Middleware Mechanisms for Interaction Interoperability in Collaborative Virtual Environments

Mario Ciampi*†, Luigi Gallo*†, Antonio Coronato*, and Giuseppe De Pietro*
* ICAR-CNR
Via Pietro Castellino 111, 80131 Napoli, Italy
Email:{ciampi.m, gallo.l, coronato.a, depietro.g}@na.icar.cnr.it
† University of Naples "Parthenope"
Centro Direzionale Isola C4, 80143 Napoli, Italy
Email:{mario.ciampi, luigi.gallo}@uniparthenope.it

Abstract—**In this paper, we discuss interaction interoperability in Collaborative Virtual Environments (CVE), intended to mean the ability of two or more users to cooperate despite the heterogeneity of their interfaces. To allow such interoperability, rather than focusing on the de-coupling of input devices from interaction techniques and from interaction tasks, we suggest integrating interactive systems at higher level through an interface standardization. To achieve this aim, we propose: i) an architectural model able to handle differences in input devices and interaction tasks; ii) an agent-based middleware that provides basic components to integrate heterogeneous user interfaces. We also present a prototype of an agent-based middleware able to support developers in the interconnection of monolithic applications and we introduce tools and languages we have used to formalize the interaction tasks considered in the case study.**

I. Introduction and Background

In recent years, Virtual Reality (VR) has become increasingly popular, not only as a visualization technology but also as a communication interface based on interactive and immersive visualization. VR is commonly defined as "a collection of technologies that allow people to interact efficiently with 3D computerized databases in real time using their natural senses and skills" [1]. In this definition, the stress is put on the VR interaction paradigm in which the user is an active participant: the immersion of the user in a virtual world allows new ways of interaction and so makes it possible to design human-centric interfaces that better focus on his skills and capabilities.

Despite the wide availability of input devices and interaction techniques, usually new VR applications exhibit ad hoc post-WIMP interaction techniques and devices. This is not a surprise since it is a shared vision that diversity in human-computer interfaces is a resource rather than a problem. As Benford et al. state in [2], it would be impossible to develop a generic interface for every user. In order to be useful, every domain requires a custom design. Bowman et al. further refine this concept in [3]. They state that the key word in the development of usable VR user interfaces is *specificity*. In greater detail, an Interaction Technique (IT), in other words the way of using a physical device to perform an interaction task, should be designed by considering specifically the group of users (and therefore their skills), the tasks they have to perform, the particular application and the input device used.

Therefore, in Collaborative Virtual Environments (CVE), and even more in Massively Multiuser Virtual Environments (MMVE), the heterogeneity of input devices and interaction techniques is a problem to face. On the contrary, in reality, in most collaborative environments users are forced to use the same input devices and techniques, nearly always WIMP-based, to interact. For example we can consider the field of telemedicine. In daily clinical practice, each clinician has her/his own skills and uses her/his preferred tools to inspect medical data. So, developing different interfaces for different medical figures can enhance the usability of the interaction with medical datasets. Nonetheless, collaborative applications are insufficient in handling the heterogeneity of interaction tools. Group-Slicer [4] is a collaborative extension of the well-known 3D Slicer, a tool for both surgical planning and medical operations. The precondition of the interaction is the use of the same input devices and interaction techniques. The same in [5], in which the authors describe a middleware environment able to handle heterogeneity of visualization but not of interaction. In [6] the authors introduce COVE, an extensible framework for collaborative visualization which handles possible conflicts generated by multiple plug-ins that read from the same input device. The main advantage of this framework is that it allows plug-in developers to design ITs by separating their implementation from input devices.

This paper is focused on *interoperability*, "the ability of two or more software components to cooperate despite differences in language, interface, and execution platform" [7]. In greater detail, we discuss *interaction interoperability*, that is the ability of two or more users to cooperate despite differences in input devices and interaction techniques.

