Extensibility in a Visual Language for Web-based Interpersonal Communication

Steven L. Tanimoto

Dept. Computer Sci & Engin.
Univ. of Washington
Seattle, Washington 98195

Carlo E. Bernardelli

Visual Epistemologist

Rome, Italy

Abstract

One of the challenges of developing a visual language for human-to-human communication is coming up with a large enough visual vocabulary that users are not overly limited in the meanings they can express. If a language system can provide extension mechanisms directly to its end users, then the language, or various dialects of it, can grow with their use. Using an experimental prototype of a visual language called Vedo-Vedi, we present a mechanism that permits its users to define new objects, new icons for those objects, and relationships among those objects. The defined objects can be used to fill slots in frames that represent visual sentences, and they take places within a hierarchical menu system. One of the challenges of providing such an extension mechanism for a visual language is making it easy for users to define new icons, even if they are not comfortable with artistic drawing tools. In Vedo-Vedi, users can create new icons by selecting arbitrary rectangular regions within special images provided for the purpose, or for that matter, within any of the more than 80 images used in the system. Other possible extension mechanisms are also discussed along with implementation and performance issues.

1 Introduction

1.1 Motivation

In order to extend the possibilities of Internet-based messaging to situations where parties to a communication may not speak the same language, visual languages are being developed. Unlike older visual languages, these are computer-based, and so they can exploit computation in various ways. One of these ways is internal representation of some or all of the semantics of messages, which in turn, can permit animation, translation and data compression. The development of these computer-based visual languages therefore poses new challenges. Because users may want to communicate about many ideas and subjects, an important capability is a means within the language or language system to create new language objects and new constructs. In visual languages, new constructs typically involve new images or graphics, and the extension mechanisms may involve tools for design or synthesis of new images and icons. While visual languages can be easy for users to understand, the need to create new images can make it difficult for users to define their own terms. This paper addresses this paradox, discussing several techniques for extending visual languages. Some of them are demonstrated using the experimental prototype called Vedo-Vedi.

1.2 Previous Work

While cave paintings, hieroglyphics, written Chinese, and other visual modes of communication demonstrate a long and sophisticated evolution, computer-based visual languages are much more recent. Those created or proposed for interpersonal communication rather than for control of computers are very few and limited. Beardon [1] has proposed the use of diagrams representing case frames for human communication. Various systems for authoring of multimedia presentations can be considered to be visual scripting languages for human communication. However, these systems produce documents which must be "played" rather than read. Visual languages for animation include systems like CAEL [4], which tend to concentrate on control of geometry, lighting, and motion, rather than story elements.

At VL'97, one of us presented a survey of past work in visual languages for human-to-human visual communication along with a collection of research issues needing attention [3]. The reader is referred to that paper for additional background to the current one.

Previous work on extensibility of visual languages takes two forms: that applicable to all languages (e.g., subroutine mechanisms, means to alter syntax, preprocessing techniques), and those specific to visual languages (e.g., means to create new visual representations). There is a long tradition of research of the first type, documented in such publications as publications of the POPL conference.

The important question of how new icons can be easily created when needed in a visual language was addressed by Repenning for the case when the new icons are geometric transforms of existing ones [2]. A related technique is used in the Vedo-Vedi system reported here.

2 The Vedo-Vedi System

Vedo-Vedi is a visual language that permits graphical messages to be composed, "explained" (via sim-

ple animations), sent through the Internet, viewed, and translated into various natural languages. It currently is oriented towards messages about vacation travel that 10-year-old children might send to each other on the backs of picture postcards. The program is written in Java 1.1, and it can be run either as an applet from a server at the University of Washington or as a stand-alone application.

2.1 General Design

The Vedo-Vedi system consists of a script editor, a semantic subsystem, an animation engine, a translation component, and a messaging system.

The script editor contains a hierarhical menu of object types, a script panel in which the current script is displayed, and a panel of control buttons. A script is made out of Vedo-Vedi objects. Each Vedo-Vedi object is either a frame (playing the role of a verb) or a slot filler (playing the role of a noun). A script is a sequence of frames, each of which may contain zero or more slot fillers. A message is represented by one or more scripts.

The semantic subsystem contains a database that keeps track of certain relations defined among Vedo-Vedi objects. It also includes inference mechanisms that can derive certain simple conclusions from knowledge in the database. The semantic subsystem can keep track of time by counting minutes, hours, and days that elapse within a narrative time frame; it can work with both relative and absolute time and dates. It can keep track of space in terms of an inclusion hierarchy. It also keeps track of membership in the narrator's travelling group and the current location of the travelling group in the parts of the world that it knows about.

