A Software Solution Broker for Technical Consultants

A distributed client-server system gives HP's worldwide technical consultants easy access to the latest HP and non-HP software products and tools for customer demonstrations and prototyping.

by Manny Yousefi, Adel Ghoneimy, and Wulf Rehder

On a typical working day an HP consultant, one of thousands worldwide, sits down with a customer to solve a business problem. The challenge, the customer may tell the consultant, is to move sales data to headquarters more quickly so that management can make timely strategic decisions. For a solution, the consultant might propose a decision support system that integrates the customer's older legacy system where the sales data has been stored traditionally with a faster "warehouse" database and easy access tools that present the information in just the form needed, right on the customer's desktop. "Let me show you what I mean," the consultant says, turning on a laptop computer (which had previously been connected to a LAN or telephone socket). Navigating through the windows on the screen, the consultant invites the customer to look through a virtual shelf filled with databases and access tools, all represented by icons, together with middleware and application development toolkits (see page 98). The consultant clicks on an icon and the tool becomes immediately available for browsing or for self-paced learning. From here the consultant may show one of the demos that are included, or navigate the customer through a hypertext document to more information, alternate products, additional options, and prefabricated software building blocks. No wonder that this virtual software laboratory is called by HP consultants, "the software sandbox." This consultant is actually building—from the tool and product portfolio in front of them—a prototypical decision support system for this customer. How much of this is fantasy and how much reality?

The answer is that it is all reality now. The software sandbox that the consultant was starting to "play in" is called the HP Software Solution Broker (or Broker, for short) and is available now to HP consultants. Defining and creating a decision support system is, of course, not play but serious work. However, the ease and immediacy of the Broker, the ample choices, and many helpful hints make even urgent business problem solving an experimental sport. Best of all, the consultant receives these products and tools, together with support and on-line documentation, free of charge. For this convenience, substantial research efforts had to be poured into building such a virtual software depot, using HP's own hardware platform and the most advanced object technology. Before explaining this implementation more systematically, it is useful to watch our technical consultant and the customer at work.

Using the Software Solution Broker

To get a feeling for how the Software Solution Broker is used we will briefly watch the technical consultant show the customer how to build a prototypical decision support system.

After clicking on the icon in the ORB control panel, which starts the object request broker (an action that in effect opens the lid covering the sandbox), the consultant activates the Software Solution Broker icon. Another window opens offering the Broker's classification of products, either by vendor, by technology, or by product name. (Alternative paths into the Software Solution Broker, such as a classification by business problem, are under development.) Choosing the **i**(nformation request) button for technology, the consultant asks whether the customer wants to see database information first or options for the user interface. As an executive, the customer is eager to see or build a nice GUI. Clicking on the graphic user interface **i** button brings up several choices of which three are shown in Fig. 1. Having heard about VisualWorks the customer selects it and is presented with the VisualWorks Showcase.

The consultant then shows a VisualWorks demonstration to explore with the customer what kind of data display windows, control buttons, analysis tools, and other features would be appropriate. After jotting down these initial requirements the consultant is ready to build a first prototype. The help button launches a palette of GUI building tools, and it takes only minutes to draw an example of a transaction entry tool for the transactions underlying the decision support system the customer wants built (see Fig. 2). Here the customer interrupts and requests that the data be shown in spreadsheet form as well as graphically. They agree on bar chart and pie chart presentations for a first cut and proceed to discuss the requirements for the underlying database. The Software Solution Broker has a "virtual shelf" of relational databases that work with VisualWorks, and among these the customer may have a favorite system, or an already installed legacy database. They again discuss the pros and cons while viewing various product demonstrations.

We meet the customer and the consultant again after another hour or so. By then the VisualWorks front-end tool displays some real data pulled from a database (Fig. 3). At this point we leave the executive's office and describe how the Software Solution Broker is constructed.