To the best of our knowledge, this question has recently been raised by Ahmed et al. in [8]. In this paper, the authors propose a framework to support interaction interoperability, the main benefit of which is the ability to use ontologies to define interaction tasks and techniques without specifying how these tasks should be accomplished. By using ontologies, the framework binds tasks to the user's preferred interaction techniques and then to available input devices. In other works by the same authors, the framework capabilities are further described (see [9] and [10]). Interoperability is granted by using a standardized taxonomy built with the Ontology Web Language (OWL). The framework they propose should also be able to choose the best input device for a particular user by analyzing her/his preferences in the interaction. The idea of decoupling interaction techniques from interaction tasks and input devices is also described in [3], in which the authors explore the integration of different technologies to support co-located collaborations.

From the point of view of an interface programmer, decoupling interaction techniques from input devices may not be worthwhile. As Poupyrev et al. outlined in [11], similar interaction techniques vary depending on the particular implementation. Studies of the particular implementation of a technique may not be easily applied to other implementations of the same technique. Therefore, decoupling interaction techniques from input devices and mixing them according to user patterns could reduce the usability of the interface.

In light of these considerations, assuming that a formal description of interaction tasks is still necessary, we propose to let interaction designers plan ITs coupling them strictly to the particular input devices used. In the approach, we propose that different ITs and input devices are handled in a common framework in the same way as black boxes. We suggest using an agent-based middleware to provide interaction interoperability in CVEs. This middleware is built upon an architectural model in which the typical system layers have been extended to provide interaction standardization between different systems. The aim is to allow both interaction designers to plan ITs, coupling them with the corresponding input devices, and users to collaborate by using the combination of ITs and devices they prefer.

The organization of the paper is as follows. First, in section II, we describe the architectural model we suggest. Then, in section III, we introduce the agent-based middleware and its basic components. Finally, in section IV, we describe a proof-of-concept case study, in which two users collaborate in a virtual environment by using different interaction techniques and input devices.

II. Architectural Model

CVEs are complex systems in that they involve the crucial concept of *sharing*. Users can share knowledge of each other's current activities, environments and actions. Most of the issues to deal with in typical CVEs are beyond the scope of this paper. We focus on the question of interaction interoperability, so sharing in this context refers to interaction tasks.

Since every user has her/his preferred formalism and tools, we speculate that in a CVE each user has to be free to use her/his preferred interaction techniques and devices. To allow such a heterogeneity in the execution of interaction tasks, interaction formalization plays a primary role. Possible interactions between heterogeneous representations have to be formalized in a common, shared way.

As Wegner states in [7], there are two main mechanisms for interoperation in this field: *interface standardization*, which means that each users interface is mapped to a common representation; and *interface bridging*, which means that there is a two-way map between user interfaces. Interface standardization makes explicit the common properties of interfaces, thereby reducing the mapping task, and separates communication models of user interfaces. On the contrary, interface bridging is more flexible, since it can be tailored to the needs of specific user interfaces.

The approach we suggest is to let the user who is in charge of creating the CVE to implicitly formalize, by exposing her/his user interface, the common language that other user interfaces have to speak. Together with the language, the user also has to specify the peculiar interaction rules he/she requests.

To better introduce this approach, it is worth considering the typical logical layers of a collaborative system, as described by Osais et al. in [12]. In such a system, we can distinguish three layers: application, control and server (see figure 1). The *application layer* is in charge of intercepting events, providing awareness to users, posting remote events to a manager and interpreting as local the remote events received. The *server layer* contains the session server, which is in charge of distributing messages to all users. The *control layer* is the cornerstone of any collaborative system, since it has to replicate state changes to maintain consistency among users, to exchange control information and to replicate signaling messages.

In a control layer, we need several different figures. An *event manager* performs the required conversion of messages in a common format. A *session manager* handles sessions, maintaining a list of active participants. A *floor manager* handles floors, that are temporary permissions to access and manipulate resources, so regulating collaboration. A *communication manager* provides services necessary to transport messages.