The animation subsystem consists primarily of a script interpreter and a cinematographic library of simple effects that the script interpreter uses to present a dynamic display of a script.

The translation mechanism employs for each target language (e.g., Italian) a file of schemata, one per frame type, to translate a script into natural language. This mechanism is designed to coordinate with the animation engine in such a way that fragments of sentences can be displayed as cinematographic subtitles synchronized with animation events.

The messaging system provides a means to post a message on the Vedo-Vedi server and send an email notification and key to the intended recipient of the Vedo-Vedi message. If one includes the server in this discussion, then we should also mention the existence of the message database and CGI scripts for accessing it as parts of the messaging system.

2.2 Script Editing

Most of a user's time interacting with Vedo-Vedi is spent writing scripts. The script editor supports this activity with a visual display of the script, a tool bar of command icons, and a hierarhical menu of icons for creating script elements. Figure 1 shows how the program appears in a typical script-editing session.

The script is displayed in a panel as a sequence of rectangular boxes, laid out left-to-right and top-to-bottom (like English text). Each box represents a

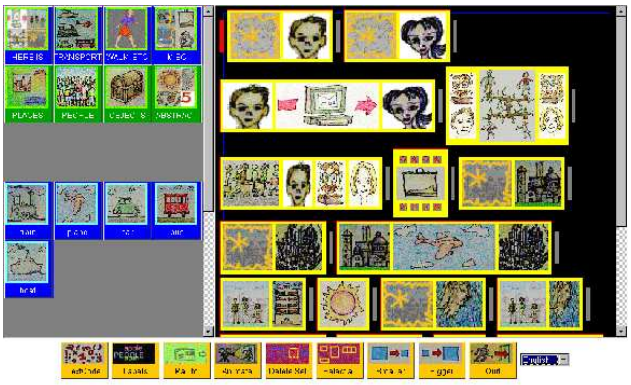

Figure 1: Screen shot of Vedo-Vedi in script editing mode.

Vedo-Vedi frame, and each frame, together with any objects it might contain, represents a sentence. Let's call the boxes that represent frames by the same name, "frame." Between each pair of frames in the script is a small vertical bar which can be selected in order to set the insertion point. When the user creates a new frame instance by clicking on a frame icon from the hierarhical menu, the new frame is placed at the insertion point, and then the insertion point is updated to follow the new frame.

The box representing a frame is divided into several regions: the frame canvas, zero or more slots (each of which is a rectangle), and the frame border (which consists of all the remaining area). Each frame has a frame image, which identifies the type of the frame, and this image is displayed on the frame canvas. The frame canvas is usually the largest part of the frame in terms of area. Each slot of the frame has its own rectangular area which contains a slot canvas (a slightly smaller rectangular area) and a slot border. Frames and slots can be selected for editing. A single frame can be selected by clicking on its border. A range of frames can be selected by first selecting a single frame to indicate one end of the sequence and then shift-clicking on another frame to indicate the other end of the sequence. A slot can be selected by clicking anywhere on it. Slots and frames are treated as different kinds of objects, and so they have separate selection states. It is possible to have a frame selected at the same time that a slot is selected in the same or another frame. However, clicking on the script panel background cancels any current selections whether of frames or slots.

Clicking on the frame canvas of a frame indicates a request for additional information or editing capability for the frame. For most types of frames, this causes a translation into the currently selected natural language to appear for the frame. For definition frames, as will be explained later, this opens up the definition for editing.

The meaning of a frame does not depend on its size. Therefore, frames can be resized within the script panel at the user's convenience. More detail can

be seen in images when frames are enlarged, whereas more frames can be seen at one time when frames are made smaller.

The menu of frame and slot-filler icons is arranged in a hierarchy. The root of the hierarchy is not seen by the user and has value only to the system. The next level consists of eight nodes, each represented by an icon. These icons are permanently displayed at the top of the menu panel. They are divided into two groups of four. The first four are used to select submenus for frames, whereas each of the second four brings up a submenu of slot-filler object types. When a submenu is selected, its icons appear below the icons of the permanent level. In a submenu, an icon may correspond to a leaf node, or it may stand for an even lower-level submenu. If it's a leaf, clicking the icon either inserts a new frame instance into the script or indicates what object to use to fill a particular slot in an existing frame.

