HEWLETT PACKARD

Prototyping a Collaborative CAD System by Taking Advantage of Software Reuse and a Software Bus Framework

Mark A. Gisi, Cristiano Sacchi*
Software Technology Laboratory
HPL-93-27
April, 1993

software reuse,
software bus, CAD,
BART,
collaboration, rapid
prototyping

BART, a software bus, was designed to provide flexibility in the development of software systems by promoting component independence. This framework allows components to be constructed independent of the context in which they are to be used, thereby allowing them to be reused in many different situations. Our experience using BART to prototype a system that supports group collaboration between people residing in geographically different places demonstrated the feasibility of this approach. We were able to connect a number of existing components with little effort in a surprisingly short time. In this paper, we discuss the role BART played in supporting the integration of software components and in enabling us to rapidly prototype a shared-space collaboration system.

Internal Accession Date Only
To be published as "*A Positive Experience with Software Reuse Supported by a Software Bus Framework*" in the Proceedings of the *Second International Workshop,* March, 1993
*CAD Group, National Research Center, Milan, Italy

Prototyping a Collaborative CAD System by Taking Advantage of Software Reuse and a Software Bus Framework

Mark A. Gisi

Software Technology Laboratory
Hewlett-Packard Laboratories
Palo Alto, California, USA
gisi@hplabs.hp.com

Cristiano Sacchi

CAD Group
CNR-IMU
Milano, Italy
cris@cad.imu.mi.cnr.it

Abstract

BART, *a software bus, was designed to provide flexibility in the development of software systems by promoting component independence. This framework allows components to be constructed independent of the context in which they are to be used, thereby allowing them to be reused in many different situations. Our experience using* BART *to prototype a system that supports group collaboration between people residing in geographically different places demonstrated the feasibility of this approach. We were able to connect a number of existing components with little effort in a surprisingly short time. In this paper, we discuss the role* BART *played in supporting the integration of software components and in enabling us to rapidly prototype a shared-space collaboration system.*

1 Introduction

We set out to prototype a computer system that would enable a group of people to interact in a common space, yet be located in geographically different places. One example of a popular model that supports collaborative work between distributed people is the shared whiteboard[6, 7]. It allows people located in different places to interact (via teleconferencing) by writing to and reading from a shared whiteboard. This paradigm is based on the notion of WYSIWIS (What You See Is What I See), which requires that people see the same semantic information about the world and share the same view of the world.
A major limitation of the WYSIWIS paradigm is that it does not enable multiple users to customize their local view of the world while sharing the same semantic information. In order to address this limitation we decided to develop a shared-space collaboration model that enabled engineers not only to collaborate over a design while located in geographically different places, but also to modify their local view of the world. Therefore we prototyped a simple CAD (computer-aided design) system that supported these requirements.
Initially, we did selected BART, a software bus, not to facilitate the software construction process, but to provide a platform for propagating semantic design information among distributed users. We were surprised by how much of the prototype we were able to construct in an unusually short time. The effectiveness of BART's approach to software component integration was key to our success.

BART was designed to provide flexibility in software systems by supporting component independence[1, 2]. This framework allows components to be constructed independent of the context in which they are to be used, thereby allowing them to be reused in many different situations. We were able to demonstrate this flexibility by prototyping a shared-space collaboration paradigm in which we took a number of existing components varying in size and connected them with very little effort. In this paper we describe this experience.

In section 2 we provide a scenario of a real-world problem that motivates the rationale behind the shared-space collaboration paradigm. Section 3 provides an overview of BART. Section 4 presents the design of the system we prototyped to support a multi-user CAD system and discusses the role played by BART. In section 5 we describe our process and look at why software reuse was key to our success. In section 6 we briefly discuss BART's potential role in supporting general-purpose and domain-specific reuse. In the final section we summarize our experience.

2 A Shared-Space Collaboration Scenario

We begin by providing the reader with a scenario of a real-world problem that motivates the rationale behind the shared-space collaboration paradigm.

