Real Time Visualization of Robot State with Mobile Virtual Reality

Peter Amstutz and Andrew H. Fagg
Department of Computer Science
University of Massachusetts
Amherst, Massachusetts 01003
{amstutz, fagg}@cs.umass.edu

March 7, 2002

Abstract

With the deployment of large, distributed networks of cameras and other sensors, it is becoming necessary to also address the issue of how to effectively present the large volume of gathered information to a user. One approach to this problem is to summarize the information gathered by these sensors using a three-dimensional, virtual environment which enables a user to engage her own natural abilities to absorb the spatial information inherent in the data streams. Tasks that require a user to have access to this information while in the field (e.g., search and rescue) point toward the need for portable solutions to this problem. This paper presents a virtual/augmented reality architecture that has been explicitly designed for use with a fully-portable, wearable computing system. A critical component of this system is a network-based mechanism for the representation of virtual objects and the live communication of changes in their state to users located elsewhere on the network. By presenting virtual objects in a uniform manner over the network, it becomes easy to construct new dynamic, virtual environments that reflect the state of robots or humans within the real environment. We demonstrate the utility of the architecture through several robot and human tracking examples.

1 Introduction

Widely distributed sensor networks are becoming commonplace in our environments. Web-available cameras allow anyone with an Internet connection to peek in real-time into office spaces, national parks, and sporting events; security guards have access to tens or hundreds of cameras; and robot explorers carry cameras and other sensors into dangerous or out-of-reach areas. However, the individual sensors generally provide constrained viewpoints from which to experience these locations. Furthermore, there is often very little information that makes explicit the spatial relationship between the different sensors. These difficulties limit the ability of the user to immerse herself in the experience provided by the space, to construct coherent models of the spatial geometry, and to make real-time, life-critical decisions. One approach to solving this problem is to synthesize a three-dimensional virtual experience from the live sensor streams.

Many of the application areas also require access to this real-time information while the user is on the move. For example, security guards on patrol need access to information about individuals moving within a building; and search and rescue personnel located in the field require access to summaries of survivor and hazard data gathered by large numbers of robots that are assisting in the efforts. These types of tasks have led us to focus on solutions that allow full mobility of a user.

In order to construct such a system for the three-dimensional presentation of distributed, live sensory data, a wide variety of problems must be solved. First, individual sensory processing systems must be capable of extracting the information of interest (for example, the presence of a nearby moving object). Second, information from groups of sensors must be fused, taking into account the relative (and possibly dynamic) positions of the sensors, in order to compute estimates of the three dimensional location of the features of interest. Third, the data that is gathered must be communicated in some form to the mobile computers that are carried by users, taking into account bandwidth limitations. Finally, the information must be rendered within an egocentric frame of reference at a video frame rate such that virtual object movements on a head-mounted-display (HMD) correspond in a convincing manner to the movements of the user's head.

In this work, we leverage ongoing work at UMass in the development of smart spaces. *Containment units* provide the organizational mechanism by which multiple sensors (including visual, thermal, and acoustic sensors) are marshaled together in a robust manner in order to provide continuous tracking of a subject throughout the smart space [1, 2]. High level sensory state (including the position of the tracked subject) is made available through a JINI service, thus enabling access to the information by a wide variety of applications.

This paper describes our approach to constructing a versatile framework for the representation, communication, and rendering of dynamic, three dimensional object state information. Virtual objects are represented in a distributed fashion, with components possibly spread across multiple computers located on the network. Objects may represent volumetric primitives (e.g., boxes, spheres and meshes) or may constitute collections of objects. Live sensory data is represented through the real-time update of the position or other state of these objects. Application programs with an interest in specific objects register this interest with the hosting server. As object state is updated, the server communicates these state changes through an XML-based protocol. The three dimensional rendering of the virtual environments is accomplished using a fully-mobile, wearable computer equipped with a HMD, a head orientation sensor, and wireless networking capability.

2 Previous Work

Mobile virtual and augmented reality has been the focus of several wearable computing-based research efforts. Feiner et al. demonstrated a fully portable implementation of an augmented reality system for a tour guide task [3]. This system relied on the use of GPS/dGPS, a digital compass, and tilt sensors to infer the head pose of the user. Three dimensional textual information was superimposed on top of the user's field of view in real time as the head pose changed, thus simulating the existence of the text within the real world.