2.3 Post-Office Support

In order to send a Vedo-Vedi message, assuming its script has been created, the user clicks on the MailTo button, fills in the recipient's email address, a return email address, and then the user clicks on the Send button. The program then generates a unique key for the message. Assuming the user has an Internet connection open, the script is posted on the Vedo-Vedi server using an HTTP POST operation. A database on the server stores the key-message pair in an associative array. The program also sends an email message to the recipient which not only serves to notify the recipient that a message has arrived at the server but also to provide a URL or key for retrieving the message. If picked up with the URL, the Vedo-Vedi applet is started. If picked up using the Vedo-Vedi application and the key, then Vedo-Vedi is not run as an applet.

Running Vedo-Vedi as an application is usually preferable to running it as an applet, because when run as an applet all class files and image files must be downloaded from the server, a process which can take ten or more minutes with typical bandwidth available. On the other hand, if run as an application, the class files and image files are simply loaded from the local disk of the user's machine, taking only a few seconds.

3 Definition Mechanism

Vedo-Vedi's primary means by which the user can extend the language is the object definition frame. There are three varieties of these in version 1.0: new place frame, new person frame, and new portable object frame. Here we describe how the object definition frames work.

3.1 New Object Frames

Object definition frames (or new object frames) are used to extend the set of objects that can be used in Vedo-Vedi scripts. Each new object frame introduces a single new object. It specifies the following for the new object: (1) an icon, (2) a more detailed image, (3) a submenu within the menu hierarchy, and (4) a textual name. The submenu is generally one of "places," "people," or "portable objects." However, it can be a submenu of one of these.

New objects are defined using frames for these reasons: (a) so that the definitions are part of a script, and (b) so that frames consistently handle all the needs for expression within the language. Thus we have followed the philosophy that definitions are part of the script and the script is the document that gets authored, not a meta-script.

A new-object frame has an appearance in the script like other frames, and its instance variables are slot fillers, too. Not all of the slot fillers are shown visually, however. For example, the textual label for the object is not shown on the frame, although it does appear in the expanded view and on the icon button in the menu. In Figure 2 there appear a new-place frame, a new-person frame, and a new-portable-object frame.

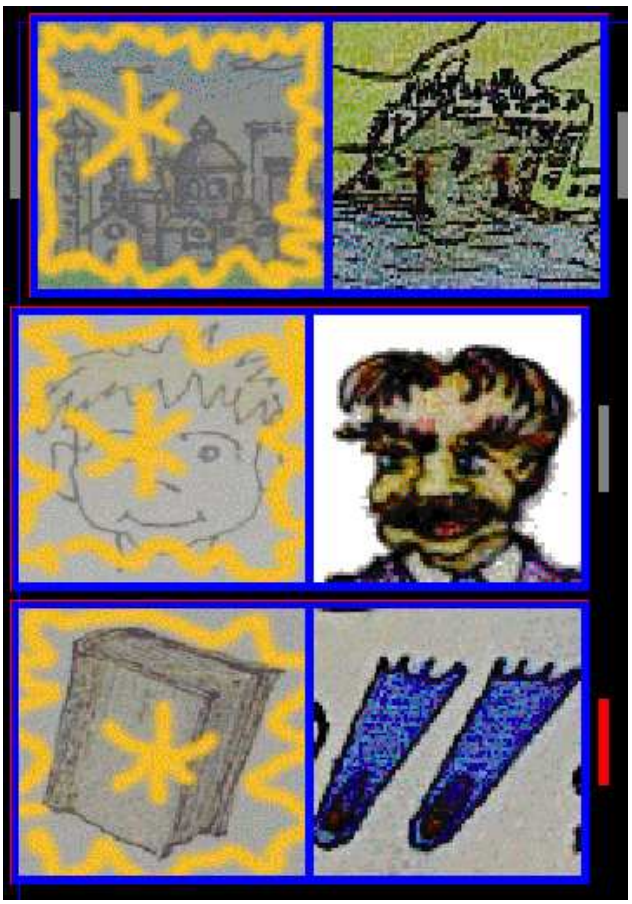

Figure 2: Examples of the three kinds of new-object frames.

The new-object frames are identified with an image consisting of a monochrome (gray) background image and a freehand yellow overlay that encircles the image and also marks it with a stylized supernova. The background image is either of a place (the city image), a person (the boy image) or a thing (the suitcase image), depending upon whether the new-object frame intro-

duces a new place, person, or portable object. Thus the image for a new-object frame indicates the type of object being defined together and also indicates that the frame is a new-object frame.

