A distributed architecture for searching, retrieving and visualizing complex 3D models on Personal Digital Assistants

A. Sanna, C. Zunino, F. Lamberti

Dip. di Automatica e Informatica - Politecnico di Torino,
C.so Duca degli Abruzzi 24, I-10129, Torino (Italy)

Abstract

Mobile devices are significantly changing the human-computer interaction. In particular, the ubiquitous access to remote resources is one of the most interesting characteristics achievable by using mobile devices such as Personal Digital Assistants, cellular phones and tablets.

This paper presents an architecture that allows users to search and visualize complex three dimensional models over Personal Digital Assistants. A peer-to-peer network of brokers manages queries for searching objects among several data providers. The object selected for visualization is forwarded to a specialized graphics providers; this provider allows the users to investigate the object remotely rendering the scene and sending back to the Personal Digital Assistant the computed image. The user can interactively analyze objects that would not be otherwise visualizable locally on the Personal Digital Assistant.

Key words: mobile virtual reality, 3D model visualization on Personal Digital Assistants, integration between mobile technologies and distributed architectures.

1 Introduction

This project aims to develop a distributed architecture able to provide the user an effective tool for searching, retrieving and remotely displaying 3D models.

Email addresses: andrea.sanna@polito.it (A. Sanna), claudio.zunino@polito.it (C. Zunino), fabrizio.lamberti@polito.it (F. Lamberti).

The interaction between the user and complex geometries via mobile devices is a new and challenging task. On one hand, mobile devices provide an ubiquitous access to remote resources, on the other hand computational and storage local resources cannot allow to manage, display and investigate objects described by hundred thousand textured primitives.

Some tools for designing realistic 3D graphics applications for mobile devices, and in particular for Personal Digital Assistants, already exist. For instance, PocketGL is a 3D graphics library similar to OpenGL (Woo et al 1999) for PocketPC written in C/C++ which allows to draw 3D objects and manage transformations. Despite of the evolution of hardware and software for handheld devices, realistic visualizations of large and complex models are possible only demanding the rendering task to remote servers. Silicon Graphics, Inc. proposed a commercial solution called Vizserver that can be used to provide application transparent remote access to high-end graphics resources; moreover, Vizserver enables collaborative visualization functionality for multiple simultaneous users. Kilgard (2002) proposed an OpenGL render server facility to generate images on remote machines. In addition, it allows varying image sizes and hybrid implementations (e.g., combining local and remote rendering). Other non commercial solutions have been proposed, but they often lack architecture/application independence. A parallel distributed volume-rendering application has been presented in Bethel (2000); this application called Visapult provides interactive remote visualization exploiting two components: a parallel volume-rendering engine and a OpenGL-based viewer. Engel et al (2000) proposed a framework for remote visualization; compressed images are sent to Java clients from a visualization server while the interaction with the application is obtained by sending requests in the Common Object Request Broker Architecture format. A solution studied for time-varying data has been proposed in Ma and Camp (2000); two daemons implement the architecture: the former receives data from a renderer process, compresses information and sends it to the latter that decompresses the data and presents the results to the user. A generic solution for hardware-accelerated remote visualization has been recently presented in Stegmaier et al (2002); this solution is based both on the concept of dynamic linking and the functionalities of the OpenGL extension to the X Window system. OpenGL-based applications run locally to a render server but the output is configured in order to be sent to a display server. Custom functions have been implemented in order to accomplish this mechanism. The system uses the Virtual Network Computing client-server infrastructure for image transmission.

This paper focuses both on the problem to search and retrieve resources (3D models) over a distributed architecture implemented as a hybrid peer-to-peer network and on visualizing objects on Personal Digital Assistants (client peers) by demanding the rendering task to ad-hoc service nodes belonging to the architecture. The user connects to the framework via special peers named

gateways which provide an access to the system. Users's queries are managed by broker peers that forward queries to data providers. Data providers offer data (3D models) and metadata (catalogues describing resource characteristics). Query results are presented to the user which can choose the model to be displayed. At this point, the model is sent to a graphics service provider (in this case high-end clusters based on the Chromium technology) that sets a direct connection with the Personal Digital Assistant. The user can analyze the model using a local visualization application; all commands are sent to the remote rendering software that computes the animation frames and interactively sends them back to the user.

The paper is organized as follows: Section 2 briefly reviews the Project Juxtapose (JXTA) and Chromium technologies. The proposed architecture is described in detail in Section 3 and some remarks about system performance are reported in Section 4.

2 Background

This section briefly reviews two technologies used to design and implement the proposed architecture. In particular, JXTA, used to develop broker, data provider, and gateway peers, is presented in Section 2.1, while Chromium, that is the underlying layer of service providers for the remote rendering is reviewed in Section 2.2.