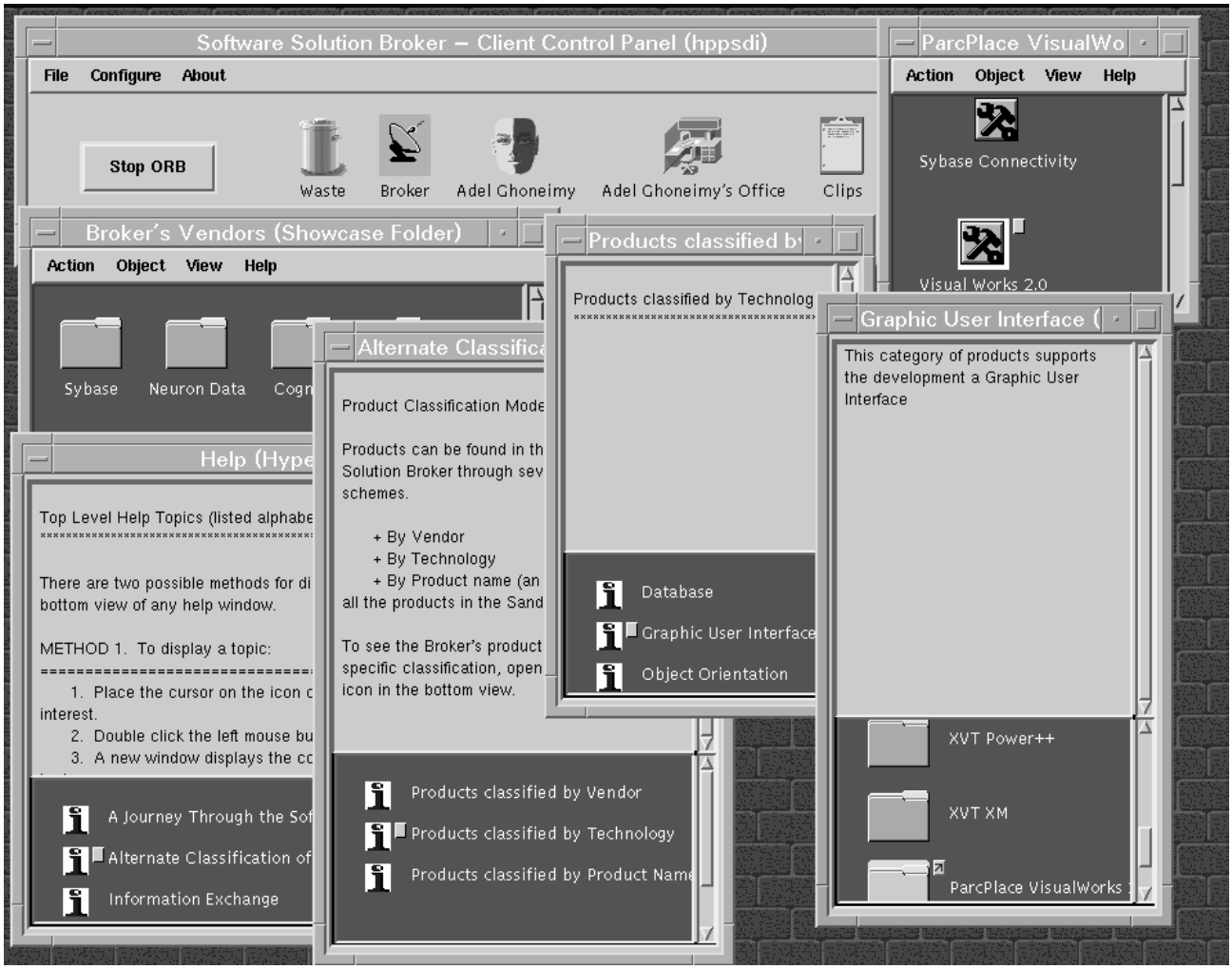

Fig. 1. Software Solution Broker user interface.

Constructing the Software Solution Broker

Two considerations determined the architecture and consequently the implementation of the Software Solution Broker. First, since the products on the Broker have to be accessible worldwide but will be updated and maintained locally, the global partitioning between distributed users and a central server functionality called for a client/server implementation on a wide area network (WAN). Secondly, the need to accommodate many different types of clients and to be able to encapsulate many different products in the software server strongly suggested vendor independence (openness) and adherence to certain industry standards such as the Common Object Request Broker Architecture (CORBA).

Software Substrate

Here we will not focus on the WAN implementation but instead will concentrate on the software substrate on which the Software Solution Broker is built. In the software substrate (see Fig. 4) we include the entire software kit composed of server and client development tools, tools for building the client/server interaction components of the system, and repository tools. Repository tools are essential for the construction of a depot that contains the information in the system, including the logic for accessing this information. After a careful technical analysis of five alternative complete substrate kits, VisualWorks from ParcPlace Systems was chosen as the development software for the PC, UNIX® client, and UNIX server, while HP's Distributed Smalltalk (see article, page 85), which also works with VisualWorks, was the tool of choice to build and manage the client/server interaction. All system information (e.g., documentation) at this time of writing (release 2.0) still resides with the products and a central repository has not yet been chosen. Tools such as Object Lens (working with VisualWorks) or HP Odapter make relational databases look like object databases, so we know that the selection of a repository can be made very quickly when needed.

VisualWorks was the easy winner because it provides a complete environment for the development of true graphic applications that run unchanged on UNIX-system-based, PC, and Macintosh computers under their native windowing systems. Three of VisualWorks' features made it especially appropriate for the Software Solution Broker:

- VisualWorks is built on Smalltalk, a pure object-oriented language designed for fast modular design.
- VisualWorks possesses a tested set of development tools, including browsers for object classes, a thread-safe debugger, and a change manager to track modifications to the code, as well as an inspector for use in testing.