3.2 Examples

The new-object frames shown in Figure 2 have expanded views that give the composer of the script the means to specify the details of the definitions. The expanded view for a new-place frame starts out with a display of the "Place World" image. This image contains a rendering of a variety of geographical types, including mountains, sea, islands, cities, rivers, fields, etc. This can be seen in Figure 3.

Corresponding expanded views for a new-person frame and a new-portable-object frame are given in Figures 4 and 5.

Figure 3: An expanded view of the new-place frame with a definition of "Genoa."

Figure 4: An expanded view of the new-person frame with a definition of "Giovanni."

In each of these expanded views there is a textfield in which the label for the object is given. There is also a red rectangle overlaid on the base image that indicates what portion of the base image is to be used for the defined object. The selected region is used as the animation image for the defined object, and a 32

Figure 5: An expanded view of the new-portable-object frame with a definition of "fins."

by 32 scaled version of the region is used as the icon for the object. This icon appears in the hierarchical menu after the definition has been executed.

The program includes two features by which new images can be used in the definitions. One of these is access to the "gallery." The gallery consists of an iconic index to all (or nearly all) of the images used in Vedo-Vedi. By clicking on the gallery button and then on the icon for the desired image, the base image for the definition can be chosen instead of being assigned by default. The second feature is a provision for the user to type in a filename that contains an image to be used for the defined object. If the program is being run as an application, then Java permits this file to be anywhere on the user's hard disk. However, there is no provision at present for this image data to be automatically posted on the server when the message is mailed to someone, so images from files are useful only in non-mail demonstrations at this time. If the image resides on the server in the default directory, and the program is being run as an applet, then the image can not only be used by the sender, but the receiver should be able to see the image in the message that is received as well.

The animation for a new-place frame consists of a presentation of the new-place image followed by a transition into the defined place's image. Stills before and after the transition are shown in Figures 6 and 7.

3.3 Limitations on Icon Design and Importation of Images

We would like it if the users of Vedo-Vedi could have more options and tools for extending the system with their own images. For example, when Vedo-Vedi is run as an applet, the Java security provisions prevent the user from accessing an image file residing locally on the hard disk. Also at this time, there is no practical way to cut an image to the clipboard in a paint program such as Adobe Photoshop and subsequently paste it into Vedo-Vedi. Perhaps we could implement a means to convert an image file into ASCII text and then permit the user to paste it into a TextArea and have Vedo-Vedo convert it into a GIF or JPEG image,

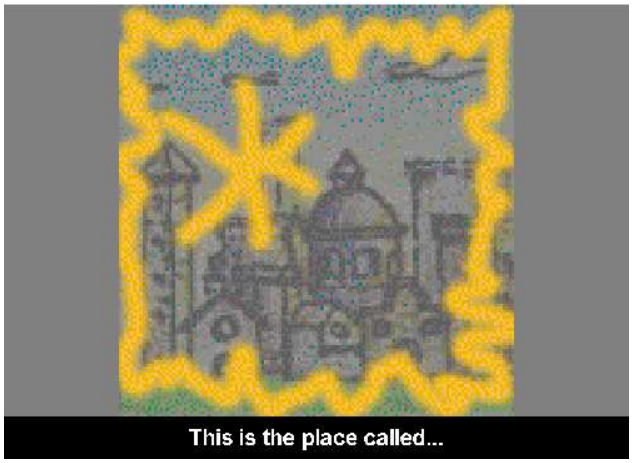

Figure 6: Screen shot at the beginning of the animation of the new-place frame.

Figure 7: Screen shot at the end of the animation of the new-place frame for "Rome."

but we have not done so. We are hoping that a subsequent release of Java or a Java toolkit will include a means for pasting images from the clipboard into a Java applet.

Were we to provide a means to paste images into Vedo-Vedi, we would also need a mechanism by which any custom images in a message would be posted on the server when the message itself is posted. Then there would need to be a corresponding mechanism by which the applet started by the recipient of the message could access these custom images and delete them from the server when used. Because images can take large amounts of disk space, there is a set of questions that would need to be answered regarding disk quotas, expiration dates, and policies about spatial resolution.

The present primary mechanism for defining new images and icons, which consists of the above-described means for selecting arbitrary rectangles out of any of the images used in Vedo-Vedi, has a disadvantage with its current implementation: if the base image for a new image has not already been downloaded from the server, then it must be downloaded in its entirety, even if only a small portion of it is used in the new image. This is obviously true during the act of defining the image, because the user needs to see the base image in order to select a portion of it. At this time, it is also true when the recipient views a message containing such a definition, because the server software is not capable at this time of responding to a request for only a portion of an image. Because each of the standard images for new definitions provides a variety of imagery, it is large. Downloading these images in their entireties is consequently time consuming.

