Journal of Universal Computer Science, vol. 7, no. 6 (2001), 458-471
submitted: 1/6/01, accepted: 3/6/01, appeared: 28/6/01 © *Springer Pub. Co.*

Knowledge Management and Collaborative Virtual Environments

Ivan Tomek
(Jodrey School of Computer Science, Acadia University, Canada
ivan.tomek@acadiau.ca)

Abstract: Knowledge management systems provide three basic services: information capture, storage and organization, and access. This paper argues that collaborative virtual environments (CVEs) provide features that make them uniquely suited as an integral part of information capture. After introducing CVEs, we present our work in this area and outline our future plans.

Key Words: Knowledge management, computer supported collaborative work, collaborative virtual environments, information capture.
Categories: C.2, D.2.6, H.1.2, H.3, H.5

1 Introduction

The realization of the essential value of information and knowledge in modern organizations, and the increasingly electronically based nature of work processes led to the development of the concepts of organization memory and knowledge management systems (KMS). The goal of KMS research is to create computer-based environments that capture both formal and informal data along with tacit knowledge hidden in the minds of the employees of a company, store it in an organized form, and allow its easy search. A knowledge management system thus essentially consists of three components that provide these functions to the utmost possible degree.

One of the most difficult problems of knowledge management is how to capture the maximum knowledge important to the organization. In a metaphorical sense, the ideal KMS capturing system would be a collection of agents who follow every employee day and night and monitor their work, communication, artefacts, and even thoughts, and enter them into a database. Although this is not yet possible (and hopefully never will), the fact that much of the work process now relies on the use of computers and computer networks makes this 'ideal' largely achievable. It is imaginable, that one could create a system that integrates all the tools that an employee uses – from work tools through Web browsers and e-mail clients - and captures, possibly preprocesses, and stores all relevant information. To maximize the organizational benefit from such a system, as much information-producing activity as possible should be performed in this system, including informal and unplanned encounters that are known to contain some of the most important work-related information. This is where Collaborative Virtual Environments could be useful.

In a very general sense, a Collaborative Virtual Environment (CVE) is a software environment that emulates some of the features of the real world. In practice, CVEs emulate at least the following aspects of physical reality: The concept of space, the representation of an object and a tool that can be moved within the universe defined by the CVE, the representation of a human or robotic individual (avatar) that can navigate within the space, hold on to objects and move them within the space, and communicate with other avatars. Most importantly, CVE universes are not only configurable but their users can even extend the features provided by the universe at runtime and without requiring the intervention of the manufacturer of the software.

How can CVEs be beneficial to KMS? Simply by channelling much of the work activity and created artefacts into a single integrating software medium. By capturing the relevant parts of the work process, organizing it, and providing data retrieval and data mining functions, a KMS could come close to the ideal described above and provide an environment that is not only productive and effective, but also fun to use, thus potentially improving workers' satisfaction.

In the following sections, we will explain why we think that CVEs are good for knowledge management, say more about CVEs, describe two experimental implementations that we developed to examine the possibilities for support of software development teams, and conclude with a section describing the goals of our current work and a list of selected references.

2 Why a KMS should include a CVE

As already stated, our premise is that a part of the knowledge capturing part of a KMS should be a collaborative environment should emulates and extends the essential properties of a physical workplace and complements physical and temporal collocation with virtual one. These properties include spatial organization of work places and their navigation, communication, awareness of the presence and basic activity of co-workers and work-related events, objects and tools that can be moved from one place to another, and unlimited extendibility and customizability. We will now give several reasons for the claim that these features are beneficial for KMS.

Emulation of physical topology is important as a natural metaphor because appropriate metaphors are very important for usability and learnability, which are the basic prerequisites for successful software in general and groupware in particular. It also provides a natural organizing principle - offices, meeting rooms, and document libraries are the concepts used to organize personal and team workspace. Besides, not only people, but information too can be organized spatially as we will show later.

In the physical world, space may be allocated for a particular use, equipped for it, and restricted for particular group of people. A virtual environment should also exhibit these features and provide means for enforcing privacy and separation. Navigation among the disjoint places is necessary for virtual displacement of users and their virtual holdings and, when properly implemented, can provide opportunities for useful chance encounters.

Communication is essential in every the work process and multiple simultaneous channels allowing communication to take place in disjoint and mutually invisible scopes occupied by separate teams provide a natural means for discussion and a basis for recording communication in context.