In the architectural model we suggest, there is a new figure, the *interaction manager* (see figure 1). Its main responsibility is to handle transitions between interaction tasks. The interaction manager is interposed between the event manager and the communication manager, since events corresponding to interactions not allowed in a particular context are not further propagated in the system.

This component has to be able to deal with the complexity of post-WIMP interfaces. It works thanks to a formal model of the interaction rules that specifies how interaction tasks are connected and how and when floors are granted. Interaction rules change significantly with the cooperation model. By following the cooperation model proposed by Margery et al. [13], there are three levels of cooperation: *level 1* is a basic cooperation level, in which users simply perceive each other in

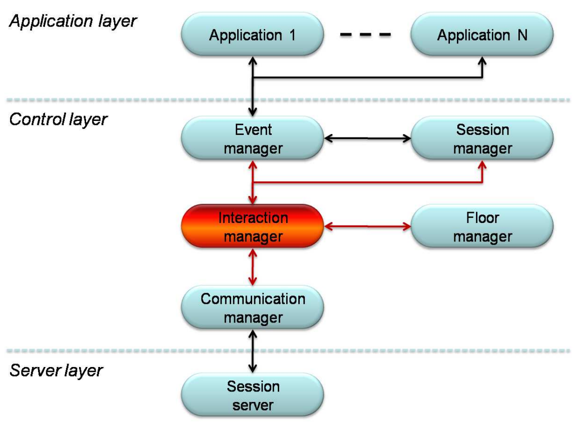


Fig. 1: Architectural model

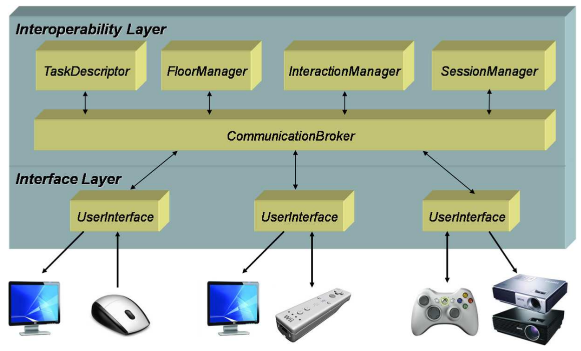


Fig. 2: Middleware architecture

the virtual world; *level 2* means that each user can change the scene individually; finally there is *level 3* in which users can act on objects in independent ways (*3.1*) or in a co-dependent way (*3.2*). Obviously different cooperation levels correspond to different interaction rules.

A collaborative system which exploits interaction interoperability should provide mechanisms to express such fine differences in interaction models. The formalism we suggest using is Interactive Cooperative Objects (ICO) [14], an object Petri nets-based formalism suitable to model real-time interfaces. In section IV, an example is reported of the adoption of such formalism to model interaction rules in a real case study.

III. Agent-based Middleware

This section presents the key elements of an agent-based middleware infrastructure devised to overcome interoperability issues for CVEs. The middleware has been implemented by an agent infrastructure, since it facilitates the communication between distributed components and provides, at the same time, a certain degree of autonomy [15]. Such infrastructures, named Multi-Agent Systems (MAS), focus on systems in which intelligent agents interact with each other to cooperate or to solve complex problems. Thus, they are well suited to Collaborative Virtual Environments, where collaboration and problem solving are significant requirements.

The middleware we propose enables users participating in virtual collaborative sessions to use different input devices and interaction techniques. In greater detail, the middleware allows users to interact with each other by i) visualizing simultaneously the same multimedia objects and ii) replicating on every participant's node the execution of all interaction tasks performed by other participants.

A. Middleware Functionalities

In general, the middleware infrastructure has to aim at i) sharing a formal description of interaction tasks and a number of interaction rules, ii) handling floor control, by authorizing or not users to perform interaction tasks on multimedia objects, and iii) dispatching event notifications to all participants in order to synchronize the execution of certain tasks.