3.4 Hidden Scripts

Early user testing revealed that although users appreciate the ability to define their own places, people, and portable objects using new-object frames, they did not want the new-object frames to appear either in the script or the animation of the script. Consequently, they deleted the new-object frames from their scripts once the definitions had been executed. This posed two problems: (1) the definitions would not be transmitted to the recipient as part of the message, and so the references within the message to the new objects would be uninterpretable at the receiving end, and (2) the sender, no longer having any handle for the definition, could not reopen the expanded view in order to edit the image or to adjust the label for the new object.

We solved this problem by creating an additional script for each message called the "hidden script." Whenever a new-object frame is instantiated, it is entered into both the regular script and the hidden script. If the user deletes the frame from the main script, it remains on the hidden script. Since the animation is based on the main script and not the hidden script, the animation need not include any presentation of the introduction of new objects. However, if the user needs to edit the definition of an object after its defining frame has been deleted from the main script, it is possible to call up the hidden script for editing and

find the frame there. (If the user deletes the definition frame from both the main and hidden scripts, then the definition frame is really gone.) When the message is sent to the server for posting and eventual pickup by the recipient, both the hidden and main scripts are sent. That way, the definitions for new objects can be delivered to the recipient's copy of Vedo-Vedi even if the definitions have been deleted from the main script.

In addition to new-object frames, the hidden script may contain certain other kinds of frames that are used to establish background knowledge for a message. For example, instances of INSIDE frames are placed in the hidden script; an INSIDE frame expresses a containment relationship between a pair of places: "Rome is in Italy." These obvious facts may be distracting in a message, but they can be useful if and when the program needs to make inferences about whether the travelling group is in a particular place or not.

3.5 Textcodes of New Objects

The system manages references to new objects internally using textual codes generated for each object. These codes called textcodes belong to a name space that can potentially have conflicts. At present, little effort has been made to avoid conflicts that might happen across multiple conversational threads. However, it would not be difficult to provide means for the reduction or elimination of these conflicts in a variety of situations.

One obvious means by which to reduce the likelihood of name conflicts is through the use of longer textcode symbols, perhaps containing randomly generated substrings. Generated symbols might be prefixed with the name of the sender and the date the definition is executed, for example.

Currently, name space conflict handling is restricted to avoidance of old text codes as new objects are defined. However, a substitution mechanism could be used to make sure that two sets of user-defined extensions to Vedo-Vedi use non-conflicting textcodes.

A more ambitions means to manage the growth of extentions to the language is to design a large catalog of preselected names and categories, encourage users to select names from this catalog when they create new objects, maintain a record of all these definitions, and administer a knowledge base (linked to the catalog) consisting of validated definitions. Such an effort has been beyond the scope of our project.

4 Other Extensibility Issues

4.1 Performance

Our experience designing Vedo-Vedi has underscored the existence of a tradeoff between extensibility and network delays in image-oriented visual languages. As long as both the sender and the recipient are working with the same database of images stored locally on their machines, extensions based on those images can be communicated rapidly. However, if one user extends the database of images by incorporating an entirely new image with lots of pixels, then a substantial bulk of new data must be delivered to the recipient. When bandwidth is limited, this delay can be not only annoying but an impediment to the adoption of the language.

4.2 Image Transformations

Even without provisions for letting users import their own scanned images, more could be provided to help users make new icons and images for their extensions to the language. At present, users can select rectangular subsets of existing images to use for their new objects. It would be nice if they could also transform the images with color changes (e.g., negation, tinting and shading), composition of two or more selected subimages, and the superimposition of geometric shapes. An objective of such a mechanism for deriving new images from old is not only the provision of greater variety in the set of possible resulting images but also a means to constrain the lengths of the representations of these images to amounts that can be efficiently transmitted through the Internet and posted on the server without requiring vast amounts of storage. The current representation of a new image in Vedo-Vedi is very efficient, consisting of an abbreviation of the name of an existing image and a short string giving the coordinates of the selected rectangular subset.