2.1 The Project Juxtapose - JXTA

JXTA (http://www.jxta.org) is an open-source project that defines a set of protocols for ad-hoc, pervasive, peer-to-peer computing. Protocols of the JXTA project establish a virtual network overlay on top of the Internet and non-IP networks, allowing peers to directly interact and organize independently of their network location. A deep description of JXTA can be found in Gong (2001).

On one hand, many peer-to-peers are built for delivering a single type of service. For instance, Gnutella (see the Gnutella protocol specification) provides generic file sharing and instant messaging. Given the diverse characteristics of these services and the lack of a common underlying peer-to-peer infrastructure, each peer-to-peer software vendor tends to create incompatible systems. On the other hand, JXTA technology is designed to be independent of programming languages, system platforms, and networking. JXTA architecture is divided into three layers: platform Layer (JXTA Core), services layer, and ap-

plications layer. The platform layer encapsulates minimal and essential primitives that are common to peer-to-peer networking. It includes building blocks to enable key mechanisms for peer-to-peer applications, including discovery, transport, the creation of peers and peer groups, and associated security primitives. The services layer includes network services that may not be absolutely necessary for a peer-to-peer network to operate, but are common or desirable in the peer-to-peer environment. Finally, the applications layer includes implementation of integrated applications, such as peer-to-peer instant messaging, document and resource sharing, and so on.

A JXTA architecture consists of a series of interconnected nodes (called peers). Peers can include sensors, phones, Personal Digital Assistants, as well as Personal Computers, servers, and supercomputers. Each peer operates independently and asynchronously and it is uniquely identified by a peer Identifier. Peers can be organized into peer groups, which provide a common set of services (for instance, document sharing or chat applications). JXTA peers advertise their services in Extensible Markup Language (Harold and Means 2002) documents called advertisements. Advertisements enable other peers on the network to learn how to connect to, and interact with, the peer's services. JXTA peers use pipes to send messages to one another. Pipes are an asynchronous and unidirectional message transfer mechanism used for service communication. Messages are simple Extensible Markup Language documents whose envelope contains routing, digest, and credential information. Pipes are bound to specific endpoints, such as a port and the associated address.

Three essential aspects distinguish the JXTA architecture from other distributed network models: the use of Extensible Markup Language documents (advertisements) to describe network resources, the abstraction of pipes to peers, and peers to endpoints without reliance upon a central naming/addressing authority such as Domain Name Systems, and a uniform peer addressing schema.

Peers (commonly named edge peers) are not required to have direct point-to-point network connections between themselves. Intermediary peers (called rendezvous peers) may be used to route messages to peers that are separated due to physical network connections or network configurations (e.g., firewalls, proxies). The JXTA protocols describe how peers may publish, discover, join, and monitor peers and peer groups. A peer group provides a set of services called peer group services. JXTA defines a core set of peer group services. Additional services can be developed for delivering specific tasks. Two peers must be part of the same peer group to interact.

The core peer group services include the following: discovery service (used by peer members to search for peer group resources) membership service (used by current members to reject or accept new group membership application),

access service (used to validate requests made by one peer to another), pipe service (used to create and manage pipe connections between the peer group members), resolver service (used to send generic query requests to other peers), and the monitoring service (used to allow one peer to monitor other members of the same peer group).

2.2 Chromium

Chromium (Humphreys 2003) is a technology derived from the WireGL (Humphreys et al 2001) project. The basic goal of these two architectures is to combine a large amount of "inexpensive" rendering hardware (e.g., graphics adapters equipped by last generation Graphics Processing Units) in a computing cluster to tackle real-time requirements of advanced graphics applications. From the user point of view, Chromium acts as driver that replaces the native OpenGL calls; therefore, traditional OpenGL-based applications can take advantage from Chromium without needing recompilation.

While WireGL was introduced to simply display large data sets on tiled displays, Chromium is able to support generic hybrid parallel rendering algorithms. These algorithms are implemented developing ad-hoc Stream Processing Units that can determine both how the workload has to be divided among cluster nodes and how the parts of a frame have to be managed on a single node. A master machine has to be identified for each Chromium cluster; a Stream Processing Unit, running on the master machine, defines the parts of image to be computed on each node (equipped by a Graphics Processing Unit) of the cluster. A specific Stream Processing Unit called *tilesort* implements the sort-first rendering schema also used by the WireGL library. This tilesort Stream Processing Unit redefines the OpenGL functions so that they are packaged and sent over a network to a group of destination nodes. Several Stream Processing Units can be linked into an arbitrarily long chain to perform more complex tasks: Stream Processing Units basically are a collection of OpenGL calls to be executed on a given graphics hardware.

The parts of image computed by Graphics Processing Units can be either reassembled by a Stream Processing Unit called *compositor* to build an image on the screen or sent to a Stream Processing Unit called *integrator* in order to pilot multi-screen (tiled) displays. Figure 1 shows a typical Chromium configuration where a tiled display is managed.