Imagine that you are an automobile designer, working in the United States, who has been up all night trying to finish the design of a new sports car. It's four o'clock in he morning and the design is due on your manager's desk in five hours. You are nearly done, except for the doors. The design specification states that the doors must open upward, rather than outward in the conventional manner. You have never designed such doors and are not sure how the hinges should connect to the body. Frustrated, you staring into your CAD display when you remember that you have a colleague, Pino, who has designed doors like these before (he once worked for the Delorean Car Co.; now he works for Fiat in Italy). Luckily, it's 11:00 A.M. in Italy and he is at work. You call him and explain your problem.

You are also fortunate because you are running a state-of-the-art CAD system that enables two or more people to view the same 3D object (e.g., a car design) in different locations. Not only can you share the object, but both of you can modify the car design and see each other's modifications simultaneously. Furthermore, you can both see the same semantic information that describes the object, but each person can choose to view the object from a different perspective.

While talking to you over the phone, Pino brings up your car design on his screen. The first thing he tells you is that it is traditional to design a new sports car in red (not blue, the color initially chosen). He changes the color and now you are both looking at a red sports car without doors. Pino then goes on to explain that the best way to conceptualize adding doors that open upward is to rotate the car body so that the doors appear to swing open in the conventional manner (out to the side). Therefore he rotates the image of the car 90 degrees in his display, as illustrated in figure 1. He then proceeds to add doors to the design.

Note that Pino was able to change the color of the car body such that it updated both views, yet he was able to rotate the car design in his display without affecting the car design in your display. In this scenario, color is a semantic feature of the car design. If one person changes the semantics of an object that is viewed by n people, then all n people receive the semantic update, yet they can display that semantic information however they please. In this case, Pino decided to rotate his image, but when he added the doors, they appeared in both displays. Now you are done!

3 An Overview of Bart

In this section we provide a very brief overview of BART's architecture. A more detailed discussion of the architecture and its design rationale appear elsewhere[1, 2, 3].

Many techniques have been developed to support interaction between software components. These techniques can be placed into two broad categories: *control signals* and *data sharing*. A common example of a control signal is a remote procedure call (RPC). It permits one process (i.e., component) to make a procedure call to another process that happens to be remote, passing control information and data.

Although data sharing is often required between interacting components, few platforms support it elegantly. Because of this insufficient support, applications implement data sharing in an ad hoc fashion. For example, one process may be designated as a server that maintains the data to be shared by multiple client processes, which access the data by polling the server. Often it is more desirable to have the server provide notification to the clients when data changes or new data is added. In both cases, two problems arise. First, code complexity is increased for each component, since the components must perform a considerable amount of bookkeeping to know what data is available and where to find it. Second, dependencies between components are increased, thereby reducing flexibility, because components need to know about each other's existence. These problems also exist with the traditional RPC approach described in the paragraph above.

In order to overcome these limitations and provide a greater degree of component independence, BART

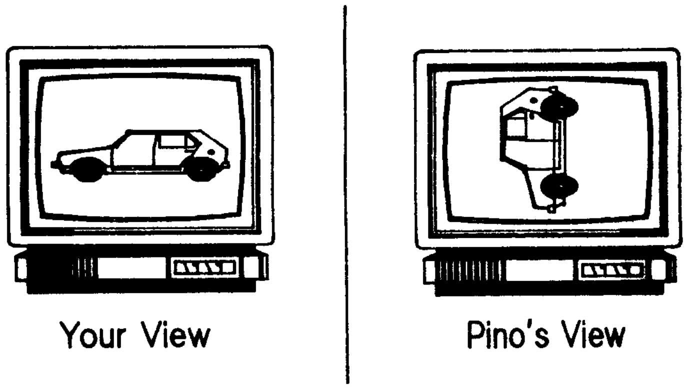

Figure 1: Your view and Pino's view of the world.

provides two different abstractions: *control ports* that support control signals, and *data ports* that support data sharing.

The fundamental idea that underlies these two abstractions, and is vital to achieving component independence, is that they support *anonymous* interaction between components. That is, components can interact with each other without needing to explicitly identify themselves. This increases the flexibility of components because the developer does not need to know, a priori, the channels of communication that might occur. Hence, components can be developed independent of the context in which they may be used.