It is worth noting that formal descriptions of interaction tasks, along with interaction rules, have to be produced by the user who creates the collaborative session. Every participant willing to join the session has to check if he/she has an implementation of those tasks at her/his disposal.

Formal descriptions and interaction rules have to be specified in a shared formal language. For each interaction task, users who want to cooperate have to indicate: i) the typology and number of input and output parameters and ii) the preconditions to be satisfied in order to allow the execution of the task. It is irrelevant *how* designers implement interaction tasks; it is sufficient that they accomplish the specified requirements.

B. Middleware Architecture

Middleware architecture consists of a set of components organized in two logical layers, as shown in figure 2. The *Interface layer* handles user interactions and implements the interaction tasks requested to participate at a session. The *Interoperability layer* is the cornerstone of the middleware, since it is in charge of replicating state changes to maintain consistent views.

The components of middleware architecture are described below:

- *UserInterface*: this component acts as an adaptor and aims at handling interaction events generated by a particular input device while performing interaction tasks. For this reason, each participant has her/his own UserInterface, within which ad hoc developed interaction techniques can be enclosed.
- *CommunicationBroker*: this component is in charge of dispatching messages received from component publishers to component consumers in an asynchronous way according to the publish/subscribe paradigm. The CommunicationBroker plays a central role in the middleware, although its architecture can be distributed.
- *TaskDescriptor*: this component is in charge of archiving and sharing formal descriptions of interaction tasks. The interfaces of UserInterface components have to satisfy such formal descriptions.

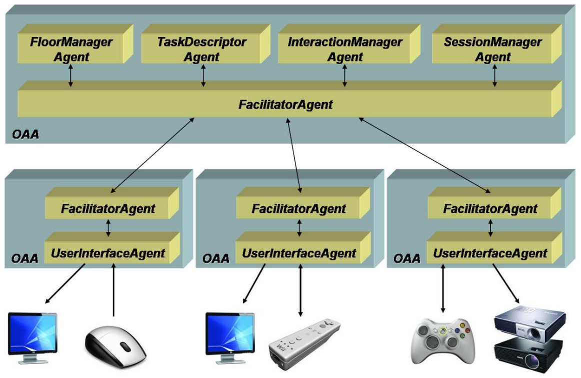


Fig. 3: Middleware implementation with OAA

- *FloorManager*: this component handles the concurrency control for shared multimedia objects and stores the interaction rules. Each UserInterface questions the FloorManager to perform an interaction task. The FloorManager grants or refuses the authorization according to the interaction rules shared between participants.
- *InteractionManager*: the InteractionManager ensures that each participant has the same visual representation by requiring all users, via the CommunicationBroker, to execute an interaction task launched by a user.
- *SessionManager*: this component allows users to create, join and destroy collaborative sessions by handling a list of participants in a virtual collaborative session. Every participant has to register with such a component to access the system.

IV. A Case Study

A proof-of-concept prototype of middleware has been developed to evaluate the interoperability facilities provided by the agent-based middleware architecture previously described.

As shown in figure 3, current implementation relies on the Open Agent Architecture (OAA), a framework for constructing Multi-Agent Systems developed at the Artificial Intelligence Centre of SRI International. OAA is structured so as to minimize the effort involved in creating new agents and ”wrapping” legacy applications [16]. Moreover, OAA facilitates the development of flexible and dynamic agent communities using a wide variety of programming languages. Members of agent communities cooperate with each other in order to perform computation, to retrieve information and to carry out user interaction tasks [17].