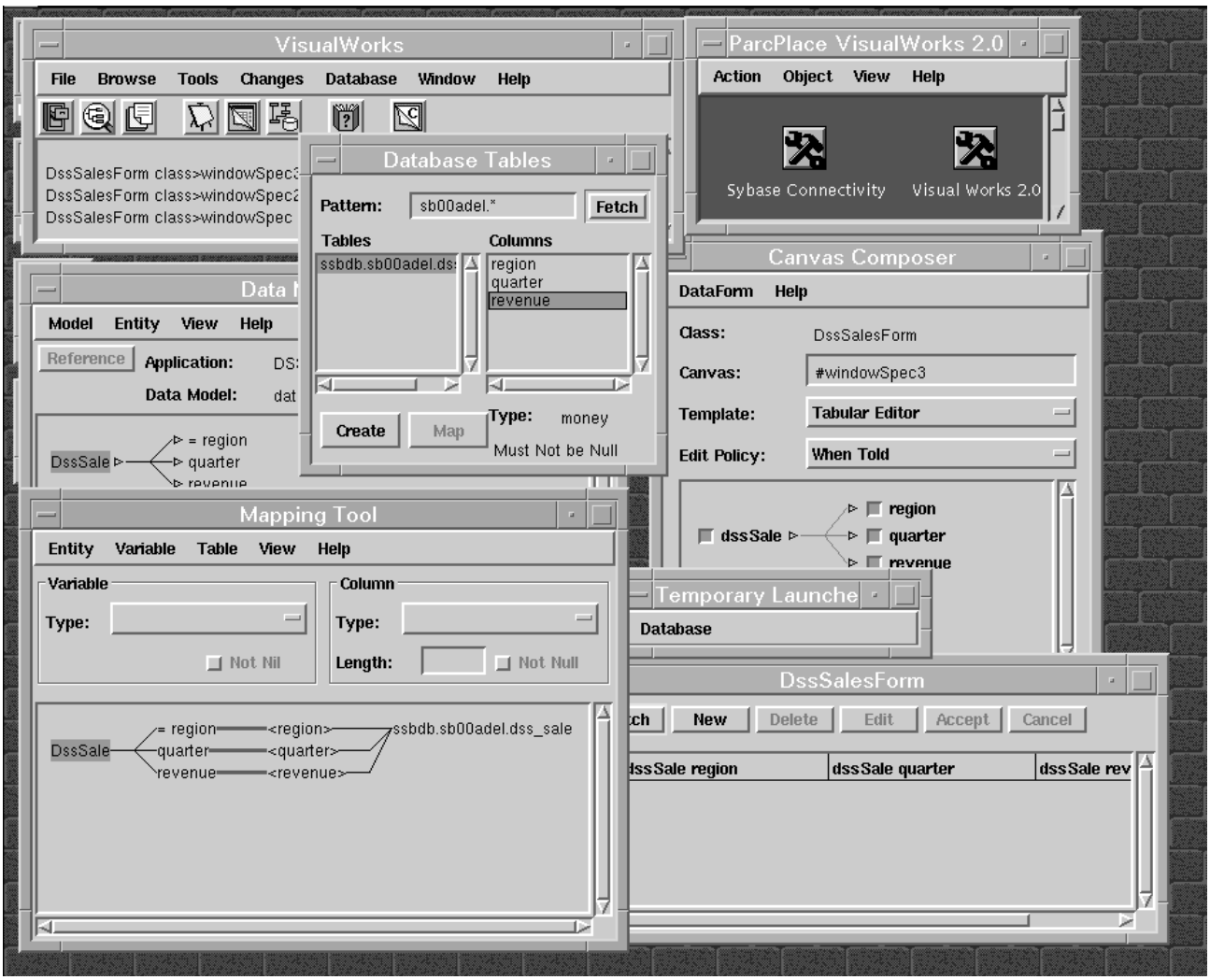

Fig. 2. A window within the Software Solution Broker showing VisualWorks tools for prototyping a customer application.

- VisualWorks has a large class library of more than 350 types of portable objects. These include a rich user interface development toolkit suitable for all major windowing systems.

  HP Distributed Smalltalk extends VisualWorks' capability for developing standalone systems into an environment for creating distributed object systems (see Fig. 5) by adding the following:
- A full implementation of the Object Management Group (OMG) Common Object Request Broker Architecture (CORBA) core services
- Common Object Services for life cycle operations such as creating objects and the relationships between them
- Sample application objects, for example for the modular partitioning of client/server functionality into semantic and presentation objects.

  These objects and services for building distributed applications are portable to all platforms supported by VisualWorks. Furthermore, they are compatible with the OMG CORBA standards. HP's Distributed Smalltalk provides seamless support of client/server interactions between VisualWorks images. CORBA compliance makes our Software Solution Broker implementation open and capable of interoperability, for instance with C++ CORBA-compliant applications, and as soon as HP Distributed Smalltalk is OSF DCE-compliant,* also with DCE remote procedure calls (RPCs). For the current release, TCP/IP or HP Sockets are being used.

* OSF DCE is the Open Software Foundation's Distributed Computing Environment.

Product Encapsulation

Everybody who has worked with spreadsheets, word processors, or CAD systems knows that similar or identical functionality does not mean that the user interfaces and more generally the visual, iconic, and mental models are comparable. For the Software Solution Broker, too, each product has its own artifacts and idiosyncrasies, its own look and personality by which we can identify it when we see it in use or on the shelf of a vendor. This unavoidable fact poses challenges for the "virtual shelf" of the Broker. Without wanting to blot out the individuality of a vendor's offering it was the objective of the development team to minimize the effort needed for the user to get accustomed to this diversity. Generally speaking, the variety has to be hidden behind a simple and consistent, product independent mode of access with uniform and intuitive graphical symbolism. A particular example is the double click used consistently to launch an application.