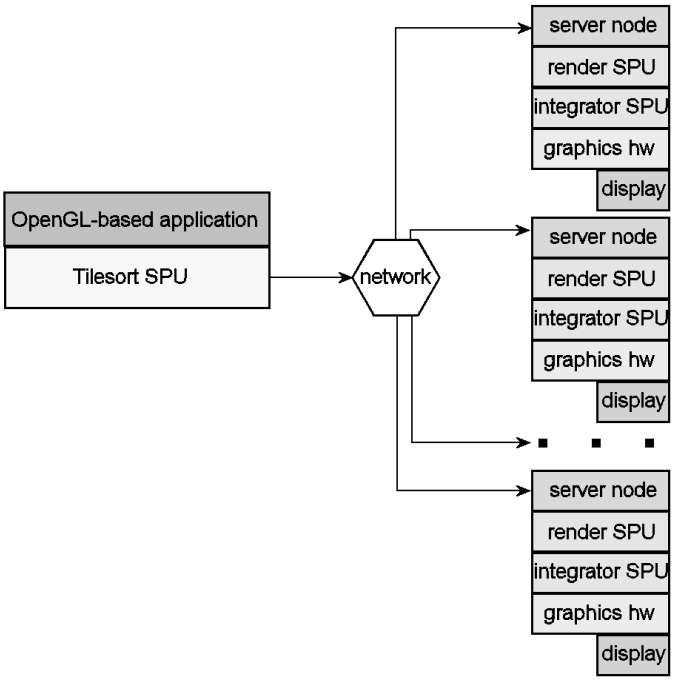

Fig. 1. A Chromium configuration.

3 The proposed architecture

The proposed architecture identifies four different roles (see Figure 2):

(1) brokers;
(2) providers;
(3) gateways;
(4) clients.

Brokers are the "kernel" of the entire infrastructure. Although the whole system cannot be considered as a peer-to-peer network (in a pure peer-to-peer all peers have the same role), brokers are connected each other in order to form a peer-to-peer. Brokers receive client queries, forward them to data providers and manage remote visualization services. Brokers can be realized by means of rendezvous peers of JXTA; each broker is in charge to manage a set of providers and a broker knows a certain number of other neighboring brokers.

Providers can be of two different kinds: data and service providers. Data providers offer data (3D models) and metadata (catalogues). JXTA already uses a native Extensible Markup Language database to store advertisements, in particular, an open source database called Xindice (http://xml.apache.org/xindice/) is embedded. Data providers also store resource descriptions as Extensible Markup Language documents in Xindice, whereas data (3D models) are compressed and saved on the disk. Xindice has been chosen among the available

Extensible Markup Language databases as it provides four characteristics: is written in Java, can be embedded within the application, is distributed under the open source licence, and provides a Extensible Markup Language interface.

A resource is catalogued storing name, author's name, date and version of the model, and a textual description; all these fields denote a resource descriptor. Data providers are realized by edge peers of JXTA. Service providers are in charge to remotely visualize 3D models. The broker network allows to copy the model to be displayed from a data provider to an available service provider, afterward a graphics application based on OpenGL and Chromium uses Graphics Processing Units of a high-end performance cluster to render the object. A direct connection between the service provider and the user (connected to the system by a mobile device such as a Personal Digital Assistant) is established at this point. The user can iteratively inspect and visualize the model (roto-translation operations are allowed) independently of the complexity of the model itself.

Gateways (realized by edge JXTA peers) provide the access points to the architecture and allow to clients to "see" the system resources; in particular, a form allows the user to search models according to the parameters describing the resources. The gateway interface does depend on its implementation, for instance Hypertext Transfer Protocol, Simple Object Access Protocol, and so on. In particular, it has been developed a gateway (named web-peer) that allows the user to connect the system by Hypertext Transfer Protocol. The web-peer contains an embedded version of the web server Apache Tomcat.

Clients are not really part of the architecture but they get access to services by means of gateway peers. In this case, a client has to be able to perform Hypertext Transfer Protocol connections and must know the Uniform Resource Locator of an available gateway peer; therefore, any machine enabled to connect to the Internet can be an user's front-end. When a client has retrieved a model by using the distributed architecture, a point-to-point link can be established between the client and a service provider; this direct connection is denoted in Figure 2 by dashed lines. The user interface is divided in two different parts: search form and the visualization application (see Section 3.3).