An essential part of work is timely notification of the occurrence of work-related events such as the release of a new version of code by a team member. Team members should be able to register their interest in specific events, and be automatically notified of their occurrence, for examination or to automatically trigger user-defined actions. Emulation of the physical world provides a natural basis for this kind of event-driven and event-aware operation.

Objects such as documents, and tools such as software development tools are at the heart of work. Emulation of the physical world and the ability to move objects and tools around and share them provides a natural parallel to the way that we deal with objects and tools in the real world. To be truly useful, an emulated work environment should be all-inclusive like the real world, and provide seamless access to external tools. The CVE should thus form a universe interfacing to all necessary software tools rather than being just another application disjoint from others.

The nature of work is unpredictable and development and decision processes naturally evolve along with technology. Consequently, closed and proprietary environments that can only be extended or modified by the original developer are doomed to fail; a useful work environment must therefore be reconfigurable, customizable and extendible - just like the real world. In our daily activities, we depend not only on our ability to rearrange our offices, build new buildings, use them for a variety of new purposes, move things from one place to another, and buy and install new office equipment. We also create completely new types of objects and tools, etc. A virtual environment should and can provide these facilities too.

The social aspects of work have many dimensions explored mainly in Computer Supportive Collaborative Work (CSCW). They include, for example, awareness, usage policies that control how tools and objects may be used according to the roles of individual team members, and work processes. A virtual environment emulating the physical world is a natural framework for supporting these concepts.

An important aspect of a physical work environment is that it allows contact beyond the limits of the current work team. Physically collocated workers can talk not only to people who are working on the same project, but also to others. This allows exchange of ideas and sharing of valuable expertise. Project-centred groupware does not provide this facility, but emulated virtual spaces make it possible. In recent studies of work environments [Churchill 1999a, 1999b] long-term CVE users reported that they commonly used the environment to meet co-workers working on other projects and even people who have left the enterprise, and used these virtual encounters to support their own work.

All these arguments lead us to the conclusion that emulation of those features of the physical world that are useful for collaboration - a CVE - is an optimal candidate for a KMS environment. Although it might seem that such an environment might only be useful for physically separated teams, this is not the case. As an example, it is generally accepted [Allen 1977] that as soon as distance exceeds 30 meters, offices should be considered as physically purpose as far as ordinary face-to-face communication is concerned.

What other properties beyond emulation of physical reality should such an environment have? The environment must be easy to learn and use, responsive, unobtrusive, and fun to use. An environment whose learning and use requires substantial effort or is slow and detracts from work will not be popular with busy software developers and will not be used. Churchill states that long-term users report that the 'light weight' nature of a low technology CVE and its ease of use and rapid response are among its main advantages.

3 Which Type of CVE?

Having accepted the premise that a CVE is a desirable component of a KMS, the next question is which of the several known CVE technologies is best. In our research, this translated into the more specific question, which CVE is best suited for software developers. Existing technologies include text-based [Hayes, 1998], virtual reality (VR)-based, and augmented reality. VR-based environments can be further divided into low-technology desktop environments based on VRML and similar techniques [Ames 1996, Damers, 1998], and high realism and high-cost environments [VRAIS'98, IEEE 2000]. Augmented reality is not really a CVE technology but its use can complement text-based or virtual reality environments with information gathered from sensors in the physical world, such as electronic objects in 'collaborative buildings' [Streitz, 1998]. We will now briefly comment on the suitability of these technologies for our original purpose to demonstrate that the most sophisticated technology is not necessarily the best in all circumstances.

Modern *text-based collaborative virtual environments* use GUI windows, possibly enhanced with graphics, audio, or even video input and output. We will refer to these environments as MUDs (an acronym explained in the next section). Their advantages are that they are easy to learn and use, that they allow users to focus on contents rather than manipulation, and that they do not distract users with additional and non-essential visual context. Being low-tech, they provide high performance at low cost. MUDs can also be easily extended by users via simple commands or using an extendible programming language because the laborious extensions required in VR-based environments are missing. Another advantage, important for KMS use, is that text-based information can be easily captured, organized, and searched. Although a GUI is a rather poor approximation of the real world this is not a serious drawback when focus is on textual and diagrammatic information. Besides, our experience shows that users quickly acquire a feeling of physical collocation with other occupants of the virtual space even without the visual cues.

Environments based on virtual reality provide a more accurate approximation of physical reality. The price of realism is obvious in comparison with the 1D space of textual CVEs – 3D models are more difficult to use (requiring, for example, manipulation of the user's avatar on the screen via mouse buttons) and this distracts the user from communication. 3D models are also much more difficult to expand and modify, and the additional visual cues enlarge the amount of information that the user must process without providing a significant benefit in applications that don't require spatial models.