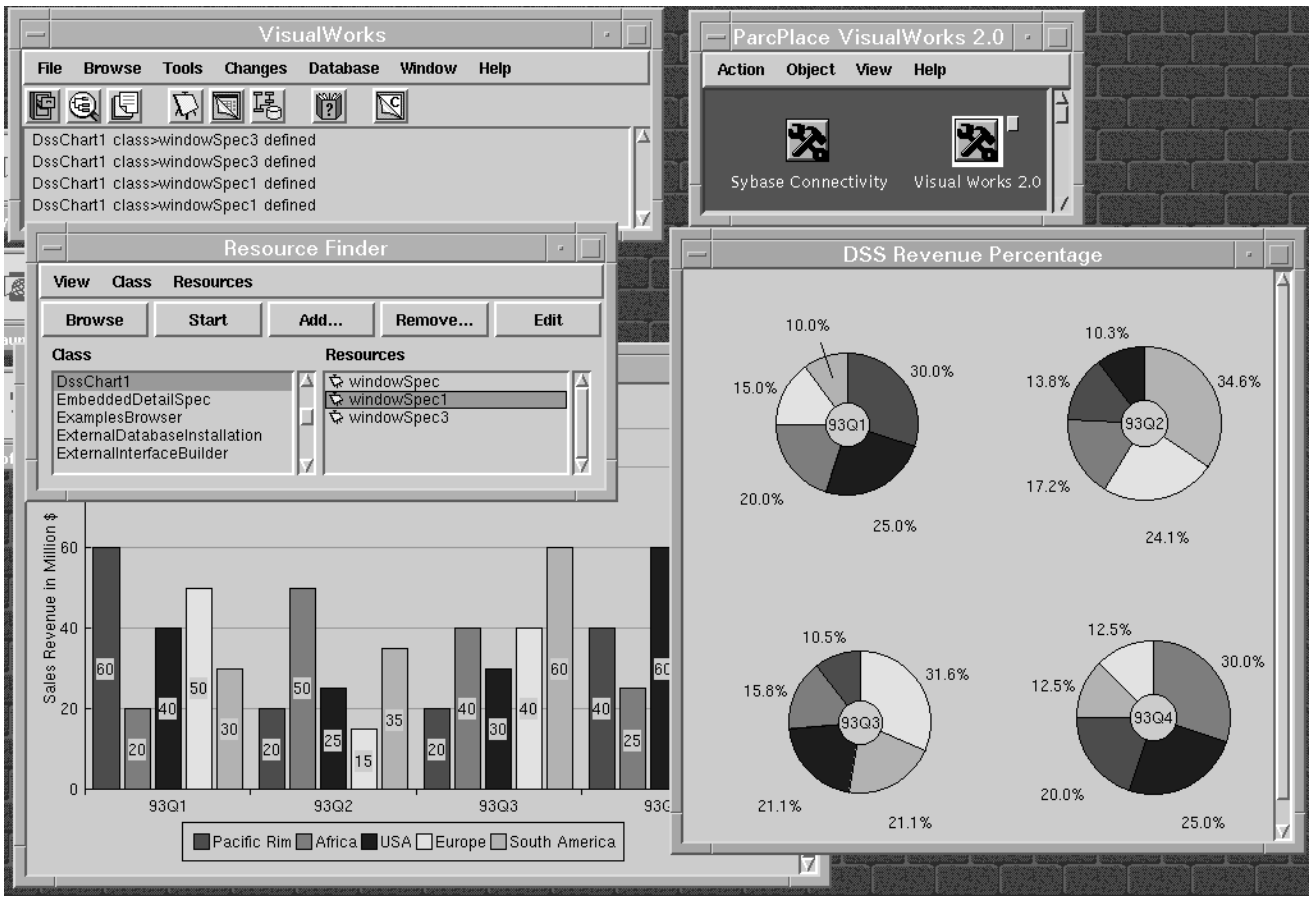

Fig. 3. Prototype display for a customer application constructed using the Software Solution Broker to select user interface and database software.

Encapsulation, in the context of the Software Solution Broker, describes a body of activities and software mechanisms that have two purposes: to integrate each product within the overall product portfolio so that the consultant can use it in its native mode, and to provide a uniform way to access the products, their associated tools, and other artifacts. This accessibility, it should be noted, is restricted to the features and artifacts that are relevant to consulting work with the customer. This means the consultant can access editors, executable code, and documentation, but isn't able to change the internal product configuration, the way it is stored and administered in folders, or the source code. Because of the intrinsic symmetry between Software Solution Broker servers and clients (see Fig. 4) the encapsulation can be done either on the server side or on the client side, provided the

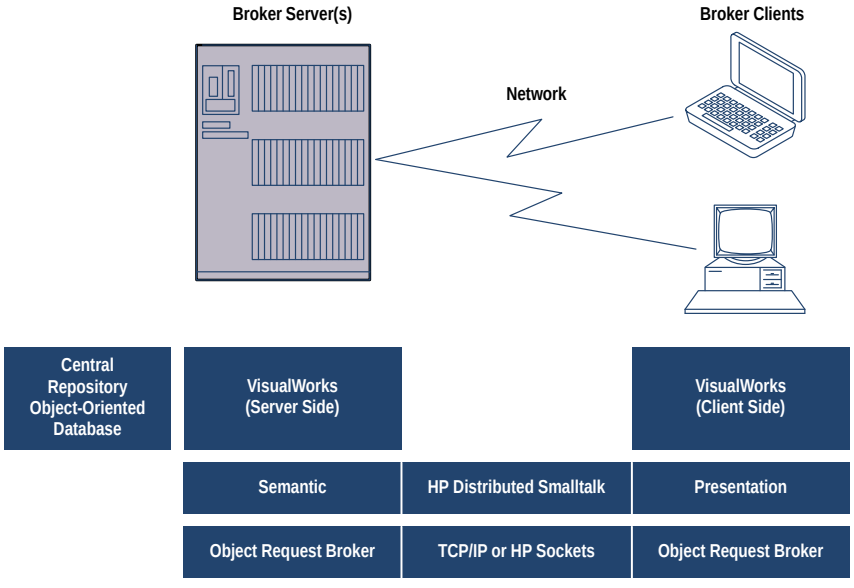

Fig. 4. Software Solution Broker software substrate, showing the client/server architecture, the user interface engine (VisualWorks), and the client/server framework (HP Distributed Smalltalk).