3.1 Broker and data provider architecture

Brokers and data providers have been implemented as Java applications in order to fulfil platform independence requirements. These applications are organized as a collection of modules; a module manager has to cope both with modules and the events generated from the modules themselves. In particular,

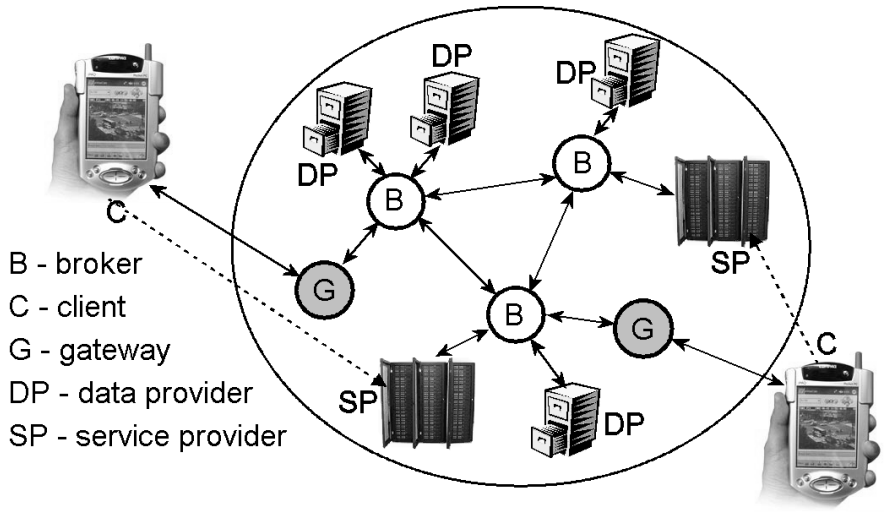

Fig. 2. The system architecture.

the manager has to load, start and stop every module. An application interface is placed at the top of this architecture; the application interface can be used to: monitor the state of the network (a list of brokers, data providers and gateways currently active is provided), add/remove models in the database of a data provider, and configure broker and data provider peers. The application interface cannot be acceded by clients.

Figure 3 shows broker and data provider architecture. The module manager provides information for the other modules and the application interface, while the event manager routes information from modules to the interface. The interaction between the interface and a module is, in general, bi-directional and it is always started from the interface. The storage module stores and indexes the resources (in general brokers differ from data providers as they do not have this module). Technological and implementation details are hidden to the other modules by means of a Java interface that provides a generic way to store, search and retrieve data.

All system resources (models) are described by means of an (potentially variable) Extensible Markup Language document/schema. For this reason, the storage module has been developed by a native Extensible Markup Language database. The storage module accesses to the database only by means of an Extensible Markup Language Application Program Interfaces, in this way, it is compliant with all databases providing this kind of interface. A resource is considered, in general, as composed by two parts: a set of files and the metadata describing the resource itself. Files are compressed and stored in a directory on the disk (Xindice does not well manage large dimension binary files) while the metadata are stored in the database. The correspondence between data and metadata is collected by a resource descriptor.

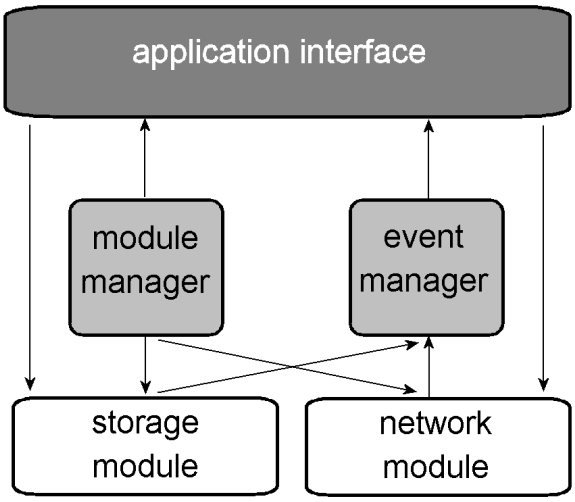

Fig. 3. Broker and data provider architecture.

The network module contains all the code needed to build and maintains the network infrastructure. It acts as the interface between the network and the other elements of the application and it is in charge to autonomously maintain the (logical) network configuration. The network module is the only element of the application that knows the underlying network technology (JXTA in this case).

When a broker receives a query from a client (through a gateway):

- propagates the query to data providers directly connected (if they exist);
- propagates the query to neighboring brokers (if they exist) using the walkers method Traversat et al (2003);
- reassembles results and presents the list of available data providers to the client.

In the same way, a broker propagates the query when a walker is received. Moreover, each broker has a registry where it stores the status of the service providers and all parameters needed to establish a remote visualization session (the client has to know the address of the service provider, number of port for the connection, and so on). A service provider can be available or not available (see Section 3.2) and a thread is in execution for managing each service provider directly connected to the broker. Threads are in charge to receive and record status changes coming from service providers.