3.1 Control Ports: Support for Control Integration

BART control ports support control integration between software components. Their design is based on the publish/subscribe metaphor in which a component can broadcast (publish) a message to a predeclared control port. All components that have expressed interest in that port (subscribed) will receive a copy of that message. Messages can be sent either synchronously or asynchronously. Control ports support control signaling because they are frequently used to signal the occurrence of an event (i.e., provide notification) or make a request to a common server. For example, consider an editor that subscribes to a port that handles requests to edit files. It is possible for a compiler to send a request to that port so that the editor could automatically display a file in which the compiler had detected compilation errors. Note that component independence is achieved because the identity of the editor being used is not visible to the compiler (e.g., it could be emacs or vi). Therefore, the editor could be added or replaced without affecting the interaction with other components.

The publish/subscribe metaphor, on which BART control ports are based, was proposed by Steve Reiss in his Field programming environment described in [4]. His initial proposal had the limitation that broadcasted messages had little structure imposed on them (they were passed as ASCII character strings). This limited the connection of fine-grained components. In order to address this problem, the designer of BART extended this paradigm by adding type structure to these messages.

3.2 Data Ports: Support for Data Integration

BART data ports support data integration between software components. Data ports allow components to export data to the world. This data can then be imported by other components, resulting in data sharing. Data ports differ from control ports in that data on a data port is persistent, i.e., they exist as long as the exporting component chooses. Furthermore, it can be modified by the exporter. In this case, all components that have expressed interest in a piece of data that is later modified will automatically receive the update. In some sense data ports follow the publish/subscribe metaphor also. Control ports differ in that a component subscribing to a control port receives all messages sent to that port starting from the time it subscribed, but all messages sent to that control port prior to subscription are lost. In the case of a data port, all data currently existing on the port at the time a component subscribes to the port is made available to that component, as are any updates performed after the component subscribes.

Consider the collaborative CAD users scenario presented in section 2. The car design is a semantic data object that could persist on a data port. All CAD systems (each system being a single component) subscribe to that port. When a semantic change is made to the car design (e.g., changing the color of the car to red)i, the data port is updated and all subscribers receive that update. Furthermore, any system that subscribes to that data port in the middle of a design session will automatically receive the most up-to-date version of the design (e.g., Pino received the design as it existed at 4 A.M.).

4 The Initial Design

In this section we give an overview of the design of our collaborative CAD system prototype. We would like to build a system that permits an arbitrary number of users to participate in a car design session. Before we present the infrastructure that supports collaboration, we first present the design of a simple, single-user CAD system and describe later how we extended it to support multiple users.

4.1 A Single-User System

A single-user CAD system was constructed from the following components:

- ACIS geometric modeler, which is used to compute 3D solid objects. This is the core geometric modeler found in Hewlett-Packard's CAD software products.
- Hewlett-Packard's commercial graphics library, used to display 3D solids.
- An INTERVIEWS button box. INTERVIEWS is a C++ library used for generating user interfaces [5].
- A simple command-line editor that enables the user to enter design modification commands (e.g., delete cylinder or unite torus and cone).

Some of these components are illustrated in figure 2. The single-user system behaves as follows: using the command-line editor, a user enters commands that modify the semantic object. Each entered command represents a change in the semantics of the object and must be forwarded to the geometric modeler. The geometric modeler then performs this design change and redraws it in an X window using routines from the graphics library.

Recall that each user can also change their local view of the object (rotate the image, zoom in or out, move the image up, down, left, right, etc.). These changes do not affect the semantics of the object and are entered via a button box, built with the INTERVIEWS graphics library (actually the button box was borrowed from a previous application and required only minor modifications such as button label changes). The command-line editor and the button box were connected to the geometric modeler using control ports.

All the components and libraries existed prior to the start of our experiment and, in some sense, can be considered off-the-shelf components. They were all written in C++ and vary considerably in size. The geometric modeler is the largest component, at about 300,000 lines of code, and the command-line editor is the smallest with only a couple of hundred lines of code.

4.2 Support for Collaboration

To support group collaboration, each single-user system is treated as a single component. An arbitrary number of these can then be connected using a control port and a data port.