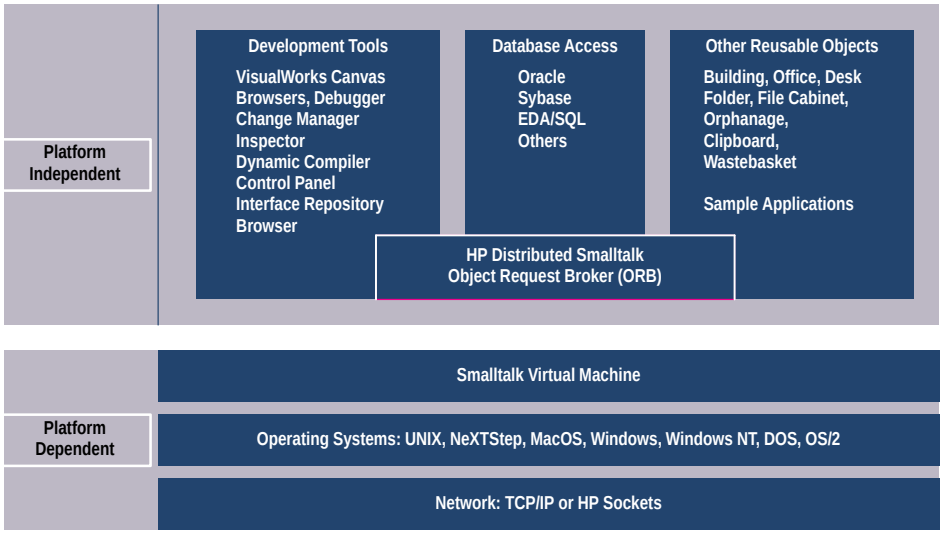

Fig. 5. HP Distributed Smalltalk and VisualWorks provide a full development and run-time environment for distributed computing.

classes FolderPlusPO and EncapsulationDialog are present in the client. These two classes will be discussed below.

It is the already mentioned semantic/presentation split, together with object-oriented features such as *inheritance* and *polymorphism* that make the encapsulation effortless. The semantic/presentation object distribution model is HP Distributed Smalltalk's implementation of a distributed client/server architecture. In this model, classes always appear in logical pairs, one representing the server semantics, the other their presentation in the client. Consequently, the class instances or objects also come in pairs. Take for instance the window object. Every window is composed of two logical parts: its shared (semantic) properties such as its rectangular shape, and its local and personal (presentation) attributes such as color. In general, a semantic object often has (and controls) many different presentation objects, which in the case of the Software Solution Broker handle the remote user interactions, thus reducing network traffic. For instance, one semantic data display object creates and controls different presentations of the data as a bar chart and a pie chart in a decision support system. HP Distributed Smalltalk allows various modes of collaboration between the semantic and presentation objects, including messages that are handled by the object request broker. (For a simple but complete example see the *HP Distributed Smalltalk User's Guide*, chapter 10.)

After this abstract introduction of HP Distributed Smalltalk's semantic/presentation split architecture we will describe in more concrete terms how it works for the encapsulation procedure. As stated above, encapsulation must achieve two goals: it has to present a graphical representation of the artifact (product, tools, demos, documentation) in its native mode to the remote client, and it must allow the remote user to launch the artifact at the server side through this representation. HP Distributed Smalltalk has a pair of classes, MediaSO and MediaPO, that accomplish exactly this. (The suffixes SO and PO imply that semantic and presentation objects, respectively, are spawned by these classes). Tracing the interaction diagram between two objects of these classes we found that there exists a ready-made method called updatePresenter, visible in the MediaSO class, that creates the remote presentation object of a product or other artifact in the server. To customize the generic MediaSO and MediaPO classes and the method updatePresenter for the encapsulation of specific artifacts we first created the narrower subclasses ArtifactSO and ArtifactPO. Then we augmented ArtifactSO with the attributes of artifacts such as vendor and product names. Finally, using overloading, we extended the method updatePresenter to include, among several other administrative tasks, the crucial behavior required for launching the artifacts while exporting their display to the client platform.

Concurrent with this architectural design of the classes and methods that bring about encapsulation in the Software Solution Broker, a few product dependent steps must also be taken. This is done at the instance or object level of every concrete artifact (such as a product) so that it will behave in its expected, native mode. This is a simple matter of inserting the right environment variables and parameters in an encapsulation dialog window. The required information can easily be gleaned from the installation manual of the particular product that is being encapsulated. Finally, products, tools, and other components are put into folders and the encapsulation is done.