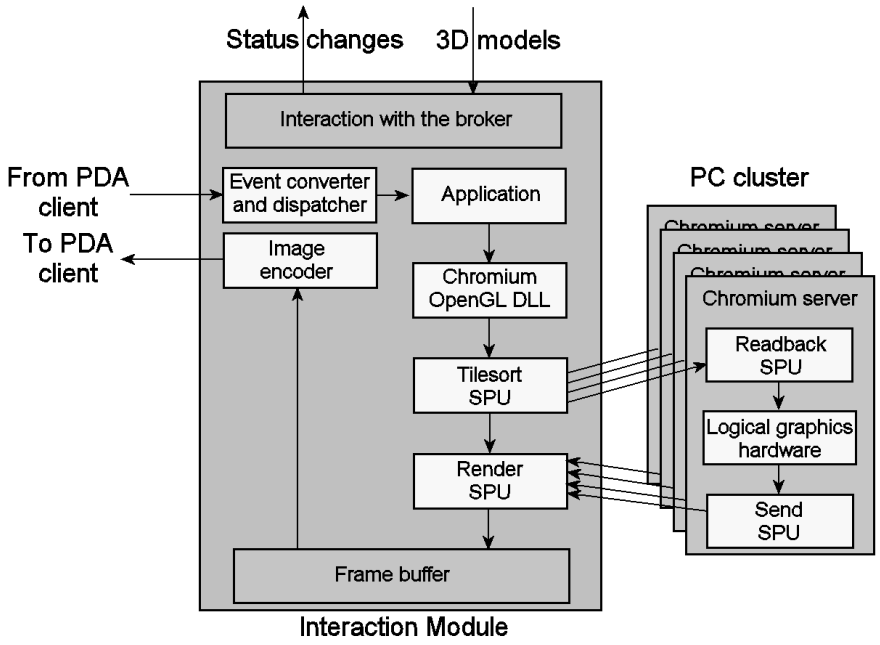

Fig. 4. Service provider architecture.

3.2 Service provider architecture

The architecture of service providers is shown in Figure 4. A service provider can be available (ready to display a 3D model) or unavailable (it is currently rendering a 3D model). As the rendering process directly involves graphics adapters that are not sharable resources just one model can be rendered at a time, and hence just one client can be served at a time. When the user selects a model from a data provider, the broker downloads the model from the provider and then checks for available service providers. The broker checks before the availability of service providers connected directly; if they do not exist, or are unavailable, a query to the other brokers for available service providers is sent. If the list of resulting service providers is empty, an error message is sent to the client, otherwise, the first service provider is selected sending to it the 3D model. Each service provider of the list is described by an Extensible Markup Language document that reports all parameters needed to establish a remote visualization session. The broker sends the client these parameters that are used to manually set the local visualization application. At this point, the service providers signals to the broker at whose is directly connected to be in the unavailable status and then establishes a **direct link** with the client.

Service providers generally use graphics hardware resources of a high-end cluster that hosts the remote visualization application; the remote visualization application is responsible for streaming image-based visualization frames generated by the rendering process. A module named Interaction Module controls

the communication between the visualization application and the Personal Digital Assistant client. In particular, it receives roto-translation commands that allow the user to inspect and analyze the model and returns the animation frames. Commands coming from the Personal Digital Assistant are translated for the remote visualization application that has been developed using OpenGL and the OpenGL Utility Toolkit (Kilgard 1996).

A command/event generated from the client handheld triggers the submission of a command packet to the Interaction Module over the wireless communication link. On the server side, this packet, which was originally encoded as a triplet (command code, x position, y position), is translated into a command complaint for the visualization application, according to a conversion table, and then passed on to the callback functions, which in turn will adjust the mapping and rendering parameters. Then, the OpenGL directives are locally executed, exploiting the processing capabilities available at the rendering site. The rendering task is split across the cluster of Personal Computers by a Chromium Stream Processing Unit; each Personal Computer is in charge to render one or more portions of a frame. When the graphics adapter of a Personal Computer has terminated to render its part sends back the content of its video memory to the Interaction Module (the Send Stream Processing Units). The Interaction Module reassembles the whole frame (the Render Stream Processing Units) and puts it in the frame buffer. Finally, the content of the frame buffer is coded and sent to the client. The coding schema does depend both on the band of the wireless link and the computational power of the client. If the band is not a critical issue (as for instance in IEEE 802.11b communication channels) frames can be sent to the client in raw format; in this way there is no the latency time for the decoding phase on the client side. Otherwise, a compression step can be performed but the handheld device has to able to decompress without noticeable delays.

3.3 The client visualization application

The local client visualization application schema is shown in Figure 5. As above mentioned, clients are not strictly part of the architecture but they act as front-ends supporting the user in the interaction with (potentially) extremely complex 3D objects.

The interaction between the user and the device is performed by a local visualization application. The user can tap a pen over the display or can use the directional pad that implements the traditional keyboard arrow-keys functionalities. Moreover, four application buttons are available, which can be completely customized via software.