Augmented reality is an extension of MUD and VR environment and remains, still relatively unexplored. Although it has great potential benefits for CVEs of all kinds, we have not considered it at this stage of our research.

After considering the above parameters presented, we concluded that MUDs are the best fit in applications where the essential features are communication and textual or graphical artefacts, where customizability and extendibility are essential, and where capturing and accessing verbal contextual information is important. VR-based environments may be the best choice for applications where visual information is essential, as in e-commerce, architectural design, etc. Since our focus was the use of CVEs for software development teams whose interests are primarily in textual and graphical documents and exchange of ideas, our work was limited to text-based CVE environments. This decision has many precedents in research and commercial products [Fitzpatrick, 1996, Harrison, 1996, Journal of MUD Research, Lindstaed, 1997, Mansfield, 1997, Poltrock, 1997, Roseman, 1996, Spellman, 1997, Steed, 1998, TeamWave].

4 What is a MUD

In the previous section, we explained why we explored text-based CVEs based on the MUD concept. MUDs first appeared in the late 1970s. The acronym originally stood for Multi-User Dungeons because the first MUDs were networked implementations of the popular fantasy game Dungeons and Dragons. Eventually, the acronym acquired additional and more respectable interpretations including Multi-User Domains and Multi-User Dialogs, particularly when new versions appeared designed for environments supporting socialization. In the late 1980s, Paul White developed an object-based implementation of MUD referred to as MOO, for MUD Object-Oriented. Pavel Curtis at Xerox then re-implemented the concept and created Lambda MOO [Haynes, 1998], which has since become the most common basis of MUD implementations. In the following, we will use MUD as a generic term representing both MUDs and MOOs.

A classical MUD implementation consists of a single server with multiple clients connected through a Telnet client. After connecting, users would communicate with one another, navigate in the textual universe, and perform other operations by typing commands interpreted by the server, which would then update its database and communicate back to all interested parties using ASCII text. Recently, Telnet interfaces have mostly been replaced with GUIs, usually a Web browser with HTML and Java support. Beyond the change in the user interface, however, the principles of the environment and its implementation have remained essentially unchanged.

As mentioned, a MUD emulates selected features of the physical world inhabited by user proxies called avatars, objects, and tools. The emulated world (universe) is divided into disjoint but possibly nested domains corresponding to interconnected rooms, buildings, and other familiar concepts. These spaces serve as repositories of objects and tools and their separation restricts communication to users collocated in them.

Avatars representing active MUD entities can move from one virtual place to another (navigate) carrying their possessions (objects and tools) with them. They can pick up and drop objects and communicate with other avatars. The environment is persistent and the database containing the universe with its interconnected places, objects, tools, and agents resides on the server. The application is continuously running.

Avatars representing users are generally constrained by roles that authorize them to perform certain actions but prevent them from executing others. The main differences between different roles are typically in the limits on how much an avatar can extend and customize the MUD universe. The lowest level avatars can instantiate a predetermined number of entities such as rooms, objects, and tools, whereas users with higher authorization can instantiate a larger number and a greater variety of objects, and possibly create new types (classes) of objects and functionalities and add them to the environment.

Our brief description shows that a MUD satisfies the requirements postulated in the previous section: It is a persistent emulation of essential features of the real world, provides navigable scopes that can be used for separate activities using and producing tools and artefacts, incorporates the concepts of time and events, and provides means for synchronous and asynchronous communication. Its users can perform assigned roles that may be used to define policies for the use of tools and other tasks. MUDs are easy to use, easily accessible, and low tech applications and, most importantly, are user-extendible and customizable at two levels: A user can instantiate existing classes of entities and extend the universe by extending existing types of entities and defining new ones.

In the following two sections, we will now describe the two environments that we developed and used to explore some of the requirements enumerated above.

5 Jersey

Our interest in CVEs was sparked by a request from a software development company with offices around the world and looking for a way to deal with geographic separation of team members. They provided us with the skeleton of a non-commercial project called Jersey implementing a basic MUD architecture. The environment used a Telnet connection and all client communication had the form of Smalltalk messages transmitted as ASCII text and interpreted by a Smalltalk server. This was not only acceptable because the intended users were Smalltalk programmers, but even desirable: Since the server interprets incoming Smalltalk messages using an open Smalltalk ‘image’, users can send not only Jersey-specific commands, but any legal Smalltalk expressions. This lets them customize and extend the universe at run time. This, of course, violates basic security because it allows the user to program the server to do such things as capture all Jersey communication, damage or even destroy the whole Jersey universe and its engine or even the Smalltalk environment itself, but this was not an important consideration because security was low on our priority list.

