymore. Wrapper and interface code to overcome differences in language and communication is not needed.

All things considered, the presented solution enriches web-based interaction and visualization. The combination of OpenSG and a web server has a beneficial effect on service-oriented scene graph systems, on the integration of immersive visualization environments into existing applications, and on (browser-based or web-based) user interfaces in virtual/mixed reality.

Figure 4: *The SweetHome3D output is transferred with our web service to an OpenSG CAVE application which lets the user walk through a 3D representation of the house plan.*

References

BACU, V., MURESAN, L., AND GORGAN, D. 2008. Cluster Based Modeling and Remote Visualization of Virtual Geographical Space. *Proceedings of the 2008 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing 10*, 416–421.

BEHR, J., DÄHNE, P., AND ROTH, M. 2004. Utilizing X3D for immersive environments. *Proceedings of the ninth international conference on 3D Web technology 9*, 71–78.

BOOTH, D., HAAS, H., MCCABE, F., NEWCOMER, E., CHAMPION, M., FERRIS, C., AND ORCHARD, D. 2004. Web Services Architecture. *World Wide Web Consortium 20040211*, 1–98.

BROWN, N., AND KINDEL, C., 1998. Distributed Component Object Model Protocol – DCOM/1.0.

CHRISTENSEN, E., CURBERA, F., MEREDITH, G., AND WEERAWARANA, S. 2001. Web service definition language (wsdl). Tech. Rep. NOTE-wsdl-20010315, World Wide Web Consortium, March.

CROCKFORD, D., 2006. Rfc4627: Javascript object notation.

FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H., MASINTER, L., LEACH, P., AND BERNERS-LEE, T., 1999. Hypertext Transfer Protocol – HTTP/1.1. RFC Editor.

FIELDING, R. T. 2000. *Architectural Styles and the Design of Network-based Software Architectures*. Univeristy of California, Irvine.

GUDGIN, M., HADLEY, M., MENDELSOHN, N., MOREAU, J.-J., AND NIELSEN, H. F., 2003. Soap version 1.2 part 1: Messaging framework. W3C Recommendation, June.

HAGEDORN, B., AND DÖLLNER, J. 2007. High-level web service for 3D building information visualization and analysis. *Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems 15*, 8:1–8.

JOSUTTIS, N. M. 2007. *SOA in Practice: The Art of Distributed System Design*. O'Reilly, Beijing.

OBJECT MANAGEMENT GROUP. Common Object Request Broker Architecture: Core Specification.

PAUTASSO, C., ZIMMERMANN, O., AND LEYMANN, F. 2008. Restful web services vs. "big"' web services: making the right architectural decision. *Proceeding of the 17th international conference on World Wide Web 17*, 805–814.

REINERS, D., VOSS, G., AND BEHR, J. 2002. OpenSG: Basic concepts. *Proceedings of OpenSG Symposium 2002 1*, 1–7.

SRINIVASAN, R., 1995. Remote Procedure Call Protocol Specification Version 2.

SUN MICROSYSTEMS INC., 2004. Java Remote Method Invocation Specification. http://java.sun.com/j2se/1.5/pdf/rmi-spec-1.5.0.pdf.

VOSS, G., BEHR, J., REINERS, D., AND ROTH, M. 2002. A multi-thread safe foundation for scene graphs and its extension to clusters. *Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization 4*, 33–37.

WEERAWARANA, S., CURBERA, F., LEYMANN, F., STOREY, T., AND FERGUSON, D. 2005. *Web Services Platform Architecture : SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall PTR, march.

ZHANG, X., AND GRACANIN, D. 2008. Streaming web services for 3D portal applications. *Proceedings of the 13th international symposium on 3D web technology 13*, 23–26.