We chose a decentralized design in which users run a local copy of the stand-alone single-user system at their sites. As before, they enter requests to modify the design through the command-editor. The difference now is that the request must be broadcast to all users participating in the design session, so that they can perform that operation on their local copy. This ensures that all sites have a consistent representation of the car design.

The decentralized design requires that only minor changes be made to the single-user system. For one, we need to provide access to a common control port that will be used to broadcast all user requests. Pre-

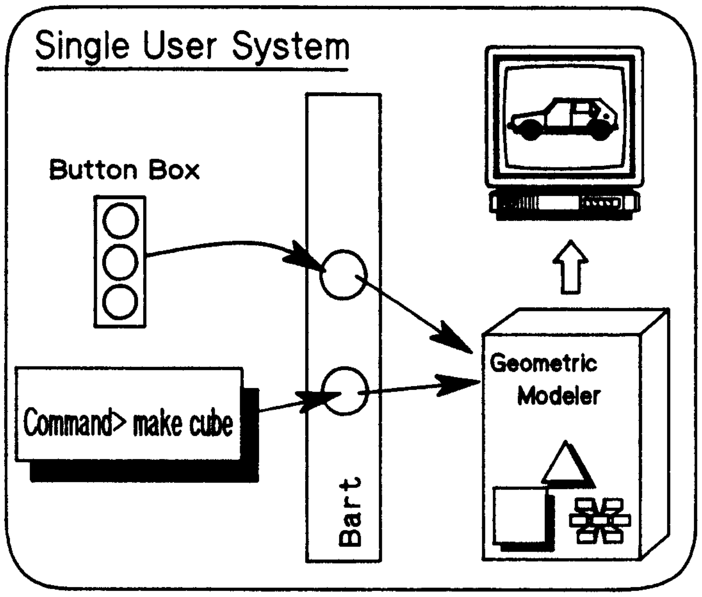

Figure 2: Some of the components of a single-user CAD system.

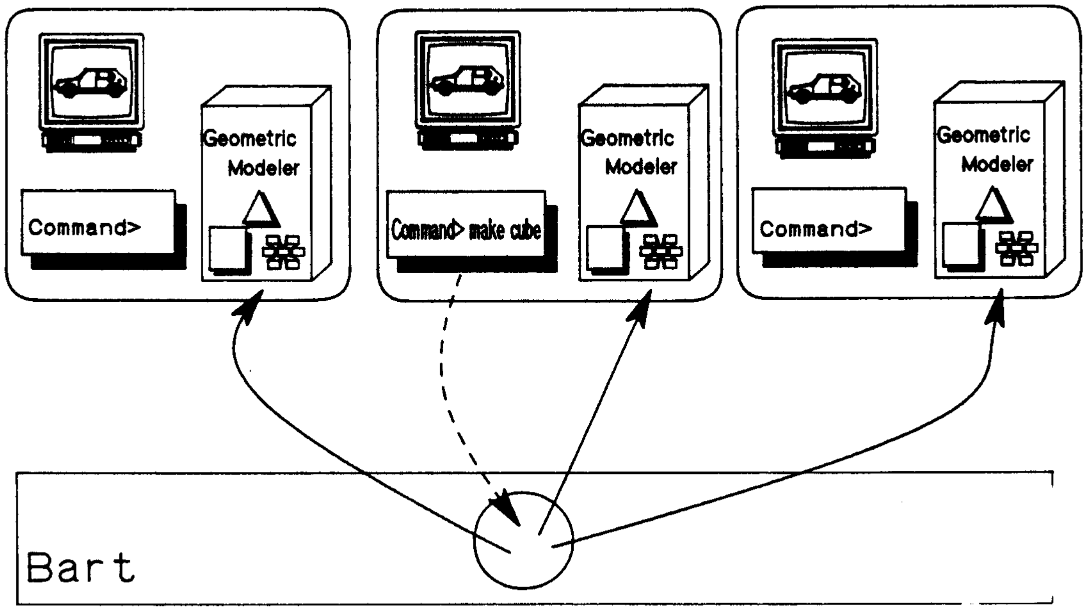

Figure 3: Broadcasting a modification request. The circle represents a control port. The dotted line represents the sending of a request and the solid lines represent the broadcast.

viously, the command-line editor would send all modification requests directly to the geometric modeler. Now the editor also publishes it on a common control port. The geometric modeler of each system will listens (subscribes) to that port. When a request arrives, it executes the request as before. This scenario is illustrated in figure 3.