Besides this peculiarity, the Jersey was a typical MUD, although the implemented functionality was minimal. The framework provided support for instantiation, communication, and navigation, a basic architecture for creating software agents, and a hierarchy of 'utility' and 'information holding' objects. A substantial limitation of Jersey was its flat layer of virtual rooms with no support for containment. After installing Jersey, we embarked on experimentation and expansion [Tomek, 1998, Tomek 1999 a]. Our work and use was restricted to a small team of faculty and MSc students and to several software-engineering courses. We held class and team meetings on Jersey and used it for inspection of Jersey code. We also used the concept of disjoint domains to organize our project artefacts (see below).

Our extensions of the original implementation included new user interfaces, new objects and tools, new types of agents, on-line help and query support, access to Smalltalk code for use in software inspections, and Development Clusters - templates of room combinations to accommodate development teams. We will now briefly describe these features.

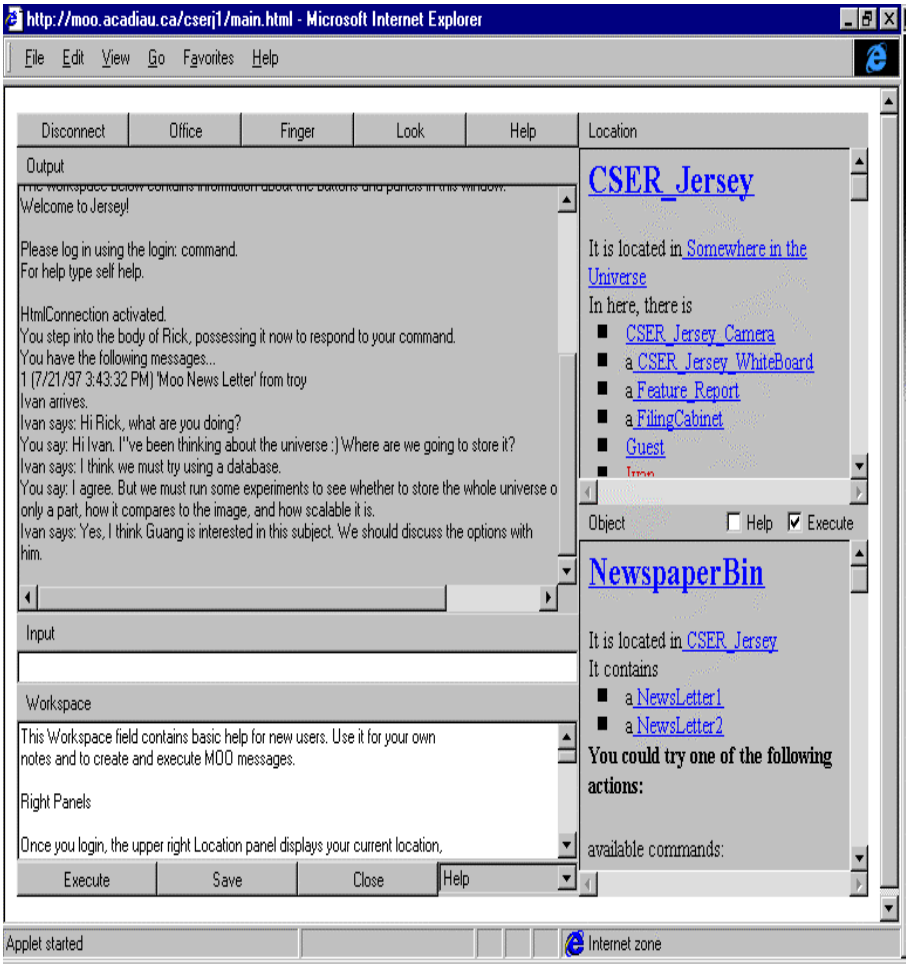

Figure 1: Main Jersey WWW Interface

5.1 Functionality

Our most visible change to Jersey was an interface using a Web browser. The main window is shown in Figure 1. Besides providing buttons to perform the most common actions, it contains several specialized widgets for sending commands to the server, removing the necessity to type them as Smalltalk commands.

