NISTIR 7441

Collaborative Augmented Reality for Better Standards

Matthew Aronoff[a1] and John Messina[a]

[a]Computer Scientist, National Institute of Standards and Technology, USA.

Abstract. Concurrent engineering depends on clear communication between all members of the development process. As that communication becomes more and more complex, the quality of the standards used to move and understand that information likewise becomes more and more important. If the standard is incomplete, redundant, or ambiguous, most of the expected benefits are lost. In order to improve data exchange standards, explicit data models are required. However, creating those data models is a process that requires collaboration between domain experts. What is needed is a solution that encourages interaction without requiring a high level of data modeling expertise. Focus is a software tool designed to provide such an environment. It is a distributed design and conferencing application which uses augmented reality to allow domain experts to come together in real time for data modeling. By developing Focus, we hope to allow domain experts to create data models without first having to learn complex UML modeling programs. Because of the networked nature of Focus, it is easier to ensure the participation of the best domain experts regardless of location. This paper details the development, features, and expected benefits of Focus in a collaborative engineering environment.

Keywords. 3d, collaboration, data modeling, UML, standards development, Ruby.

1 Problem

Concurrent engineering depends on clear communication between all members of the development process. As that communication becomes increasingly complex (including CAD models, diagnostic data, process control data, etc.), the quality of the standards used to move and understand that information likewise becomes more and more important. Current information transfer standards tend to be designed by domain experts. Traditionally, a group of domain experts gather together and begin to develop the new standard in an *ad hoc* manner. Requirements and document structure are discussed, a written description is produced, and the standard itself is created in a way that agrees with the documentation.

Standards developed this way often suffer from ambiguity, redundancy, missing information, and/or excessive complexity. Terms used in the textual description may be ill-defined; the result is often that the standard contains a vaguely defined element, or that one developer expects an element to mean one thing, while another developer is equally sure that it means something else. Apart from the confusion

[1] Computer Scientist, National Institute of Standards and Technology (NIST), 100 Bureau Drive, Gaithersburg, Maryland, 20899, USA; Tel: 301 975-8778; Fax: 301 975-8069; Email: matthew.aronoff@nist.gov; http://www.eeel.nist.gov/812/IIEDM/

caused, ambiguities mean that constraints upon the valid range of element data are often underused, which can impede automated processing of the standard. Furthermore, as the standard itself is created by hand, information found in the textual description may be missing from the standard altogether. If information is repeated, or referenced multiple times, in the textual description, it may also appear several times in the standard. Empty (and therefore unnecessary) "container" elements may become part of the standard because they were part of the implementing experts' mental framework during the conversion from textual description to formal standard.

All of the above problems stem from an attempt to combine two separate steps of the development process into one. The first step is to determine the standard's domain, which is generally the answer to the question "*what* information and/or interactions are we trying to capture?" The second step is to create a particular implementation, which can be characterized by the question "*how* will that information be stored and moved around?" When those two steps are performed simultaneously or in reverse order, the standard often suffers. What is needed is a tool that allows domain experts to work solely on the domain of a standard without falling into the trap of designing an implementation at the same time.

2 Solution: Focus, a distributed 3D modeling tool

The Focus 3D telemodeling tool attempts to improve the quality of information exchange standards by borrowing a proven technique from the software industry: data modeling. A formal data model captures only the domain information of a standard; implementation questions are entirely removed from the process. Domain experts are therefore able to describe the required data and interactions with the appropriate amount of complexity, without concern about the eventual language used to implement the standard and its particular idiosyncrasies. This approach not only provides a clear, unambiguous description of the scope of a standard, it also reduces the amount of time necessary to create the implementation, as much of that process can be automated using existing software tools. Moving the definition of a standard away from a textual description also makes for quicker consensus between experts, as language and style issues no longer factor into acceptance of the standard.

A proper collaborative data model, however, can be difficult to achieve. The cost of face-to-face meetings continues to rise, and existing solutions for remote collaboration fall short when considering the problem of data modeling. Audio conferences are insufficient, as data modeling is almost entirely a visual task. Existing visual tools, such as web and video conferencing, are limited in that only one person can actively work on a model at any given moment.