Use of Object Technology

The design and building of the Software Solution Broker were characterized by a short development time, a minimal amount of new coding, and a high degree of reuse. The major reason is the application of object-oriented technology. The object-oriented use is pervasive throughout the design, as indicated above, but it is helpful to point to specific examples. We'll give two examples for the object-oriented features inheritance and polymorphism in the context of encapsulation.

One of the examples has just been described: the subclass ArtifactSO of the class MediaSO inherited the method updatePresenter, which in turn, through the feature of polymorphism, was overloaded (that is, extended to include additional functional behavior).

The encapsulation dialog window provides another example. As an administrative tool, it is not available to the user. It is an object built from a subclass of the existing HP Distributed Smalltalk class called SimpleDialog. From this class, the window inherits characteristics such as its property to pop up in front of other windows (it's not obscured), its basic layout

HP Software Solution Broker Accessible Products

Vendors

- Cognos Corp.
- ParcPlace System
- XVT Software
- Itasca System
- Informix
- Neuron Data
- Sybase
- Unison Software
- ProtoSoft
- Oracle
- Dynasty
- NetLabs

Tools

HP-UX*

- Cognos Corp.
  - PowerHouse 4GL 7.23
- ParcPlace System
  - VisualWorks 2.0
  - VisualWorks with Sybase connectivity
- XVT Software
  - XVT-Design (C Developer Kits)
  - XVT/XM (C Developer Kits)
  - XVT-Power++
  - XVT/XM (C ++ Developer Kits)
  - XVT-PowerObject Pak I
- Itasca System
  - ODBMS Server
  - Developer Tool Suite
  - C Interface
  - Lisp Interface
  - API Libraries (C++, CLOS, Ada)
- Informix
  - Informix Online R4GL
  - Informix WingZ
  - Informix SE R4GL
  - Informix SE ISQL
  - Informix Hyperscript Tools
  - Informix Online ISQL
- Neuron Data
  - Smart Elements (Nexpert object)
  - Smart Elements (Openedit)
  - Open Interface Elements (Open edit)
  - CS Elements (Openedit)
- Sybase
  - SA Companion (client & server)
  - SQL Monitor (client & server)
  - SQL Debugger inspector
  - SQL Debugger console
  - SQL Data Workbench
  - SQL APT Edit
  - SQR Workbench (Easy SQR)
  - Open Client/Server
  - ISQL/SQL Server
- Unison Software
  - Maestro
  - Load Balancer
  - Express
  - RoadRunner
- ProtoSoft
  - Paradigm Plus
- Oracle
- NetLabs
  - Net Labs/AssetManager
  - NetLabs/Vision
  - NetLabs/Assist
  - NetLabs/NerveCenter
  - NetLabs/Manager
  - NetLabs/OverLord Manager
  - NetLabs/Discovery

MS Windows

- Cognos Corp.
  - PowerHouse Windows 1.2E
  - Axiant
  - Impromptu
  - PowerPlay
- ParcPlace System
  - VisualWorks 2.0
  - VisualWorks with Sybase connectivity
- XVT Software
  - XVT-Design (C Developer Kits)
  - XVT/Win (C Developer Kits)
  - XVT-Power++
  - XVT/Win (C ++ Developer Kits)
  - XVT-PowerObject Pak I for MS Windows
- Itasca System
  - ODBMS Server
  - API Libraries (C++)
- Informix
  - New Era
- Neuron Data
  - Smart Elements (Nexpert object)
  - Smart Elements (Openedit)
  - Open Interface Elements (Open edit)
  - CS Elements (Openedit)
- Sybase
  - Net-Library
  - Open Client /C
  - SQL Monitor Client
  - SQR Workbench
  - APT Execute
- ProtoSoft
  - Paradigm Plus
- Oracle
- Dynasty Technologies
  - Dynasty
- NetLabs
  - NetLabs/Vision DeskTop

Artifacts

- Cognos Corporation
  - QUICK Application
  - QUIZ Application
  - PDL Application
  - QDESIGN Application
  - QTP Application
  - QUTIL Application
  - PDL And Utilities Reference Manual
  - PowerHouse for UNIX – Primer
- ParcPlace System
  - Product Overview
- XVT Software
  - Product Overview
  - XVT Design Tutorial
  - XVT Database Demo
  - XVT-Power++ Overview
  - XVT-Power++ Demo Guide
  - XVT Power ++ Earth Demo
- Itasca System
- Informix
  - Product Overview
  - Informix R4GL Demo
  - Six demos with source codes
  - Informix ISQL Demo
  - Informix Hyperscript Demo
- Neuron Data
  - Product Overview
  - Notepad Widget example with source files
  - Pack example with source files
  - Print widget example with source files
  - Resize widget example with source files
  - Resource Picker example with source files
  - Scripting example with source files
  - Scroll area usage example with source files
  - Scroll bar usage with source files
  - Sliders usage with source files
  - Special widget example with source files
  - String search example with source files
  - Text edit validation example with source files
  - Windows MD example with source files
  - Alert Windows example with source files
  - Browsex example with source files
  - Browsinc example with source files
  - Cbox example with source files
  - Chart example with source files
  - Clock Widget example with source files
  - C++ Notepad widget example with source files
  - Drag drop example with source files
  - Draw example with source files
  - DropDown pale example with source files
  - File manager example with source files
  - File name translator example with source files
  - File Picker example with source files
  - Floating window example with source files
  - Cantt chart example with source files
  - Help engine example with source files
  - Help viewer example with source files
  - ICON generator example with source files
  - List Box example with source files
  - Local drag drop example with source files
  - Menu example with source files
  - Multiple font text example and source code
  - Notepad example with source files
- Sybase
  - SyBooks
  - APT Demo
  - Compute example with source files
  - Csr_disp example with source files
  - i18n example with source files
  - blktxt example with source files
  - Five other examples with source files
- Unison Software
- ProtoSoft
- Oracle
- Dynasty
- NetLabs