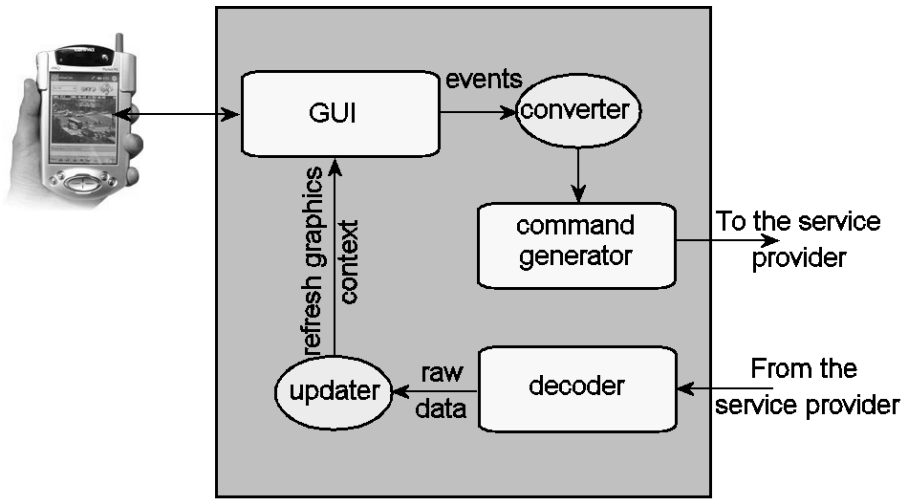

Fig. 5. The local visualization application schema.

The visualization application has been developed using the Microsoft embedded Visual C++ environment for WindowsCE 3.0 and Microsoft Windows Platform Software Development Kit for PocketPC. The application is composed by three modules:

(1) the graphics user interface;
(2) the command generator;
(3) the image decoder.

The interface handles the events generated by input devices (e.g., pen, navigation pad and application buttons) and forwards them to the converter that translates these events in commands to be sent to the service provider (a triplet containing the command code and the screen position). The command generator receives commands from the converter and sends them to the service provider; commands are encoded in a packet suitable for transmission through a connection over a low bandwidth channel.

A set of control areas are placed around the rendered frame (see Figure 6) and they provide the same control capability as the remote OpenGL visualization application. Furthermore, the user can display a comprehensive set of information related to the status of the visualization (camera position, camera orientation, texture, lights and shadows status, and so on) as well as all the information related to the current scene (including polygons count and frame rate).

The image data decoder receives the encoded information streamed by the service provider over a Transmission Control Protocol channel established on a wireless link. A support for different coding schemas has been implemented in order to take into account the heterogeneity of the available communication

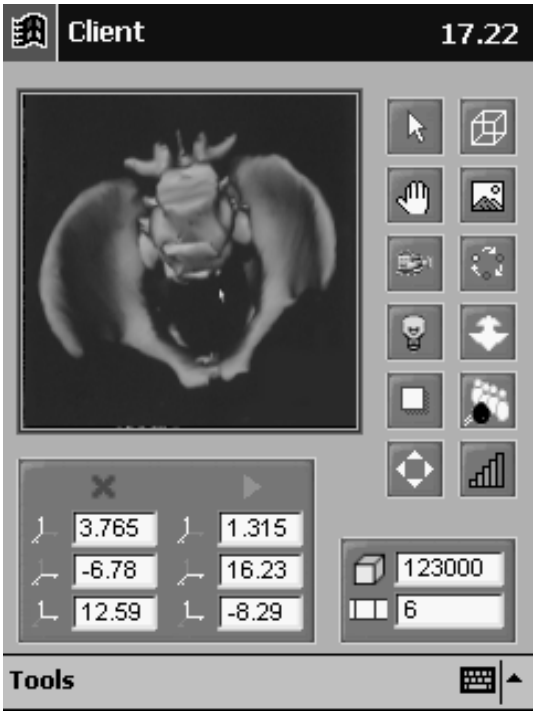

Fig. 6. The graphics user interface. A set of control areas are placed all around the frame.

channels; in particular, the support for both compressed and uncompressed image data streams has been included (see Section 4).

4 Remarks

A complex infrastructure has been built to test system performance. In particular, a broker has been placed in every site of the Politecnico di Torino in the north west of Italy: Alessandria, Aosta, Biella, Ivrea, Mondovì, Torino and Vercelli. Each broker manages two or three data providers, while one service provider was connected to the Torino's broker (that also manages a gateway peer). The service provider has been deployed placing the Interaction Module on a "master" machine in charge of distributing the rendering workload on a gigabit-ethernet cluster of eight Personal Computers hosting 1.4GHz AMD Athlon processors endowed with NVIDIA GeForce2 graphics adapters. The selected Personal Digital Assistant is the Compaq iPaq H3630 with a 206MHz StrongARM processor, a 12-bit color depth touch screen display (240x320 pixels, 2.26x3.02 inches), 32MB RAM, 16MB Flash ROM, and the Microsoft PocketPC operating system.