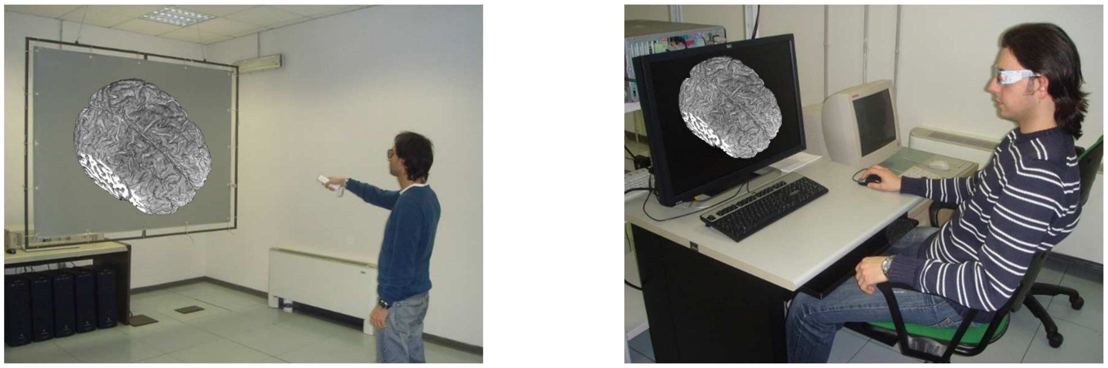
The OAA-based middleware we have developed enables two users to cooperate in a semi-immersive virtual environment. In greater detail, the application allows them to visualize and interact with 3D anatomical parts reconstructed from CT and MRI images.

Users can perform two distinct interaction tasks: rotation of a 3D object, which has to be performed in an exclusive way; and pointing at 3D objects, that is moving a 3D cursor in the scene. The two users considered in this scenario use different interaction techniques and input devices: the user in figure 4a interacts with a Wiimote, the main controller of the Nintendo Wii console [18], whereas the user in figure 4b interacts via a common mouse.

The interaction tasks executed by the two users are implemented differently: the rotation technique of the first user (Wiimote-based interaction) makes use of quaternions as a mathematical notation for representing orientations of objects in three dimensions, whereas the rotation technique of the second user (mouse-based interaction) makes use of three Euler angles. In this scenario, since the first user is assumed to create the collaborative session, quaternions are used as the shared notation. Thus, the adaptor of the second user has to implement the conversion from Euler angles to quaternions.

All components have been implemented as agents, as shown in figure 3. It is worth noting that an agent has to declare its capabilities, named *solvables*, in order to join a community of agents. Such solvables establish a high-level interface to the agent, used by a facilitator in communicating with the agent and, most importantly, in delegating service requests to the agent. Such a model is called the *delegated computing model* in OAA. Two major types of solvables are distinguished: procedure solvables, which describe how agents perform actions, and data solvables, which provide access to a collection of data. In the following, a description of every agent constituent of the middleware is provided.

- *UserInterfaceAgent*: this agent analyzes input interactions and manages the presentation of multimedia output to the user. Each UserInterfaceAgent implements particular ad hoc developed interaction techniques to perform interaction tasks. These tasks, together with the languages they are able to speak, have to be declared as procedure solvables to the FacilitatorAgent.
- *FacilitatorAgent*: this agent acts as the CommunicationBroker component. It coordinates agent communications, by maintaining lists of what UserInterfaceAgents can do, that is their solvables. Each UserInterfaceAgent has its own FacilitatorAgent, that *delegates* service requests to appropriate agents. The FacilitatorAgent implements the publish/subscribe paradigm by means of the OAA’s delegated computing model. Each FacilitatorAgent is linked to all the others by means of a root FacilitatorAgent, creating thus a hierarchical configuration. Such a hierarchy has been arranged on two layers, even if it may be organized on more than two.
- *TaskDescriptorAgent*: the TaskDescriptorAgent declares to the FacilitatorAgent a data solvable, in order to archive and share task interfaces written in a formal language. The UserInterfaceAgent that creates the collaborative session sends to the TaskDescriptorAgent the interfaces of interaction tasks by means of the OAA’s Interagent Communication Language (ICL), an interface and communication language designed as an extension of Prolog. In our scenario, the first user specifies the interfaces of the rotation and pointing tasks.
- *FloorManagerAgent*: the FloorManagerAgent is a meta-agent that manages the floor control of the objects of the