Starner et al. and Jebara et al. described the use of augmented reality for constructing tutoring systems for copy machine maintenance and billiards playing [4, 5]. In both examples, three dimensional graphical objects were painted on top of images captured with a stereo camera pair. These images were then presented to the user using an opaque stereo display. The alignment of the virtual objects with the captured view was accomplished through the application of image processing techniques to identify landmarks in the images. Although the non-see-through display eliminates the problem of calibration between the cameras and the HMD, this work required the use of off-board computation for the image processing.

Foxlin and Harrington introduced a wearable computer interface that relied on a head tracking system to scroll the viewport of HMD through a space of windows that surrounded the user [6]. Their system utilized a set of gyroscopic sensors that were corrected by a compass and inclinometers to determine head orientation. A sonar-based sensor was used to determine the position of the user's hand relative to the head. This mechanism allowed the user to direct the position of the cursor in three dimensions using hand movements.

Gibson and Murta have explored the use of reflection mapping, light sourcing and shadowing extracted from the real-world environment to better integrate synthetic objects into a fixed-frame scene [7]. Their system is able to render fairly complex models at 10 FPS, but requires an SGI Onyx2 workstation to do so.

3 A Network-Distributed Representation for Virtual Environments

The focus of this work is the design of a network-distributed object system that facilitates the representation and dissemination of sensor-driven virtual environments. At one end of the network are sensors which may be anything from complex autonomous robots to simple, fixed-location cameras. This network also includes the visualization platform implemented on a wearable computer which must communicate with the sensor system, extract information and present it to the user in an intuitive, graphical manner.

For ease of explanation we will utilize an illustrative example: the network-accessible, three-dimensional visualization of the state of the UMass robotic torso (figure 1). The torso consists of a pair of seven-degree-of-freedom jointed arms, each with a three-fingered hand (the hands are not presently modeled in the visualization). The joint angle information is used to update the positional state of the 3D model, which may be accessed by any machine on the network.

In our system, components such as data sources, processing nodes, models, and user interfaces can be

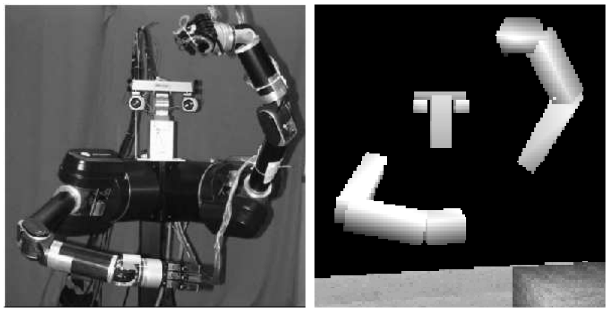

Figure 1: The virtual representation of the arm mirrors, in real time, the position of the real arm. The visualization can be performed on any computer with access to the network.

freely distributed to the various nodes of a network. For example, the workstation that provides the user interface and 3D rendering could be located right next to the robot, across campus, or (as we have done) a free-roaming wearable computer utilizing a wireless LAN. This allows for the components of the system to be located for the convenience of implementation. The particular software that ties these components together that we have developed is the Virtual Object System, a distributed object system designed to support (among other things) multi-user Virtual Reality (VR) applications.

3.1 Virtual Object System Goals

One of the most important aspects of the design of a complex system is the ability to reduce the system into appropriately-simple conceptual units. A very well-known example is the Unix philosophy of "everything is a file." The fundamental abstraction of the system is sequences of bytes which allow read/write operations. Basic files and directories are rather static and uninteresting; however fifos, devices, Unix domain sockets and /proc file system found on Linux and other Unices are all examples of ways in which file system objects can actually be quite dynamic and interactive, while still being like a normal file in various ways (read/write operations are the same, permission bits and ownership apply, etc.) Crucial to the success of this idea is the fact that everything exists in a single structured hierarchical name space: at the very least one can inspect or manipulate anything with simple command-line tools. By comparison, conventional Remote Procedure Call or Remote Method Invocation systems generally do not impose this structure on the objects under their control. On the other extreme, network files systems obviously impose the file system organization on their file objects, but generally lack the notion of sending arbitrary messages to those objects. It is the desire for this pervasive structure inspired by file systems that constitutes one design goal for the Virtual Object System (VOS).