As in any MUD, the total number of Jersey messages is large and their details difficult to remember. The new interface provides several unusual ways to alleviate this problem. One is the Workspace field that can be used as to hold frequently used commands and other text. A help facility can be activated by the Help button and displays help information in the upper part of the window. The Tree Help agent described below can also provide environment help and switching from *command execution* mode to *command help* mode via radio buttons provides another form of help. Besides direct command (or Smalltalk code) entry, commands associated with rooms and objects can be activated via clickable text. These simple improvements greatly improved Jersey's usability and usability is, as we mentioned above, one of the critical requirements on a successful CVE. The top right-hand side of the main window displays clickable information about the user's current location, showing all accessible objects and agents, as well as exits to adjacent rooms. When the user selects an item, the list underneath shows information about the selection and its executable commands as clickable text.

In addition to the new UI, we also added a number of new classes of objects and agents. The new objects can be divided into utility objects and information capturing objects. Utility objects include offices equipped with tools, desks consisting of a desktop and drawers, garbage cans emptied periodically by janitor agents, and copiers for duplication of documents and other text objects.

Objects for capturing, organizing, and keeping information include filing cabinets, document objects providing URL-based links to information stored beyond Jersey boundaries, such as Word documents, a camcorder (a moveable version of a stationary 'security camera', which is ' bolted to a wall'), a whiteboard for recording and displaying information, and other objects.

A general-purpose query facility is implemented by an agent-based Help Tree, a variation on Answer Garden 2 [Ackerman, 1996]. The agent called Tim processes queries from users and tries to match them against answers stored in folders in a 'filing cabinet'. If it fails to find an answer, it sends the question to registered experts, stores answers in folders for future use and delivers them to the user who issued the query. Tim is an example of an autonomous agent, a software entity that can autonomously move from one place to another, interact with the environment, and execute scripts. Agent actions are based on a combination of mission actions defining the purpose of the agent, autonomous random actions providing a semblance of free will, and response actions that allow the agent to respond to events in its environment. Examples of agents include messengers who deliver messages, janitors who empty office trashcans, and an agent who makes copies of new issues of the Jersey newsletter and delivers them to subscribers. A notifier agent, notifies interested users about new releases of code modules to which they subscribed. This agent requires a brief comment because it illustrates that 'rooms' don't have to be used only in the obvious real-world-like interpretation. Although Jersey is developed in Smalltalk, its user interface cannot access code in other images and we thus created Class that allow viewing of selected classes. Tim's operation is based on navigating Class Rooms.

To make it easy to create virtual spaces for teamwork, we created a template of a basic floor plan called the Development Cluster. This is a collection of rooms centered around a project room used mainly for meetings and providing access to the offices of the developers working on the project, the leader of the team, and documentation. It also includes the above-mentioned Class Rooms.

5.2 Experience

Although Jersey was a fully working and user-friendly environment, its final state was still quite rudimentary for use in a commercial software development environment. We used it in a senior programming class and evaluated it informally by a questionnaire. The results can be divided into general MUD experiences and an evaluation of Jersey-specific issues.

On the MUD side, we found that the powerful and user-friendly interfaces described above made the use of Jersey much easier than the original Telnet interface. The evaluation also showed that the interface was surprisingly immersive, giving the physically separated users (often working as much as 50 kilometres apart) a profound sense of being together. After the initial chaotic formal meetings, we developed a meeting tool with simple support for agenda, chair and secretary roles, and floor control, which helped to make meetings more manageable and improve usability. It is interesting to note that this support was developed by a student using Jersey during a Jersey meeting.

One obvious disadvantage of text-based communication is the relatively slow speed of typing and the resulting occasional lack of coherence of individual contributions. This, of course, is the same as in any other form of computer-based synchronous communication. Without some control, threads of communications quickly diverge, and following them becomes difficult. This has been noted and various approaches attempted as in [Highsmith, 1999], but we have not attempted to address it.

As we started using Jersey, we quickly discovered the need for new objects and new functionality. The ability to extend and modify the environment turned out to be essential, and is one of the most important features distinguishing MUDs from related environments such as chats, e-mail, or virtual environments based on predefined structures. Not even the most intelligent and creative designers can anticipate all the possibilities and needs, and cannot create an environment that will satisfy all potential users in all possible applications. Even providing users with a scripting language and customizable templates seems too restrictive.