To address the aforementioned problems, we developed Focus. It allows domain experts to create data models without having Unified Modeling Language (UML) expertise, and without being in the same physical location. Focus uses augmented reality to create a shared, 3D environment. Abstract data modeling concepts like

classes, associations, and generalizations are represented with concrete objects that can be directly manipulated by the users.

Augmented reality deals with "the overlay of virtual imagery on the real world." [2] More technically, it is the process of using specified, predefined marker objects to position rendered, 3D objects within a real video stream. In Focus, that process is combined with a 3D display device like a head-mounted display, a head-mounted camera, and a hand tracking device. The result is that the user is presented with a 3D data model that appears to float, for example, atop the user's desk. The user can reach into the model and make changes, and any other logged-in users, in remote locations, can see the same model. The model behaves like a real object -- the user can get up and walk around it to see it from different angles.

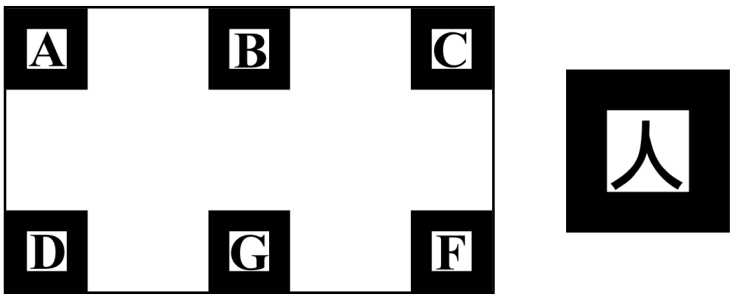

Figure 1. Example primary (left) and secondary (right) fiducials

Each user in the system is given a virtual representation (an *avatar*) that shows their position relative to the model.

3 Tracking the User

In order for a user to interact with the Focus environment, it is necessary to know three things: their position relative to the model, the position of their hand relative to the model, and any actions they are performing on the model. Position tracking relies on two visible markers, or *fiducials*, as seen in Figure 1. The primary fiducial represents the position of the model; it is free standing, and would typically rest on a desk or table in front of the user. The secondary fiducial is attached to the user's hand, and tracks that hand's position.

The user's perspective on the scene is extrapolated from the apparent size and orientation of the primary fiducial in the user's video camera feed. Each user's camera provides a video feed that duplicates their real-world point of view. Whenever a new frame of video is available, Focus searches that frame for the two fiducials, and provides their position in 3D space. The data model is rendered at the position of the primary fiducial; a cursor and other indicators are rendered at the position of the secondary fiducial. Anyone else in the system also has a

perspective on the model, so the user sees other avatars appear where they are "standing" relative to the model.

Actions performed by the user are captured in one of two ways. A data glove, capable of tracking finger positions, can be worn, in which case the user interacts with Focus using a simple gesture language. Alternatively, the user can simply hold any device that can provide the system with left- and right-click functionality; in this case, the system uses standard mouse events to trigger actions. Since Focus uses the secondary fiducial to track the hand's position, a small presentation remote works just as well as a mouse. Focus treats all actions as occurring at the hand's position. For example, if the user wants to create a new object, they would reach their tracked hand into the model at the position where the new object should be, and perform the "Create" gesture. The resulting configuration is flexible, in that only a camera and a pointing device need to be connected to the system, and scales well with increasing video (and therefore fiducial-tracking) quality, since no changes are required when an upgraded camera or display device is attached.

4 Modular and Distributed-Computing-friendly

Focus was designed from the beginning to be modular. Each piece of the system can be replaced with a minimum of fuss. Video input, fiducial tracking, user input, and 3D rendering are all designed to be swapped out with new modules as technology improves. Focus uses a NIST-developed dependency injection framework called Trimurti [12] to decouple these application modules. Each module in the system knows what services it provides, and what services it requires. Trimurti is responsible for connecting the requested modules together, taking into account each module's requirements.

One advantage to this approach is that it makes the addition of new functionality very easy; as an example, we have implemented two different modules for user input (gesture and mouse input), which can be interchanged with a single name change in the module assembly code for Focus. Another advantage is that modules can be provided remotely. Each module knows that its required services have been provided, but it speaks to those services through Trimurti, which can make use of a distributed computing framework. The clearest example of this process is that the main repository, in which all Focus projects are stored, will be kept on a separate server; all Focus clients will access the repository as if it were local, but it will be supplied remotely. Distributed computing also allows us to run a Focus client on a nominally underpowered machine. Fiducial tracking, a fairly computationally expensive process, can be offloaded to another machine, again requiring only a single-line code change.