This design goal is important because hierarchical structures are fundamental to modeling of three-dimensional environments and the objects contained therein. Recall the robot arm: the three-dimensional position of the wrist in the absolute coordinate frame is determined by the position of the joints that precede it in the kinematic chain. Since this computation is easily expressed by the composition of a series of homogeneous transformations, we make this computation implicit in the hierarchical structure of the object name space.

A second design goal is relative simplicity and generality of the actual network protocol. This is measured in terms of human readability and ease of writing simple, dedicated-purpose scripts to manipulate the objects. For comparison, HTTP passes this test extremely well. To retrieve a URL, all one has to do is send a "GET" method with the path in question to the server, and the contents of that URL "object" will be returned. Common Object Request Broker Architecture (CORBA)[8] is a very commonly used distributed architecture typically based on a binary protocol that requires at the very least that one generate stubs from an Interface Design Language (IDL) specification and link against an Object Request Broker (ORB). In contrast, our protocol, being XML-based, should make this sort of rapid development possible. Another XML protocol, XML-RPC [9], is trivial in the extreme which makes rapid development easy but doesn't specify much beyond the most basic message syntax. Simple Object Access Protocol[10], also an XML-based protocol, does not have any sort of implicit object structure either.

In the sections that follow, we illustrate the design of our virtual object system in the context of real-time visualization of the current configuration of the UMass Torso.

3.2 Virtual Object System Design

The virtual object system is a distributed object system which provides services for dispatching of messages between objects in a location-independent manner. Messages express all communication between objects of the system. Unlike most other systems, however, the naming scheme imposes an explicit notion of standard, hierarchical interconnections between objects. Specifically, a virtual object (henceforth referred to as a *Vobject*) is defined by the following properties:

- It is bound to a *site* (described below) and has a site-unique name;
- It has a list of types which define its supported interfaces as well as other attributes;
- It has a set of one or more parent Vobjects (the site is always a parent);
- It may have an ordered, associative list of child Vobjects; and
- It may send or receive messages to/from any other Vobject.

A site is a Vobject which serves as a bridge between the Vobject system and the underlying (generally socket-based) network layer (see figure 2). It provides a point (such as an open port) by which the Vobjects of the site are accessible. The site accepts messages on behalf of its hosted Vobjects and redirects those messages to their proper targets. Every Vobject must be bound to a site; this simply means that the Vobject is an immediate child of that site in addition to any other structural (parent/child) information that Vobject may have.

The type information associated with a Vobject describes the syntax and semantics of messages this Vobject may receive and generate. This is based on the notion in object-oriented programming of interfaces and method signatures. All Vobjects support a common interface which allows for the querying and changing of the information which is fundamental to all Vobjects listed above. For example, the main object types that make up the robotic arm consist of "object3d", "object3d.box" and "property." Vobjects of type "object3d" support generalized operations that can be performed on anything that can be manipulated in three dimension (translation, rotation, and scaling). Vobjects of type "object3d.box" represent a box or rectangular prism and (unlike the amorphous "object3d" type) suggest a way to visualize this object in three dimensions. Being a subclass of "object3d", "object3d.box" supports the operations of "object3d" as well. Finally the "property" type is an extremely common type specifying a simple interface to access some store of data. The usage of this data is determined by the contextual name given by their parent (explained below). The robot arm example uses properties to store the position (a 3-element vector) and orientation (a rotation matrix) of the object relative to the parent object; it also stores information texture which is mapped onto the surface of the 3D primitive.

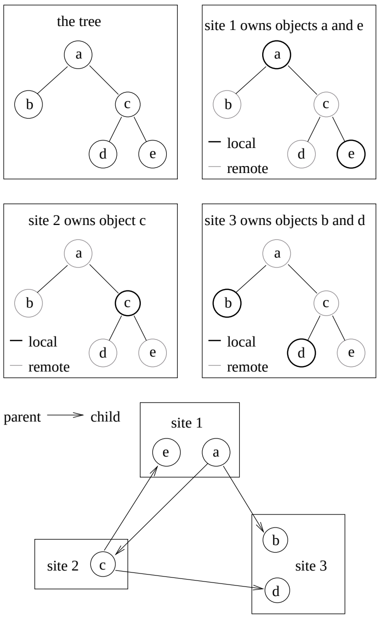

Figure 2: A graphical example of how the objects on various sites might interconnect. Objects which are distributed among several sites can be linked together to form a tree structure.