HP-UX is based on and is compatible with Novell's UNIX® operating system. It also complies with X/Open's* XPG4, POSIX 1003.1, 1003.2, FIPS 151-1, and SVID2 interface specifications.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

X/Open is a trademark of X/Open Company Limited in the UK and other countries.

MS Windows is a U.S. trademark of Microsoft Corporation.

with a text box and O.K. and cancel buttons, and its link to a value holder that holds the environmental variables, names, and other information needed for the encapsulation. The only method needed in addition to inherited ones is the one requesting the encapsulation parameters mentioned above.

The same procedure, that is, the use of predefined classes and thus minimal coding, applies to HP Distributed Smalltalk's folders containing the encapsulated product with its tools and other artifacts. The HP Distributed Smalltalk class FolderPO (PO indicates it is the folder class spawning presentation objects) has a method windowMenu, which creates a window with several pop-up menus that have labels such as Action, Edit, and so on. For a subclass of FolderPO called FolderPlusPO, these properties of windowMenu are inherited, but windowMenu is also changed (while keeping the same name), by the addition of a method artifactCreate and its label in one of the pop-up menus of windowMenu. The method artifactCreate is responsible for the inner workings of the encapsulation dialog window mentioned above.

Development Methodology

Funding for the Software Solution Broker project was subject to the condition that the development team find, justify, and implement a design that brings the tools to the consultants in the fastest possible way with the least amount of resources, including development, maintenance, and support resources. At the same time, every released version, even the very first one, had to find immediate user acceptance. Based on these stipulations the team chose a development method that is a hybrid of *iterative prototyping* and the *Fusion method.*

Our reasons for favoring iterative prototyping over a classical software design paradigm that starts with a complete specification (such as the so-called *waterfall model*) were:

- Time constraints. There are never enough engineer-months to write a complete specification, implement and test it into production strength.[1]
- Constraints imposed by the intrinsic nature of the Software Solution Broker tool we were building, that is:
  - Client-side usability. The GUI that was eventually chosen is the result of repeated testing by potential users to achieve maximum ease of use and intuitiveness, and this amount of trial-and-error cannot be specified in advance.
  - Tool accessibility. The different products on the virtual shelf have different behaviors and their own requirements for resources and administration, and creating the encapsulation process again requires much experimentation and gradual maturation based on experience that cannot be specified a priori.
  - Using the object paradigm. The software substrate chosen (HP Distributed Smalltalk with VisualWorks) is well-suited for the rapid development of GUI and client/server applications.

Based on these considerations, our overall approach was that of evolutionary prototyping, in which a fully functional prototype is ushered through repeated refinement steps into a production-strength end product. We realize that often a prototype leads only to an executable specification or a validated model, not a high-quality, stable product. However, in our case the sophisticated framework of HP Distributed Smalltalk with its semantics/presentation split and VisualWorks with its Model View Controller ensured full functionality and high quality at each refinement step because we reused the existing, high-quality code (including the library of classes) and very sparingly added new, thoroughly tested code, preferably as instances (objects) of the existing class library.

Fusion Method

While iterative prototyping can be seen as a software development philosophy that is primarily dictated by business requirements such as time to market, break-even time, or optimal return on investment, the Fusion method[2] was developed with the goal of creating a language independent, comprehensive, software project management method. Being a systematic object-oriented development method, it blends well with our software substrate, which we chose based on openness, compliance with industry standards, ease of use, and the ability to separate the server (semantics) from the remote clients (presentation). The Fusion method emphasizes a modular design process in clearly demarcated phases, so it synchronizes well with the iterative prototyping approach, which requires the repetition and refinement of certain development stages without impacting others. Furthermore, the Fusion method insists that a software development process of the complexity encountered today must cover the entire software development life cycle. The Fusion method's phased development process served as the blueprint for the Software Solution Broker. It can be summarized as follows[2] (our italics):

Starting from a *requirements document*, the *analysis phase* produces a set of models that provide a declarative description of the required system behavior. The analysis models provide high-level constraints from which the design models are developed. The *design phase* produces a set of models that realize the system behavior as a collection of interacting objects. The *implementation phase* shows how to map the design models onto implementation-language constructs.

In our hybrid approach we take an early, loosely defined functional prototype as our initial requirements definition (an executable specification), to be modified and refined in subsequent iterations through the three phases of analysis, design, and implementation. After each of these phases a review of the phase outputs is conducted by the development team in conjunction with users. The results of this audit are prioritized and, if deemed important, incorporated into the prototype which, through several of such review loops, evolves after a full cycle into the production product. (For details about the outputs mentioned and the complete Fusion process breakdown see reference 2, especially Appendix A.)