(a) Case I - Wiimote-based interaction

(b) Case II - Mouse-based interaction

Fig. 4: Case study: two users interact remotely by using a Wiimote and a mouse respectively

visual representation. The FloorManagerAgent resolves conflicts on the basis of interaction rules. This agent declares a data solvable too, for the purpose of storing such interaction rules. These have to be specified by UserInterfaceAgents via ICO. For simplicity, in our proof-of-concept prototype the ICO-based interaction rules of the rotation and pointing tasks have been implemented directly as C++ code in the FloorManagerAgent. In each case, several Petri nets interpreters are available (see [19] and [20]) and can be used to parse UserInterfaceAgent requests.

- *InteractionManagerAgent*: the InteractionManagerAgent requires every UserInterfaceAgent to execute certain tasks in order to synchronize the visualization among users. This is accomplished by extending the ICO formalism to the agents. Specifically, the notion of object has to be replaced with the concept of agent: when a token passes through a transition, the action to perform is the invocation to an agent rather than to an object. This can be accomplished via the FacilitatorAgent by means of the OAA delegated computing model.
- *SessionManagerAgent*: this agent acts as the SessionManager component. It is invoked by other agents to share collaborative sessions or to verify if a participant has the credentials to request the execution of certain tasks.

The object Petri nets-based ICO formalism deserves a further description. Petri nets are a mathematical representation used to model discrete distributed systems, characterized by *places*, which are symbolizing states, *transitions*, which are representing actions, and *arcs*, which link places and transitions. The state of the system is defined by a distribution of *tokens*, which flow through places along the arcs every time transitions are enabled. Object Petri nets attribute a type to each place on the net, allowing thus the use of classes as types and object references as values. Instead, transitions are associated to method invocations. ICO extends object Petri nets by adding the notion of interaction events, which enable the transitions.

By using the ICO formalism, the FloorManagerAgent can deduce if an exclusive task has already been executed by a user. Figure 5 reports the ICO-based formalism of interaction tasks considered in the case study scenario. The places *Watch* and *Pointing* are initialized with N tokens, where N is the number of participants (N=2 in figure 5), so a maximum of two users can move their 3D cursor at the same time. The place *ObjectStill* is initialized with only one token. If a user $< u >$ asks the FloorManagerAgent to rotate an object, this enables him/her to perform the task only if the token is in the place *ObjectStill*. In such a case, one token of the place *Watch* and the token of the place *ObjectStill* fall into the place *Rotation*. Until $< u >$ rotates the object $< o >$, any other user can rotate the object. When $< u >$ stops rotating, the two tokens return to their initial states and the rotation task becomes executable. This configuration ensures the exclusivity of the rotation task. On the other hand, the pointing task is not exclusive. Hence, the initialization of the place *Pointer* with N tokens, specifying that N pointers $< p >$ are available, guarantees that every user can perform the task. It is worth mentioning that the synchronization between two places (such as *ObjectStill* and *Rotation*) cannot be modeled by a finite state machine.