The parent set is the set of links to Vobjects of which this Vobject is a child. Note that the usage "parent" and "child," while derived from conventional terminology for tree structures, does not actually mean this structure follows the strict definition of a tree. It is actually a general directed graph. As such, the edges (links) between nodes (objects) are bidirectional, but asymmetric. Edges are described with the parent-child relation tuple. This tuple stores the parent object, the child Vobject, the child's position in the parent, and the child's contextual name in the parent.

The child Vobject list is an ordered, associative list of links to other Vobjects. Each child Vobject may be addressed by its position in the list or by contextual name. Each Vobject's child list is a separate context. The contextual name is simply an alias, similar to a symbolic link, which supplies some information about the purpose of the child in the context of the parent. For example, consider a Vobject which represents a 3D volume such as the forearm of the robotic arm. It has, as children, two Vobjects which themselves contain information about properties such as position and orientation.

Slash-delimited paths which should be intuitive from file name and URL syntax is used to address Vobjects, such as the path going all the way from the world object, along the kinematic chain, to the robot arm's hand: "/world/robotarm/shoulder/elbow/wrist/hand." This makes it very easy to browse the object tree in the same fashion one might browse a file system.

```
<message length="222"
    to="vop://zarya:4234/wrist/orientation"
    from="vop://interreality.org:4233"
    method="property-replace"
    nonce="262679938">
  <data>0.286476 -0.347896 -0.517211</data>
  <datatype>x-3tuple-float</datatype>
</message>
```


Figure 3: An example message changing the orientation of the robot's wrist. The *method* designates the operation to be performed on the designated object; the *nonce* is a unique message identifier.

3.3 Network Protocol

When Vobjects exist in the same process — that is, a thread of control and the associated memory within an operating system of a single machine — method invocations are made directly with no intermediate packaging of parameters of the method call into a message or network packet. When a Vobject exists over the network, however, the method call will be made through a piece of code called a stub. A stub simply converts the calling parameters of a method into the fields of a message. Unlike other RPC systems there is presently no interface design language, so stubs must be coded by hand. However, an advantage of this is that it becomes much easier to include code which may streamline message processing or provide caching logic. If necessary, the stub will send a message to the remote Vobject and wait for a reply. Because these stubs separate interface from implementation, it also becomes useful for code reuse, for some stubs can be written in terms of other Vobject types, or provide additional logic for that Vobject. This has proven to be especially useful in implementing A3DL (discussed below).

The underlying protocol used between sites is XML-based, sent over TCP/IP sockets. Messages are parsed using a flex and bison based parser. Figure 3 is an example of changing the value of a property. In this case, the orientation of the robot's wrist is being changed; the contents of the property are the Euler angles of the transform with respect to the enclosing coordinate frame.

4 A Wearable Computer Interface for 3D Environments

4.1 Rendering of the 3D Environments

Using this Vobject system as a basis, we have developed a prototype API for describing 3D virtual objects. Because Vobjects may be arbitrarily distributed about the network, the world is constructed of the union of many Vobjects collected from various nodes of the network. In designing distributed systems, one important principle is that of placing specialized processing as close as possible to the data source. This is generally convenient for the implementation or specialization of the Vobjects, as well as being more robust than the alternative which would force the raw data to be delivered to and processed entirely by a centralized rendering system.

For our arm example, the set of Vobjects which represent the arm consist of a homogeneous transform for each joint and the attached geometry representing the upper arm, forearm and hand. These Vobjects exist on a workstation which is attached to the robot controller. Other aspects of the 3D environment (such as the enclosing walls of the room) exist on another, separate server. A third (wearable) computer performs the actual rendering and presents the user interface. Because the Vobjects modeling the robot arm are hosted independently of the enclosing world, they may join and leave that world (and hence appear/disappear from the visualization) without disturbing any other elements of that world. This also allows the object to appear in multiple worlds.