5 Hardware and software

When creating Focus, we felt it was important to keep the cost to a minimum. The target was to keep total system cost below the price of a dedicated video

conferencing system. Focus uses inexpensive, currently available hardware, and is built using only free software. It is designed to be cross-platform; initial development was done on Mac OS X, but all software components work on Linux and Windows as well. All Focus software will also be made freely available for further development.

For video capture, we are using a few different Firewire-based web cameras; all provide video frames in standard delivery formats, and are therefore supported on multiple platforms. Gesture input is handled with an inexpensive data glove which tracks finger positions, but the system can also be operated using a standard mouse. The most expensive component of the system is the head-mounted display (HMD). This is also the component likely to see the most improvement over the next few years. We are primarily using a display which makes use of relatively new organic light-emitting diode (OLED) displays to provide bright displays that are high-resolution when compared with other, similarly-priced HMDs. However, as the technology matures, further increases in both resolution and the user's perceived field of view will certainly improve the usability of the system. HMDs in this class can be found for less than $1000.

Focus utilizes the following software:

- ARToolkit [2] for user position tracking and real video capture
- Coin3D [5] for 3D scene management and rendering
- Libp5glove [10] to interface with the data glove
- DRb [6] for the underlying distribution of our modules
- Trimurti [12] to decouple the application modules

Additionally, Focus is written in Ruby. As a number of the software components were not capable of communicating in Ruby, we created Ruby-language bindings for ARToolkit, Coin3D, and Libp5glove using SWIG [11]. Using Ruby, a higher-level, object-oriented language, shortened the overall development time; adapting these existing and relatively mature toolkits was considered a better solution than attempting to write our own tracking algorithms and glove interface driver.

6 Designed for simplicity

The primary purpose of Focus is to allow domain experts to create data models quickly, so simplicity of interaction is critical.

- Whenever possible, the interface makes use of direct manipulation: 3D drag-and-drop is used for model arrangement, new vertices are added to association lines by grabbing and pulling on the line.
- When direct manipulation is not feasible, Focus uses familiar interface metaphors, such as a tool palette like those found in graphics programs.
- The gesture language has been designed to use simple gestures, like "Create" and "Select," which may be concatenated.
- All actions are persistent, so the user does not need to "Save" progress.
- Some actions (deletion of model elements, certain renaming actions, etc.) require acceptance by the user, but until that acceptance is received, the action is

provisionally accepted and affected objects are clearly marked as needing attention.
- All data models are also persistent. A user may create a new model, but may also resume work on an existing model, which may have other participants already at work. When the current session is complete, the model simply waits on the server for further modifications.

Part of designing for simplicity required providing users with instant, non-blocking feedback. When an action is invalid, the user is told during the action rather than after failure. If items cannot be dropped in a particular location, their appearance changes while the user is dragging them. If an action does fail (e.g. the user drops the items even though they are marked as un-droppable in that location), Focus informs the user via an information display along the edge of the screen, and does not request user intervention before continuing. The same information display is used to inform the user of other changes to the environment, such as new users joining a domain modeling session.

Searches follow the instant, non-blocking philosophy as well. As search terms are entered and modified, results are highlighted within the model, and the resulting selection can be used for any of the other available actions in Focus. One of the advantages of a 3D model layout is the ease of defining a spatial search. A user can query the model in a traditional way (looking for occurrences of a particular piece of text or range of numbers), but can also include spatial criteria (results within a certain distance of the user's position, or of a selected element in the model).

7 How it works

Now that the underlying technology and design philosophy have been discussed, the next step is to explore what a Focus modeling session is like. In this example, a data modeling project has already been created on the repository server at some previous time. The user wishes to resume work on this project. She places the main fiducial, which is a rectangle roughly 5x7 inches in size, on the desk in front of her. She puts on the HMD and glove, and starts the Focus client program. At startup, it connects to the repository server; the user authenticates for a particular project, and is logged in. She is now participating in that project's modeling session. The data model appears atop the main fiducial on her desk, and other users' avatars appear around the desk at their respective relative locations.