A shortcoming of our experiment was that our group was very small and the users were students rather than the targeted professional developers. We also failed to take full advantage of one of the most important benefits of a MUD - that it can provide constant informal access to other members in the group. Churchill reports on a longitudinal study of a work team that used a MUD over a period of several years and found that its users have a MUD window open 24 hours a day, occasionally checking the ongoing informal exchanges, and jumping in when a relevant topic is addressed or when they needed to ask a question. This closely resembles recently reported advantages of the use of 'warrooms' in certain types of projects [Teasley, 2000].

Among the weaknesses that Jersey shares with other MUDs we found the fact that it was isolated from other applications most annoying. This was somewhat mitigated by the earlier-mentioned URL-based access to various types of documents, but future MUDs should encompass other applications such as word processors, spreadsheets, e-mail, and software development tools.

Our list of Jersey conceptual weaknesses starts with its lack of universal support for events. This precludes, for example, easy subscription to events and automatic notification of subscribers when these events occur. As mentioned above, we had to resort to the concept of Class Rooms periodically scanned for new releases by a specialized agent to solve this problem.

The centralized nature of the architecture with a single server storing the universe and performing all processing is also a shortcoming. It makes the environment too vulnerable and has the potential of overloading both the processor and the network connection. The fact that the universe 'database' is, in fact, a part of the Smalltalk environment rather than a true database is also a limitation.

As already mentioned, Jersey does not address security issues but we did not consider this to be a problem because of the experimental nature of the project and the fact that more fundamental issues needed to be resolved first. Moreover, Jersey was intended for use in a limited and safe user community where security was not a problem. In a production environment, reconciling security with openness and extendibility will be very important.

Finally, Jersey does not support advanced CSCW features such as groups, roles, policies, work processes, and tasks. These seemingly exotic features are very important in team activities such as code inspections and general meetings, which require support for roles such as chairperson, secretary, meeting participant, and others, each possibly with different tasks, tools, and authorizations. Concepts such as groups are also essential for creating flexible locking mechanism for rooms and objects, and for providing basic security.

To explore some of the issues raised above, we started a new project from scratch. The project is called MUM and is described in the next section.

6 MUM

MUM is a project resulting from our experience with Jersey [Tomek, 1999b, 2000]. The acronym stands for Multi-Universe MOO and reflects one of the unusual design features– allowing an unlimited number of interconnected client/server machines that allow avatars to move from one universe to another with all their holdings. The main issues that we explored with MUM include

- minimization of the load on the server and the network by letting the clients do as much work as possible and providing a framework for creating flexible and customizable user interfaces implemented essentially as downloadable plugins
- generalization of the nature of communication between the client and the server from text-based to binary data-based

- provision for event-driven operation
- independent multiple server operation.

MUM is fully event-driven. Everything is an event and results from passing an event from one object to another. Events are ‘subscribable’ and an authorized object or agent can register interest in its occurrence and is automatically notified when the event occurs. Notification has the form of a ‘notification event’ that contains enough information to allow the subscriber to respond with a pre-programmed action. Partially dictated by the need to support authorization, MUM implements groups. Groups allow the owner of an object to specify who can subscribe to the events processed by the object and can be used for other effects such as locking rooms, controlling communication, and other control.

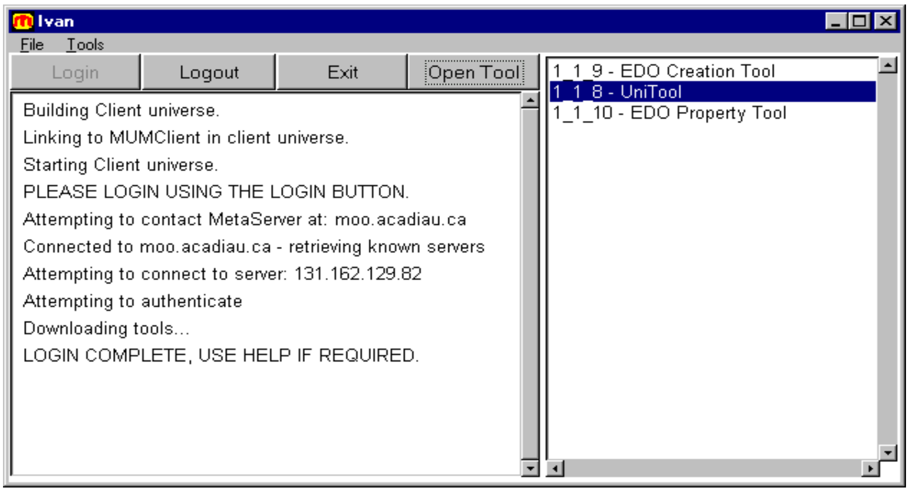

Figure 2: MUM Launcher

MUM also has a new user interface designed to be used via commands displayed in the user interface rather than typed commands. UI windows are often specialized for a particular purpose and downloadable as plugins. Figure 2 shows MUM's Launcher, which is used to connect or disconnect, obtain additional 'tools' from the server, and display system messages.