To facilitate the development of both the actual rendering software as well as clients which make use of 3D Vobjects (independent of rendering), we have developed an Abstract 3D Layer (A3DL). This is a simple interface for representing and manipulating 3D Vobjects. Using this API, the robot presents the geometric model in terms of volumetric primitives (to represent the physical parts of the arm) and homogeneous transforms (to represent their spatial relationships). The robot controller itself knows nothing about the actual rendering process – it simply exports a geometric representation that captures the arm's spatial state. The implementation of A3DL for a specific 3D engine becomes primarily a process of filling in the necessary back-end code to represent the Vobjects in the engine that is being used. For our implementation, we use the Crystal Space 3D engine[11].

4.2 The Wearable Computer Platform

One design constraint has been the goal of implementing a fully mobile system. One major factor in the development of many virtual reality systems is in the specialized hardware required. We elected to deploy our system on the Xybernaut wearable computer, which is a commercial product. The Xybernaut is a complete, self-contained Intel-based system packaged into a belt or vest form-factor. The CPU is a 200MHz mobile Pentium with 192 MB RAM and a 4 GB hard disk. The operating system is Red Hat Linux. The HMD is based on a 640x480 color LCD; a concave mirror is positioned to reflect the display image toward one eye of the wearer. The field of vision covered is approximately 30°. Although in practice this display leaves much to be desired in terms of contrast and focus, it is very easy to switch the mirror for a semi-silvered (or semi-transparent) mirror. This allows one to see through the mirror as well as see the reflection, which makes possible the idea of "augmented reality" (overlaying a virtual world onto the real one, appearing to match up with real features of the real world). Finally, the wearable operates on a Lithium-ion battery and uses 802.11b wireless ethernet. The battery presently lasts about 2.5 hours with moderate use. Despite being a relatively slow 200MHz CPU and having no extra hardware support for 3D rendering, we are able to generate 10 frames per second entirely with software-based rendering.

One of the basic features of a virtual reality system should be the ability to walk about the virtual environment freely and for the rendering system to immediately respond to what the user looks at — that is, change the viewing direction based on what direction the user's head is pointing. For tracking head orientation, we use an Intersense IS-300 gyroscopic tracker [6]. This device uses 3-axis gyroscopes measuring rotational acceleration, along with geomagnetic and gravity sensors to measure compass heading and pitch/roll of the head. The device is attached to the headset of the wearable computer. Filtered head orientation information is delivered via a serial port at a rate of up to 50 Hz.

Depending upon the application and the user context, the Cartesian position of the virtual viewpoint is translated using one of several approaches. First, the user may physically drive the forward/backward movement of the viewpoint by pressing buttons on a keyboard. Second, the position of the user may be sensed using either services provided by the UMass smart room [1, 2] or a GPS receiver. The smart room has been equipped with panoramic and pan-tilt-zoom cameras which track the movement of people and robots within the room. Unlike many localization systems which rely on magnetic or sonar sensors, this camera-based system is entirely passive and can be very quickly set up and calibrated. This tracking system provides an estimate of 2D position within the room at a rate of 9 Hz, with an error standard deviation of 30 cm.

5 Example Applications

We have constructed three examples to demonstrate the utility of our mobile virtual reality system. The first example, using the robotic torso, has been discussed previously. However, a note on the general ease of implementation: the code specific to the robot representation consists of a total of less than 150 lines of C++. Nearly all of this code is dedicated toward the creation and initialization of the arm-specific transformations and 3D volumetric Vobjects. The program then executes forever in a loop, polling joint angles every 150 milliseconds. Because the polling is done once every 150 ms, the worst-case lag from real movement to response on the screen is on the order of 250 ms at the 10 FPS screen rate.

The second example application augments the user's field of view with information about subjects moving within a nearby room. In this case, the user with the wearable computer is positioned in a known location outside of the smart room. A 3D model of the environment is made available to the wearable computer by the *world server* depicted in figure 4. The *world* Vobject contains a variety of 3D Vobjects, including doors, desks, and an avatar that corresponds to the tracked subject. Each Vobject in the model is represented using a set of properties, including position and orientation. In the case of desks and doors, these properties are stored locally to the server. However, as in the case of the avatar position in this example, the locally stored Vobject may be replaced with a reference to a Vobject that is made available by other servers. In our example, this position information is made available by the smart room tracking system, but could equally be provided by any other tracking system (including a GPS-based system).