The system analysis can be divided in two parts: data retrieving and data visualization. Data retrieving performance strongly depends both on network

connections and the underlying platform. If network connections affect response times in a way directly proportional to the bandwidth, it is much more difficult to evaluate the impact of the platform (JXTA in this case) and of the search strategy on the whole architecture scalability. The walker method used to implement the search strategy provides performance better than traditional flood algorithms used in peer-to-peers such as Gnutella (Traversat et al 2003). Moreover, JXTA designers have the goal to provide developers a platform able to scale, at least, according to the following parameters (Sun JXTA Engineering Team 2002):

- 1.5 million peers.
- 300,000 simultaneous connected peers. While there may be 1.5 million peers set up to use the network only 20% are assumed to be active at any one time.
- 1.000 contents per peer. Each peer is sharing in average 1.000 documents.
- 1.5 MB size per content on average. Shared content are expected to be image files or business documents.
- 90% peers using the Dynamic Host Configuration Protocol. Peers machines are assigned a new address every time they reboot.
- 70% peers behind Network Address Translators. Most peers will not be able to act as Relay or Rendezvous.
- 2 hours average connection time per peer. A peer is assumed to be active for an average of 2 hours.
- 120 peers/minute "churn" rate for peers joining and leaving the network during prime-time usage.
- 30.000 peers always on-line. 10% of the on-line peers will always be connected. These peers are primary target for relay and rendezvous.
- 90% peers connected via broadband.
- Transaction rates:
  - Discovery request: 1 request every 30s per connected peer
  - Content transfers: 1 transfer every 2 minutes per connected peer
  - Pipe Resolution: 1 resolution every 15 minutes per connected peer
  - Message polling: 1 poll every 5s for a peer to query its Relay for incoming messages

In our tests the same database of about one hundred models has been replicated on each data provider; therefore, delays due to database queries have been almost unnoticeable. Results are incrementally presented to the client and responses times have been from some tenths of second to some tens of seconds. In the same way, transfer delays from the selected data provider to the broker and from the broker to the service provider depend both on network connections and model size. Files of some megabytes involved transfers of almost some tens of seconds.

The latency for the data visualization mainly depends on:

(1) delay due to transmit a command from the mobile device to the Interaction Module;
(2) the "application" of the command that involves the generation of a new image;
(3) the transmission of the computed image to the mobile device;
(4) the update of the visualization area on the mobile device.

Of course, the service provider has to able to render in real time the models independently of their complexity. A lot of parameters affect the latency time of the visualization on the client; in order to quantify this latency value, without loss of generality, let us assume a scenario such as the one depicted in Figure 7: the frame resolution has been set to 150x150 pixels (this size allows both to clearly analyze objects and to maintain a significative interface around the frame) and the Personal Digital Assistant is connected by a IEEE 802.11b link.

Fist of all, the average time to transmit a command from the mobile device to the Interaction Module has to be computed. An event, for instance, is generated when the user taps with the pen over the screen: the event is inserted in the application event queue. The insertion time of the event in the queue and the extraction time of the event from the queue are almost negligible (if the queue is not overloaded). The average time to translate the command and to transmit it over the wireless link to the Interaction Module is about 6 ms.

A new image has to be computed when the Interaction Module receives a command. For the considered example, the new image has been computed, in the worst case, after 2 times the average rendering time, that is $2 \cdot 30$ ms.

Finally, the times of frame transmission and frame visualization have to be computed. The transmission time over the wireless link is about 90 ms (in this case no coding schema is used) while the visualization delay (due to the copy of the image from the receipt buffer to the visualization buffer plus the update of the visualization area) is about 80 ms.

The whole visualization latency limits the maximum frame rate that can be achieved, on the selected Personal Digital Assistant device, to seven frames per second. We have experimentally verified that the average bandwidth available over an IEEE 802.11b communication channel is sufficient to achieve the maximum throughput in terms of frames per second. Therefore, with IEEE 802.11b, we were not forced to adopt any particular coding schema to transfer the image-based stream from the Interaction Module to the visualization interface running on the client.

On the contrary, image data can be transmitted in raw format, with the advantage that no further processing steps have to be performed on the client side prior the visualization. On the other hand, performances are different in

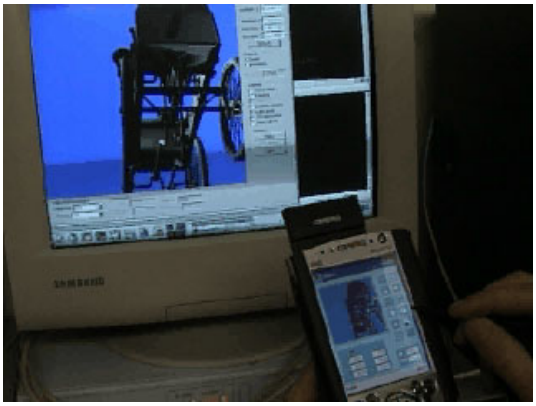

Fig. 7. The frame shows the visualization on the Personal Digital Assistant connected by a IEEE 802.11b link and, on the background, the image reassembled on the master machine's console.

a General Packet Radio Service-based scenario. In this case, we have experienced a mean throughput in the downlink direction of just 42.5kbps. This throughput is considerably lower than the one needed to sustain an acceptable frame rate, and thus network bandwidth becomes a serious bottleneck. To satisfy interactivity requirements, the adoption of compression techniques becomes mandatory.