Each object in the data model has a representation in 3D. Classes have a default representation (see Figure 2), but can also be assigned model fragments. For example, a circuit board class might have a 3D model of a board as its representation. Focus can read model fragments stored in SGI's OpenInventor format, which can be created from any number of free and for-pay 3D modeling tools. Associations, generalizations, and other connections between classes are drawn as lines.

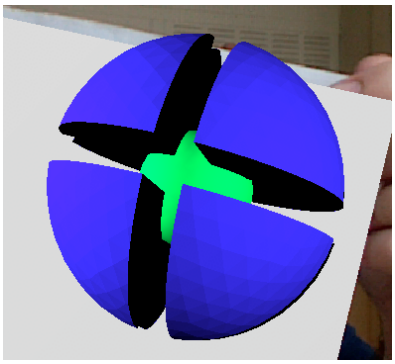

Figure 2. Default representation of a class

Each user working on the model has a 3D palette of tools available to them; one user's palette is active at any moment, and only that user may modify the model. (It should be noted that this was a design decision intended to enhance collaboration, and the single active palette may be abandoned in the future in favor of any user being able to modify the model at any time.) While inactive, the palette contains a representation of the active tool being used on the model, the name of the active user, and a button to request control of the model. Our user's palette is inactive to start. She presses the button to request control. When control is relinquished by the active user, her palette becomes active and populates with the available tools. These tools perform functions familiar to users of graphical editors -- tools for the different types of classes and associations that may be created, move, delete, and search. At this point, the user can make changes to the data model, rearranging, adding, modifying, and deleting model elements; all other users will see her changes reflected immediately in their own client applications.

8 Status

The design phase of Focus is complete, including the following steps:

- Use case catalog
- Publication on Focus architecture
- Investigated component technologies
- GUI design whitepaper published [1]

The implementation phase of Focus is ongoing. The following components have been completed:

- I/O hardware purchased
- Software components developed and tested
- Basic system functionality complete
  - Create, modify, rearrange and delete model elements
  - Simple associations between model elements

9 Future steps

Development on Focus continues, with several organizations planning to test it as the center of a model-driven process for standards development. Once the first iteration is complete, additional features such as more complete UML coverage and automated schema generation are planned. Additionally, as hardware prices continue to fall, it is possible that "see-through" HMDs (which display only the rendered 3D content on a transparent display) will be integrated into Focus, removing the display resolution issues inherent to small, inexpensive cameras.

10 References

[1] Aronoff M, Messina J, Simmon E. Focus 3D Telemodeling Tool: GUI Design for Iteration 1. NIST Interagency/Internal Report (NISTIR) 32470, 2006.
[2] ARToolkit. Available at: <http://artoolkit.sourceforge.net/>. Accessed on Feb. 23, 2007.
[3] Billinghurst M, Kato H. Collaborative Augmented Reality. Communications of the ACM, July 2002;45;7:64-70.
[4] Billinghurst M, Cheok A, Kato H, Prince S. Real World Teleconferencing. IEEE Computer Graphics and Applications, Nov/Dec 2002;22;6:11-13.
[5] Coin3D. Available at <http://www.coin3d.org/>. Accessed on Feb. 23, 2007.
[6] DRb (Distributed Ruby). Available at <http://chadfowler.com/ruby/drb.html>. Accessed on Feb. 23, 2007.
[7] Fowler, M. UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd edn. Boston San Francisco New York: Addison-Wesley, 2004; 35-52, 65-84.
[8] Griesser A. Analysis Catalog for Focus 3D Telemodeling Tool. NIST Interagency/Internal Report (NISTIR) 32090, 2005.
[9] Griesser A. Focus 3D Telemodeling Tool Use Cases For Iteration 1. NIST Interagency/Internal Report (NISTIR) 32096, 2005.
[10] Libp5glove. Available at <http://www.simulus.org/p5glove/>. Accessed on Feb. 23, 2007.
[11] SWIG. Available at <http://www.swig.org/>. Accessed on Feb 23, 2007.
[12] Trimurti. Available at <http://trimurti.rubyforge.org/>. Accessed on Feb 23, 2007.