Figure 5A shows an example view from the wearable computer. Because the virtual and real worlds are approximately aligned, the user has the impression of seeing the subject through a transparent wall. Although the subject is currently rendered as a monolith, it is possible to substitute a more interesting, subject-specific avatar (represented as general mesh

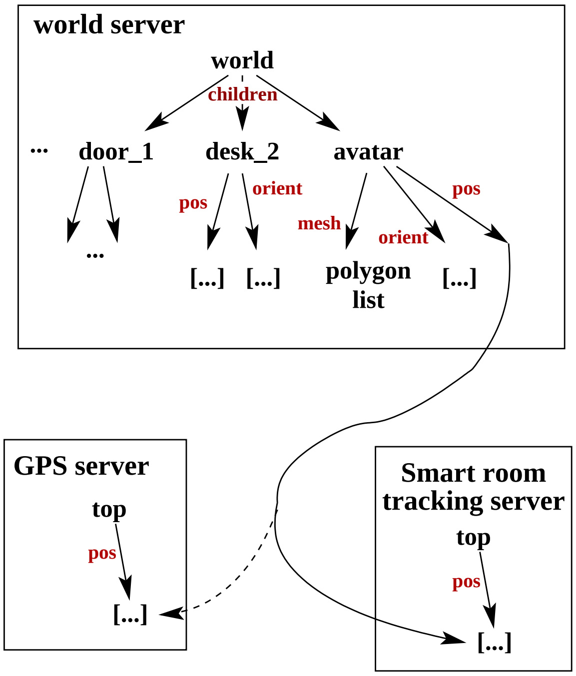

Figure 4: Vobjects hierarchy for the representation of the virtual environment in the X-ray vision example. Vobject properties may be stored and manipulated by the local server or may reference objects exported by remote servers.

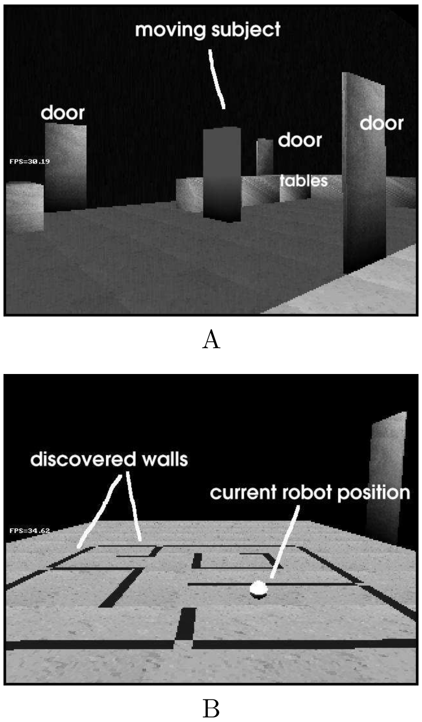

Figure 5: (A) The user view of the X-Ray vision demonstration, and (B) tracking the movements of a robot as it maps a set of rooms.

objects).

The third experiment involves the monitoring of a set of robots during a mapping and search task. In the current instantiation, a mobile robot is placed within a maze and is asked to autonomously map the space (see figure 5B). The robot communicates via a radio link its own estimated position, as well as the position of any obstacles that it encounters to to a base station. The Vobject server for this robot combines the robot position tracking with the robot's maze-discovery information to create a three-dimensional geometric model. This model is updated in real time with the robot's current position and the map that it has thus far discovered. As with the first experiment, the actual rendering is done on the wearable computer based on the dynamically-changing geometric description given by the base station. Because the user's head position is tracked by the sensors in the smart room, the user is able to "walk around" the geometric model that has been created.

6 Conclusions and Future Work

Visualization of robot state through augmented and virtual reality systems will become one of the key bases for human interaction with large numbers of robots and sensors. In this paper, we have presented some initial steps toward developing a mobile virtual reality system for visualization of state information. Our Virtual Object System addresses the issues of presenting a uniform interface for network-available data with a particular focus on the distributed representation of hierarchical, three-dimensional models. These models may be updated as a function of live data sources and rendered in real time for presentation to remote users. Furthermore, the system has been designed with limited network and computational resources in mind, allowing for the presentation of the dynamic virtual space in a virtual reality format to a user equipped with a fully-mobile, wearable computer.