It is also important that, in the particular case study we have

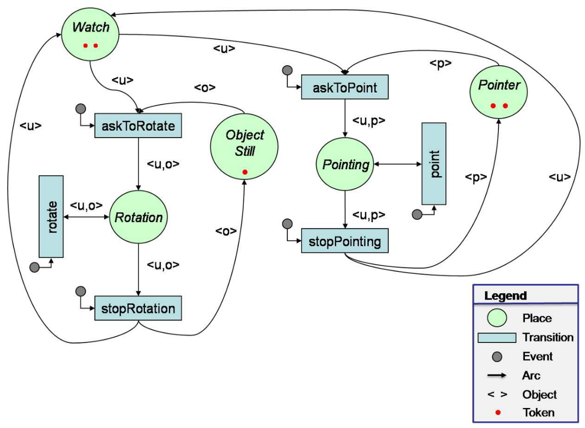


Fig. 5: ICO representation of the interaction rules

considered, two users exploit different input devices but also different interaction techniques. If performed with the mouse, the pointing task is a simple 1:1 mapping between the mouse position and cursor position in the scene. When the Wiimote is used as input device, pointing information is derived by tracking an IR emitter in space. To manage hand jitters, the Wiimote movements are processed by a PRISM-like filter [21]. The control-display ratio is recomputed immediately according to the velocity of the input device movements, so that a little cursor shift corresponds to a slow Wiimote movement and a wide cursor shift corresponds to a quick Wiimote movement. Therefore, there is no 1:1 mapping between the device and cursor positions like in the mouse based interaction. Also the rotation task is differently implemented. The Wiimote-based rotation is implemented by mapping 1:1 the orientation of the device to the orientation of the 3D object (as reported in [22]), so rotations can be performed with 3 degrees of freedom.

To grant interaction interoperability, all that is needed is that UserInterfaceAgents are able to speak the same language and execute the same task. In the pointing example, a UserInterfaceAgent communicates the spatial position of the cursor in the three dimensional Cartesian coordinate system, since this is the shared language for the pointing task. Other UserInterfaceAgents receive the pointing event information, understand it since it is in a language they speak, and know how to execute it since they have declared the pointing functionality as one of their procedure solvables.

V. Conclusion and Future Work

In this paper, we have presented an architectural model together with an agent-based middleware to enhance collaborative virtual environments with interaction interoperability facilities. The heterogeneity of user interfaces is handled through an interface standardization performed at a middleware level, so each user is able to choose her/his favorite coupling of input device and interaction technique to perform an interaction task. The paper also discusses suitable formalisms to describe effectively complex interaction tasks and floor management.

Future work will aim at building tools to model easily the behavior of interaction techniques and devices in order to simplify the interface standardization process. We also plan to evaluate server-based vs. peer-to-peer platforms to better identify points of failure and bandwidth bottlenecks in the proposed middleware solution.

Acknowledgment

This work has been partly supported by the project "MIPHA - A Middleware for Pervasive Healthcare Applications" within the "POR - Misura 3.17" funded by the "Regione Campania".

References

[1] G. Riva, "Applications of virtual environments in medicine," *Methods of information in medicine*, vol. 42, no. 5, pp. 524–534, 2003.

[2] S. Benford, D. Snowdon, A. Colebourne, J. O'Brien, and T. Rodden, "Informing the design of collaborative virtual environments," in *GROUP '97: Proceedings of the international ACM SIGGROUP conference on Supporting group work*. New York, NY, USA: ACM, 1997, pp. 71–80.

[3] D. A. Bowman, J. Chen, C. A. Wingrave, J. F. Lucas, A. Ray, N. F. Polys, Q. Li, Y. Haciahmetoglu, J.-S. Kim, S. Kim, R. Boehringer, and T. Ni, "New directions in 3D user interfaces," *The International Journal of Virtual Reality*, vol. 5, no. 2, pp. 3–14, 2006.

[4] F. Simmross-Wattenberg, N. Carranza-Herrezuelo, C. Palacios-Camarero, J. P. C. de la Higuera, M. Ángel Martín-Fernández, S. Aja-Fernández, J. Ruiz-Alzola, C.-F. Westin, and C. Alberola-López, "Group-slicer: a collaborative extension of the 3D-slicer," *Journal of Biomedical Informatics*, vol. 38, no. 6, pp. 431–442, 2005.

[5] P. Dev and W. L. Heinrichs, "Learning medicine through collaboration and action: collaborative, experiential, networked learning environments," *Virtual Reality*, vol. 12, no. 4, pp. 215–234, 2008.