Within the proposed framework, we have employed both lossless and lossy compression techniques for image data transmissions. In particular, we have developed portings for the PocketPC Operating System of both the zlib lossless compression library and the libjpeg lossy compression library. Unfortunately, the selected Personal Digital Assistant is not able to effectively decompress the incoming data stream and this strongly affects the interactivity.

5 Conclusions and future works

This paper presents a complex framework for retrieving and remotely displaying complex 3D models. A peer-to-peer network of brokers allows mobile users to search an object among several data providers. The object to be displayed is forwarded to a graphics service provider that remotely renders the scene and sends the frames to the Personal Digital Assistant client. At this point, the user is able to analyze and investigate the model in an interactive way even if the object is extremely complex and would not be otherwise visualizable locally on the Personal Digital Assistant.

A lot work is still necessary to totally integrate mobile devices in grid and peer-to-peer architectures. In the proposed approach clients are not really part of the architecture as they connect by ad-hoc peers named gateways. On the other hand, clients could be themselves JXTA peers (Maibaum and

Mundt 2002) taking advantage of all characteristics (security, reliability, data protection and so on) of this technology. Moreover, a sort of quality of service could be considered managing service providers. At the moment, the entire cluster is reserved for the model visualization independently of the object complexity; the user could choose to lease only a certain number of machines of the cluster according with the complexity of the model to be rendered. In this way, a cluster could be used to remotely render more than one model at a time.

Finally, the employment of more recent mobile devices and new wireless network technologies (e.g., IEEE 802.11g) could strongly impact on the client visualization application. Both larger resolutions and higher frame rates could be achieved providing the user a more effective and exciting mobile virtual reality experience.

Acknowledgements

Authors wish to thank Marco Ciancimino and Antonino Fiume for their support in the design and the developing of the proposed architecture.

References

[Woo et al (1999)] Woo, M., Neider, J., Davis, T., Shreiner, D., OpenGL Architecture Review Board, 1999. OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2, 3rd edition, Addison-Wesley.

[Kilgard (2002)] Kilgard, M., 2002. GLR, an OpenGL Render Server Facilitiy. Silicon Graphics, Inc. (http://reality.sgi.com).

[Bethel (2000)] Bethel, W., 2000. Visualization viewpoints: Visualization dot com. IEEE Computer Graphics and Applications 20:3, 17-20.

[Engel et al (2000)] Engel, K., Sommer, O., Ertl, T., 2000. A Framework for Interactive Hardware Accelerated Remote 3D-Visualization. In Proceedings of EG/IEEE TCVG Symposium on Visualization VisSym00, 167-177..

[Ma and Camp (2000)] Ma, K.L., Camp, D.M., 2000. High performance visualization of time-varying volume data over a wide-area network status. In Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM), IEEE Computer Society Press.

[Stegmaier et al (2002)] Stegmaier, S., Magallon, M., Ertl, T., 2002. A Generic Solution for Hardware-Accelerated Remote Visualization. In Procceedings of EG/IEEE TCVG Symposium on Visualization VisSym02.

[Gong (2001)] Gong, L., 2001. JXTA: A network programming environment, IEEE Internet Computing, 5, 88–95.

[Harold and Means (2002)] Harold, E.R., Means, W.S., 2002. XML in a Nutshell, 2nd edition, O'Reilly & Associates.

[Humphreys (2003)] Humphreys, G., 2003. Chromium Documentation: Version 1.2 http://www.cs.virginia.edu/ humper/chromium_documentation/

[Humphreys et al (2001)] Humphreys, G., Eldridge, M., Buck, I., Stoll, G., Everett, M., Hanrahan, P., 2001. WireGL: A Scalable Graphics System for Clusters, Proceedings of the ACM Siggraph.

[Traversat et al (2003)] Traversat, B., Abdelaziz, M., Pouyoul, E., 2003. Project JXTA: A Loosely-Consistent DHT Rendezvous Walker, http://www.jxta.org/docs/jxta-dht.pdf

[Kilgard (1996)] Kilgard, M.J., 1996. The OpenGL Utility Toolkit (GLUT) Programming Interface API Version 3. http://www.opengl.org/developers/

[Maibaum and Mundt (2002)] Maibaum, N., Mundt, T, 2002. JXTA: a technology facilitating mobile peer-to-peer networks, International Mobility and Wireless Access Workshop, 7–13.

[Sun JXTA Engineering Team (2002)] Sun JXTA Engineering Team, 2002. JXTA Platform Scalability Proposed Design, Draft Version 0.5.