On the technical side, our immediate focus is on the development of more complex 3D models for the representation of spaces involving multiple rooms and large numbers of objects. We are also designing avatars which will be capable of expressing subject state information, including subject identity and current activity.

We are also examining issues involving the interaction of the user with the available sensory resources.

In a search and rescue task, the virtual/augmented reality interface can make survivor and hazard information available to the field user on a wide range of spatial scales – from summary views of an entire building down to the what has been discovered around the corner from the user. In addition, the field user should be able to make high level requests of the robot/sensor network using intuitive interfaces (such as pointing toward a door), with the necessary resources being automatically marshaled to satisfy the requests.

Finally, we are examining the presentation of both static and live visual imagery within the virtual environments. One important issue to be addressed is that of conveying to the user a sense of the spatial relationships between the data taken by the disparate sensors. This must be accomplished in an informative and timely manner and may present a trade-off with presentation quality. The virtual environment allows the user to navigate amongst the data collected by these sensors and to perform queries from individual sensors for high quality images or even live image streams. Through this interface, we hope to leverage the user's own ability to construct internal cognitive maps of three dimensional spaces in order to "fill in" details not available from the sensors.

7 Acknowledgments

Preparation of this manuscript was supported in part by NSF #EIA 9703217, DARPA/ITO #DABT63-99-1-0022 (SDR) and DARPA/ITO #DABT63-99-1-0004 (MARS). The authors wish to thank T. Reed Hedges, Joshua Gay, James Davis, Michael Piantedosi, Jack Wileden, and Amy McGovern for their valuable comments during the development of this work and the writing of this manuscript. The authors wish to also thank Xybernaut Corporation for their support of this work.

The most recent version of the Virtual Object System is available for download from:

www.interreality.org

References

[1] D. Karuppiah, P. Deegan, E. Araujo, Y. Yang, G. Holness, Z. Zhu, B. Lerner, R. Grupen, and E. M. Riseman, "Software mode changes for continuous motion tracking," in *Proceedings of the International Workshop on Self Adaptive Software*, 2000.

[2] G. Holness, D. Karuppiah, S. Uppala, S. C. Ravela, and R. A Grupen, "Service paradigm for reconfigurable agents," in *Proceedings of the 2nd Workshop on Infrastructure for Agents, MAS, and Scalable MAS*, May 2001.

[3] S. Feiner, B. MacIntyre, T. Höllerer, and A. Webster, "A touring machine: Prototyping 3d mobile augmented reality systems for exploring the urban environment," in *Proceedings of the First International Symposium on Wearable Computers*, 1997.

[4] T. Starner, S. Mann, B. Rhodes, J. Levine, J. Healey, D. Kirsch, R. W. Picard, and A. Pentland, "Augmented reality through wearable computing," *Presence*, vol. 6, no. 4, Fall 1997.

[5] T. Jebara, C. Eyster, J. Weaver, T. Starner, and A. Pentland, "Stochasticks: Augmenting the billiards experience with probabilistic vision and wearable computers," in *Proceedings of the IEEE International Symposium on Wearable Computers*, October 1997.

[6] E. Foxlin and M. Harrington, "WearTrack: A self-referenced head and hand tracker for wearable computers and portable VR," in *Proceedings of the Fourth International Symposium on Wearable Computers*, 2000.

[7] S. Gibson and A. Murta, "Interactive rendering with real-world illumination," in *Proceedings of the 11th Eurographics Workshop on Rendering*, June 2000.

[8] "Common object request broker architecture (CORBA) 2.5 specification," Tech. Rep., Object Management Group, 2001.

[9] E. Kidd, "XML-RPC HowTo," Tech. Rep., 2001, xmlrpc-c.sourceforge.net/xmlrpc-howto/xmlrpc-howto.html.

[10] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer, "Simple object access protocol (SOAP) 1.1," Tech. Rep. 08, W3C, May 2000, www.w3.org/TR/SOAP.

[11] J. Tyberghein, A. Zabolotony, E. Sunshine, T. Hieber, S. Galbraith, M. Geisse M. Voase an S. Humphreys, A. Pfaffe, M. Ewert, R. Bate, G Haussmann, and P. Wyett, "Crystal space manual," Tech. Rep. 19.dev, 2001, crystal.sourceforge.net/docs/online/manual/.