Another nice consequence of using the bus is that it preserves serializability among modification requests. That is, the operations performed on the geometric modeler are not commutative; therefore, the order in which they are executed is important. For example, if two users receive (and execute) two modification operations in reverse order, it is possible that their local versions of the design could become inconsistent. BART preserves serializibility because if two users are competing to place their modification requests on the bus, one will get the bus's attention first. Once BART receives a request, it guarantees that the broadcast will be made to all users prior to giving any attention to the next request.

4.3 Joining a Design Session in Progress

The last issue that needs to be addressed is enabling a user to join a design session already under way. Essentially, what user needs to do is to obtain the most recent version of the design (as was the case for Pino). This is the purpose of the data port. The current car design is published on the data port. Whenever a user connects in the middle of a design session, he/she can initially subscribe to that data port, get the current version, unsubscribe, and join in the session.

5 Our Experience with Bart and the Reuse Process

We now describe our experience using BART and its support for the reuse process that enabled us to rapidly prototype the above design.

We were quite surprised when we realized how much we accomplished in so little time. In addition to having access to BART, we were fortunate for a number of other reasons. Not only did we know about the existence of the components that were used to construct the system, but also they were available to us, and we had considerable experience working with each of the components and libraries. Furthermore, all the components were written in C++, which we both knew. Finally, we were fortunate because our skills for the task were complementary. One of us had experience working with geometric modeling, while the other had experience with building distributed systems. The only overlapping skill was our knowledge of C++.

One day was spent working on the design and three days on the implementation. When we were done, we had a system that would allow an arbitrary number of people working on different machines (currently connected over a local area network) to send modification requests at will. We ignored a number of important issues such as security, reliability, and policies with respect to user privileges (e.g., preventing someone from entering a design session). The implementation consisted of approximately 500,000 lines of reused code and approximately 500 lines of customized code.

The first day of implementation was spent constructing the single-user CAD system. Our approach was to view the geometric modeler as the kernel and to add on the necessary user interface components that would enable a user to design 3D objects. After we connected the INTERVIEWS button box, the command-line editor and the X-window display, the single-user system was complete. This system was developed without giving any consideration to the group collaboration requirements.

The second and third days of implementation were spent providing support for collaboration. Given the new single-user component, three modifications were required. First, all semantic change requests entered by a user through the command-line editor had to be broadcast (published) to all the geometric modelers. Recall in the single-user system that a control port was used to connect the command editor to the geometric modeler. In the collaborative system, we used one common control port to connect all the command-line editors to all the geometric modelers (see figure 3). One of the nice consequences of the single-user system design was that the command-line editor and geometric modeler shared information without identifying one another. This made connecting an arbitrary number of command-line editors and geometric modelers trivial. Very little source code modification was required because each command editor and geometric modeler still saw the same control port. This is precisely the kind of component independence the BART framework promotes.

The second modification required that we prevent changes that users made to their local views from being broadcast to other users. Recall that the button box enabled users to enter requests to modify their local view of the design (e.g., zoom in and rotate). Since it is used to control the user's local view, there is no need to broadcast these commands. Initially, we connected the button box to the geometric modeler using a control port for convenience. We moved the button box code into the geometric modeler module and removed the control port (which essentially required that a remote procedure call be converted to a local procedure call).

The third and final modification required that we enable a designer to join in the middle of a design session. This could be achieved by using a data port that would enable the new user to acquire the current version of the design. Here is where we encountered some difficulty. In theory, the BART framework advocates sharing of data via data ports. The problem lay not with this model, but with the implementation. At the time, BART only supported storing simple data types on data ports. It did not support the sharing of large data items. In our case, we needed to declare a data port that could maintain an entire design image. We provided feedback to the designer of BART who addressed this problem in a later version.