4.3 Levels of Image Derivation

In the current implementation of Vedo-Vedi there is an interesting problem with the use of the gallery when a user-defined image is selected as a base image for a new definition. In this case, one definition depends on another, and the user may set up this dependence inadvertently. If the user were to delete the first definition, then the recipient's copy of Vedo-Vedi would not be able to interpret the second definition. Since any image obtained as a rectangular subset of a rectangular subset of an original image could be obtained directly as a rectangular subset of the original, there is relatively little expressive value in the system's permitting this double derivation now, and we could forbid it in a future version by including in the gallery only original images. Alternatively, we could permit the user to define images indirectly but compile them into direct derivations from the originals. In a richer image-creation environment including color transformations and geometric primitives, the expressive benefits of indirect definition would be more valuable to users, and compilation into direct references could still be accomplished.

4.4 Resolution Problems

Another lesson we have learned developing Vedo-Vedi is that our method for new image derivation suffers from a resolution problem. The "Place World" image was designed to provide a wide variety of geographical imagery. Consequently the image has substantial detail. Downloading this image, even after JPEG compression causes some of the noticeable delay in working with Vedo-Vedi place definitions. Then, when the user has defined a new place by selecting a subimage of Place World, the new image tends to be fairly small. When used as an icon, the new image works very well. However, when used in a full-screen animation, the new image is usually very blocky and unattractive. Were we to double the linear resolution

of the Place World image, download times would go up by a factor of four. Is this worth it to improve the appearance of the user's image during an animation? The answer probably is that it depends. It depends on the Internet bandwidth available to the sender and to the receiver. It also depends upon how much each of them cares about resolution. The images are somewhat symbolic already – that is necessarily the case when the user must select imagery from a constrained database to match any new concept s/he may have. The blockiness can help communicate the fact that the representation is in fact an approximation, colorful as it may be.

Ideally, the Vedo-Vedi system would provide a combination of options and smart mechanisms for the management of image resolution and transmission times. It would make maximal use of high resolution imagery already in the possession of both sender and receiver. It would dynamically select resolution on a per-image/per-session or even per-subimage/per-subsession basis to best meet the preferences of both parties to the communication given the currently available bandwidth and storage space. Progressive transmission methods would also be used. A specific enhancement we are considering to Vedo-Vedi is a means to avoid the downloading of any large images other than the icon base when bandwidth is at a premium. The system already uses a lazy downloading policy, only downloading images specifically needed for the current script or animation. However, some of these images are large, and the display of the script and the animation could often be accomplished without them, at the expense of the user coping with low resolution, of course.

5 Summary and Conclusions

Extensibility is important in visual languages for human-to-human communication because people want to share thoughts and experiences about lots of new topics, people, places, and objects. Vedo-Vedi includes a useful mechanism for defining new objects and the new icons to represent them. It provides for the hiding of definitions and transmitting them with messages. It manages the placing of new icons on submenus. Vedo-Vedi does not yet support user definition of new types of frames, extensions to the ontology, to the animation engine or to the messaging model.

Our experience with Vedo-Vedi has underscored the existence of a tradeoff between extensibility of imagery and transmission delays. More research is needed on the subject of how best to make maximum use of pixels shared by sender and receiver.

Acknowledgments

The authors would like to thank Stefano Levialdi, Luigi Cinque and Paolo Bottoni of the Pictorial Computing Laboratory, Rome, Adam Carlson of the University of Washington, Genny Tortora of the University of Salerno, Margaret Burnett of Oregon State University, Allen Ambler of Kansas State University, and S.K. Chang and Robert Korfage of the University of Pittsburgh for their help or encouragement on this project.

References

[1] Beardon, C. 1993. Computer based iconic communication. In Ryan, K., and Sutcliffe, R. (eds.) *AI and Cognitive Science '92*, London: Springer-Verlag, pp.263-276.

[2] Repenning, A. 1994. Bending icons: Syntactic and semantic transformations of icons. *Proc. VL'94*, held at St. Louis, MO, Oct. 4-7. pp.296-303.

[3] Tanimoto, S. L. 1997. Representation and learnability in visual languages for web-based interpersonal communication. *Proc. VL'97*, held at Capri, Italy, Sept. 23-26, pp.2-10.

[4] Van Reeth, F., and Flerackers, E., 1990. Visual programming in a computer animation environment. *Proc. VL'90*, held at Skokie, IL, Oct. 4-6, pp.194-199.

Appendix

The applet version of Vedo-Vedi can be found at the following URL.

`http://trillium.cs.washington.edu:8080/tanimoto/vv/applet/run-vvap.html`

It can take several minutes to initialize version 1.0 of the applet, due to the volume of image data and class files that must be downloaded.