Figure 3 shows MUM’s ‘Universal Tool’ displaying the current location with objects, occupants, and exits and simple awareness support. It displays commands understood by the selected object and can provide information about each command, execute it, or subscribe to its occurrence. The left-hand side is for communication. The top pane displays communication addressed to the user by the server, mostly communication from other occupants of the room, and the pane at the bottom is for input.

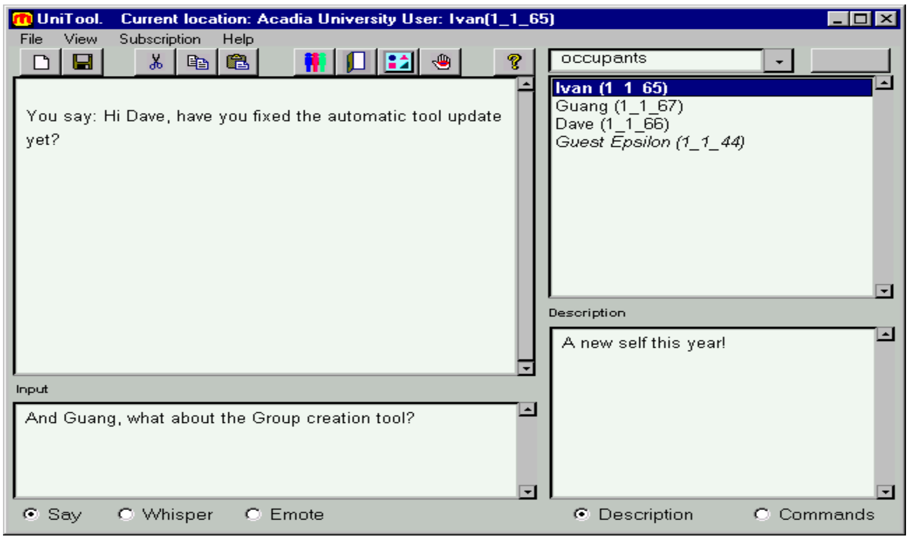

Figure 3: UniTool

6.1 Functionality and experience

MUMs functionality is limited because most of our work focused on the development of the framework. It implements basic MOO operations such as communication, navigation, instantiation of a limited number of types of objects, and management of their properties. The current repertoire includes users, groups, places (nested scopes corresponding to buildings containing floors, rooms, etc.), cameras (for recording communication on tapes), tapes, a simple mail system and a code management tool allowing versioning and code membership [Yang 2000]. Group definition, object creation, and object properties, and the camera object have their own tools with specialized windows.

Our experience with MUM has been limited to the small team of developers and further development was stopped when we realized that we should make some major adjustments in our approach. These include restricting the use of events and, most importantly, formulation of a protocol that will allow the environment to be fully portable and interoperable among different platforms and languages. These features are the subjects of our current work.

7 Current work

Our current goal is to make our CVE interoperable among platforms and languages in the way that Internet protocols such as FTP and HTTP are. For this purpose, we are now developing a protocol to support MUM-like concepts. Unlike existing Internet protocols, the essential goal of our approach is support for complete customizability and extendibility, which requires certain low level underlying categories of protocol support. CVE-specific commands will not be built into the protocol but rather a mechanism for integrating them at run-time.

Once the basic command set is defined, we will implement a framework supporting it and test how complete and usable it is by implementing features that proved to be useful in Jersey and MUM. We will then test the environment in a work setting, and release it and the protocol to the public, possibly implementing the clients and servers in other programming languages. After this, work on the integration of the CVE into a KMS can begin.

8 Conclusion

We have presented arguments to support the view that the information capturing part of a KMS should include a Collaborative Virtual Environment. Inversely, we propose that a CVE should contain a KMS as its component. We have described our earlier CVE projects that include some innovative features deemed essential for collaborating teams, and formulated requirements that such a CVE should satisfy. We have concluded that the environment should be platform independent and interoperable across multiple implementations. It should also be integrative, allowing other software tools to be interfaced to it to provide a unified seamless environment. We are currently working on the basic design and implementation of such an environment.

References

[Ackerman 1996] Ackerman, M. S., McDonald, D. W. (1996). Answer Garden 2: Merging organizational memory with collaborative help. In Proceedings of CSCW'96: 97-105.