We were able to surmount this problem by simulating the data port. We accomplished this by including an extra CAD system that executed whenever a collaborative design session initiated. Just like any other single-user system, it would wait and listen to the control port for any semantic design modification requests sent out by users. When a user request was received by this component, it was executed on the local version of the design. The only difference was that it also listened for session connection requests from users trying to join in and then passed the current version to the new user. This simulated data port took only a couple of hours to implement.

6 Supporting General Purpose and Domain-Specific Reuse

Based on our preliminary experience with BART, we believe this framework can support the reuse process from two different dimensions, ***horizontal*** and *vertical* reuse. Horizontal reuse means that an application is built by taking existing components from different domains and connecting them together using a BART-like framework. It is horizontal because one is free to select the components from across different domains. We essentially performed horizontal reuse.

Vertical reuse means that one generates an application by selecting components from within a well-defined domain and connects them using a BART-like framework. It is vertical because components are selected from within (or up and down) a specific domain. This is sometimes referred to as domain-specific reuse.

We believe these two perspectives are very complementary, and BART could play a vital role in pursuing both. One may initially attempt to rapidly prototype an application by performing horizontal reuse to demonstrate the feasibility of an idea. For example, consider the prototype discussed in this paper.

After receiving positive feedback on the prototype, one may choose to further develop the application by performing vertical reuse. That is, develop components that are more suited to the application, yet still provide some degree of configurableness. This is sometimes referred to as a domain-specific kit[2].

If we were to pursue vertical reuse, the next logical step might be to develop domain specific components that would allow one to build configurable collaborative CAD systems. That is, from this domain one might build different systems that support different groups of engineers by selecting different interfaces. For instance, one might configure a system to support a group of civil engineers, another to support a group of mechanical engineers, and yet another to support a group of automobile designers. Another way to configure a collaborative system would be to allow for interchangeable geometric modelers, since different modelers provide different functionality (e.g., they may support different types of geometric curves and surfaces).

We have had some preliminary success using Bart to support horizontal reuse. We are currently working within a research group that is experimenting with BART to determine how well it supports vertical reuse.

7 Summary

BART was designed to provide flexibility in software systems by supporting component independence. This framework allows components to be constructed independent of the context in which they are to be used, thereby allowing them to be reused in many different situations.

We were able to demonstrate the feasibility of this approach to software construction by prototyping the shared-space collaboration model. By connecting a number of existing software components using this software bus, we were able to obtain a considerably more flexible system. Because of this flexibility, we were able to connect an arbitrary number of single-user systems to obtain a system that supports group collaboration between designers residing in geographically different places.

The support provided by the BART framework for software construction will play a greater role in the reuse process as more and more well-developed software components become available.

8 Acknowledgments

We would like to thank Mike Creech and Brian Beach for providing feedback on drafts of this paper.

References

[1] Beach, B.W., "Connecting Software Components with Declarative Glue", *The 14th International Conference on Software Engineering*, May 1992.

[2] Beach, B.W., Griss, M.L., Wentzel, K.D., "Bus-Based Kits for Reusable Software", editor Selby, R.W., *Proceedings of the 2nd Irvine Software Symposium*, pp. 19-28, March 1992.

[3] Beach, B.W., "Software Glue, Connecting Reusable Software Components", PhD thesis, University of California, Santa Cruz, 1992.

[4] Reiss, S., "Connecting Tools Using Message Passing", *IEEE Software* 7(4):57-66, July 1990.

[5] Linton, M.A., Vlissides, J.M., Calder, P.R., "Composing User Interfaces With Interviews.", *IEEE Computer*, 22(2), February 1989.

[6] O'Connell, J., Edwards, N., Cole, R., "A Review of Four Distributed Infrastructures", Technical Report HPL-92-86, Hewlett-Packard Laboratories, Bristol, England, July 1992.

[7] Stefik, M., Froster, G., Bobrow, D.G., Kahn, K., Lanning, S., Suchman, L., "Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings", *Communication of the ACM*, 30(1), pp. 32-47, January 1987.