[6] S.-H. Ryu, H.-J. Kim, J.-S. Park, Y. won Kwon, and C.-S. Jeong, "Collaborative object-oriented visualization environment," *Multimedia Tools and Applications*, vol. 32, no. 2, pp. 209–234, February 2007.

[7] P. Wegner, "Interactive software technology," in *The Computer Science and Engineering Handbook*, 1997, pp. 2440–2463.

[8] H. M. Ahmed, D. Gracanin, and A. Abdel-Hamid, "Poster: A framework for interaction interoperability in virtual environments," in *3DUI '08: IEEE Symposium on 3D User Interfaces*, 2008, pp. 141–142.

[9] H. M. Ahmed, D. Gracanin, A. Abdel-Hamid, and K. Matkovic, "An approach to interaction interoperability for distributed virtual environments," in *EGVE '08: Short papers and posters proceedings*, 2008, pp. 35–38.

[10] H. M. Ahmed, D. Gracanin, and A. Abdel-Hamid, "A framework for interaction interoperability in X3D mobile collaborative virtual environments," in *INFOS '08: Proceedings of the 6th International Conference on Informatics and Systems*, 2008, pp. 84–92.

[11] I. Poupyrev, S. Weghorst, M. Billinghurst, and T. Ichikawa, "Egocentric object manipulation in virtual environments: Evaluation of interaction techniques," *Computer Graphics Forum*, vol. 17, no. 3, pp. 41–52, 1998.

[12] Y. Osais, S. Abdala, and A. Matrawy, "A multilayer peer-to-peer framework for distributed synchronous collaboration," *IEEE Internet Computing*, vol. 10, no. 6, pp. 33–41, 2006.

[13] D. Margery, B. Arnaldi, and N. Plouzeau, "A general framework for co-operative manipulation in virtual environments," in *Virtual Environments '99: Proceedings of the Eurographics Workshop*, 1999, pp. 169–178.

[14] D. Navarre, R. Bastide, A. Schyn, L. P. Nedel, and C. M. D. S. Freitas, "A formal description of multimodal interaction techniques for immersive virtual reality applications," in *Proc. of the Tenth IFIP TC13 International Conference on Human-Computer Interaction*. Springer, 2005, pp. 170–183.

[15] J. Soldatos, I. Pandis, K. Stamatis, L. Polymenakos, and J. L. Crowley, "Agent based middleware infrastructure for autonomous context-aware ubiquitous computing services," *Computer Communications*, vol. 30, no. 3, pp. 577–591, February 2007.

[16] D. Martin, A. Cheyer, and D. Moran, "The open agent architecture: A framework for building distributed software systems," *Applied Artificial Intelligence*, vol. 13, no. 1/2, pp. 91–128, 1999.

[17] W. Jia and W. Zhou, *Distributed Network Systems*. Springer US, 2005, vol. 15, ch. Overview of Distributed Network Systems, pp. 1–13.

[18] Nintendo. (2009, January) Wiimote. [Online]. Available: http://www.nintendo.com/wii/what/controllers/

[19] R. Bastide and P. A. Palanque, "A petri net based environment for the design of event-driven interfaces," in *Proceedings of the 16th International Conference on Application and Theory of Petri Nets*. London, UK: Springer-Verlag, 1995, pp. 66–83.

[20] Squeak. (2009, January) Etoys. [Online]. Available: http://www.squeak.org/

[21] S. Frees, G. D. Kessler, and E. Kay, "PRISM interaction for enhancing control in immersive virtual environments," *ACM Trans. Comput.-Hum. Interact.*, vol. 14, no. 1, pp. 1–31, 2007.

[22] L. Gallo, G. D. Pietro, and I. Marra, "3D interaction with volumetric medical data: experiencing the wiimote," in *Ambi-Sys '08: Proceedings of the 1st international conference on Ambient media and systems*. Brussels, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, pp. 1–6.