[Allen 1977] Allen, T. J. (1977). Managing the flow of technology: Technology transfer and the dissemination of technological information within the R&D organization, MIT Press, Cambridge, MA.

[Ames 1996] Ames A. L., et al. (1996). VRML 2.0 Sourcebook, John Wiley & Sons.

Baecker R. M. (1993). Readings in groupware and computer-supported collaborative work,

[Churchill 1999a] Churchill, E., Bly S. (1999) Virtual environments at work: Ongoing use of MUDs in the workplace, In Proceedings of WACC'99.

[Churchill 1999b] Churchill, E., Bly S. (1999) It's all in the words: Supporting work activities with lightweight tools, In Proceedings of Group'99.

[Damers 1998] Damers, B. (1998). Avatars! Peachpit Press.

[Fitzpatrick 1996] Fitzpatrick, G., Kaplan S., Mansfield T. (1996). Physical spaces, virtual places and social worlds: A study of work in the virtual, In Proceedings of CSCW'96.

[Harrison 1996] Harrison, S., Dourish, P. (1996). Re-Place-ing Space: The roles of place and space in collaborative systems, In Proceedings of CSCW'96.

[Haynes 1998] Haynes, C., Holmevik, J. R. (1998). High Wired: On the Design, Use, and Theory of Educational MOOs. University of Michigan Press.

[Highsmith 1999] Highsmith J. (1999), Managing distributed project teams, e-business application deliver, e-Business Application Delivery, August 1999, at http://cutter.com/ead/ead9908.html.

[IEEE 2000] IEEE, (2000), Virtual Reality 2000, http://www.caip.rutgers.edu/vr2000/

[Journal of MUD Research] Journal of MUD Research, http://journal.tinymush.org/~jomr/

[Lindstaed 1997] Lindstaed, S., Schneider, K. (1997). Bridging the gap between face-to-face communication and long-term collaboration; In Proceedings of Group'97.

[Mansfield 1997] Mansfield, T., Kaplan S., Fitzpatrick G., Phelps T., Fitzpatrick M., Taylor R., Segall B., Herring C., Johnson P., Berry A. (1997): Evolving Orbit: A progress report on building locales; In proceedings of Group'97.

[Poltrock 1997] Poltrock, S.E., G. Engelbeck, G. (1997). Requirements for a virtual collocation environment. In Proceedings of Group'97.

[Roseman 1996]Roseman, Greenberg, M., S. TeamRooms (1996). Network places for collaboration. In Proceedings of CSCW'96.

[Spellman 1997] Spellman, P., Mosier J.N., Deus L.M., Carlson J.A. (1997). Collaborative virtual workspace. In Proceedings of Group'97 ACM SIGGROUP: 197-203.

[Steed 1998] Steed, A., Tromp, J. (1998). Experiences with the evaluation of CVE applications. In Proceedings of CVE'98: 123-132.

[Streitz 1998] Streitz, N., et al., (1998). Roomware for Cooperative Buildings: Integrated deSign of Architectural Spaces and Information Spaces, Lecture notes in Computer Science 1370, Springer-Verlag.

[TeamWave] TeamWave, http://www.teamwave.com/

[Teasley 2000] Teasley, S.D., Covi, L., Krishnan, M.S., & Olson, J.S. (2000). How does radical collocation help a team succeed? In Proceedings of CSCW 2000. 339-346. New York: ACM Press.

[Tomek 1998] Tomek, I., Nicholl, R., Giles R., Saulnier, T., Zwicker J. (1998). A virtual environment supporting software developers. In Proceedings of CVE '98.

[Tomek 1999a] Tomek, I., Giles R (1999a): Virtual Environments for work, study, and leisure. Virtual Reality, Vol. 4, no. 1.

[Tomek 1999b] Tomek, I., Murphy, D., Yang, G., (1999b): MUM – a multi-universe MOO, Proceedings of WebNet '99, October 1999.

[Tomek 2000] Tomek I (2000): The Design of a MOO, Journal of Network and Computer Applications, Vol. 23, No. 3, Jul 2000, pp. 275-289.

[VRAIS 1998] VRAIS '98: Virtual Reality Annual International Symposium (1998), IEEE Computer Society, http://www.eece.unm.edu/eece/conf/vrais/.

[Yang 2000] Yang G., Ivan Tomek (2000): Team Lab: A Collaborative Environment for Teamwork, CRIWG'2000, Madeira, Portugal.