In summary, the two complementary methods of iterative prototyping and Fusion serve two main purposes. First, at the end of each prototyping cycle a fully functional production-strength product is released. Second, the three Fusion phases—analysis, design, and implementation—of every cycle are independent of the phases in another cycle. Therefore, we are in effect working towards several releases at the same time (see Fig. 6).

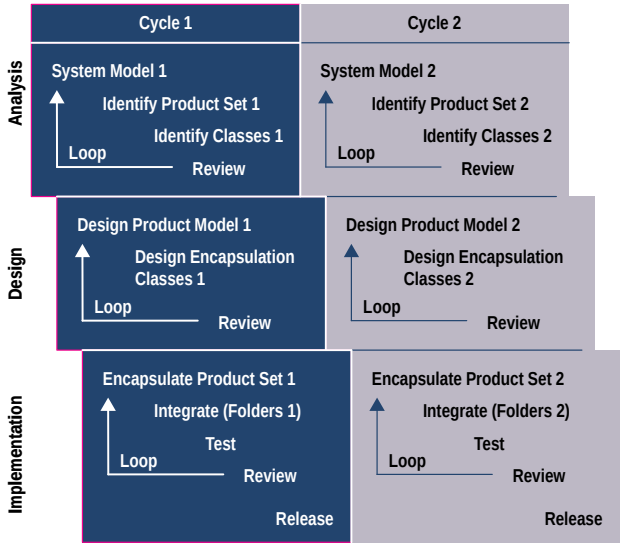

Fig. 6. Software Solution Broker development used iterative prototyping and the Fusion method, resulting in parallel development cycles.

Customizing the Software Solution Broker

In addition to being a productivity tool and a hub of product expertise for HP's technical consultants, the Broker can be customized to meet the business needs of end customers as well. To sketch how such a customization can be done using the object-oriented framework of HP Distributed Smalltalk, imagine a vendor of CAD (computer-aided design) software. Rather than offering shrink-wrapped software packages on the shelves of the store the retailer wants to offer customers an environment where they can, by navigating through virtual shelves, choose interesting products and "test drive" them in the store before deciding what to buy.

For an end customer such as the CAD software vendor, the Broker can be customized by mapping the particular customer requirements into several levels of design complexity. These levels describe in technical terms what level of intervention into the framework of HP Distributed Smalltalk is needed to alter and customize the existing classes and methods. On the lowest level, the requirements fit the HP Distributed Smalltalk framework exactly, and the system can be built from existing classes without change. A higher level of intervention would be needed to construct the Software Solution Broker for the CAD software vendor. Slight modifications of core services (relating to containment and life cycle semantics), in addition to class augmentation and overloading of methods, would be recommended. Working with predefined, well-documented levels of intervention that are necessary to meet a customer's requirements has the advantage of communicating to the customer in advance, during the analysis and before system design begins, how much reuse of the framework is possible, and how much non-framework augmentation is necessary. Intervention levels are thus not only technical assessments but also indicators of the final costs for the system.

Conclusion

The Software Solution Broker was not a typical client/server application development project. We were not primarily concerned about two-tier or three-tier architectures, about objects per se, about the one "right" programming language, or about coding. In fact, we went the opposite route. Based on the working requirements of HP's technical consultants and our own analysis of how consultants work with customers, we resolved to translate these requirements into a system built from distributed objects. The building, however, consisted mainly in the skillful choice of existing classes and the exploitation of HP Distributed Smalltalk's framework. The novelty in our approach lies not in the coding of new structures, but in the extensive application of reuse. In fact, whenever new code seemed required, we took it as a warning that further analysis was needed to look for prefabricated code within the framework of HP Distributed Smalltalk. This simple principle, essential for a fast time to market, also guaranteed a short turnaround time and high quality.

Through its first two releases, 1.0 and 2.0, the Software Solution Broker can be viewed as a distributed productivity tool offering three overlapping types of services. These three types can be described metaphorically as a virtual software shop for the display of individual products, a consultative workbench or simulated classroom for studying and experimenting with several collaborating products, and a virtual demo center with remote satellite offices where the technical consultants can build prototypes and create demos for a customer. Looked at from a broader perspective, however, the Software Solution Broker architecture and implementation are, with small customization, also ideal for other, related applications that require one (or a few) persistent centers and many locally distributed and individually presented clients. One example is software distribution. Another is the establishment of a worldwide software application development lab where each satellite group can develop its own part locally, check it in with a central repository where it is available to the other satellites, and participate remotely in the integration of the parts into a system. Furthermore, object technology, with its concept of containers, makes available compound documents (text, picture, voice, video, etc.) that can be employed also on the nontechnical side of business as vehicles for elaborate project proposals and other communication with business customers—for instance, to propose a solution by showing a video of a prior, successful installation (this would take the place of a paper document of reference sites). In this role, the Software Solution Broker can be a *worldwide business solutions exhibit* and a convenient repository for a portfolio of repeatable solutions from which the customer, advised by a consultant, can select products the way we now choose from mail-order catalogs.

Acknowledgments

A project survives by the benevolent patience of its sponsors, the enthusiasm of the project team, and the acceptance of the receiving customers. In the case of the Software Solution Broker we've tried to justify this benevolence by turning our research ideas into a useful tool in the shortest possible time.

This feat was only possible with the support of our managers David Kirkland and Sherry Harvey. Special thanks are due Chu Chang, general manager of the Professional Services Division, for his encouragement. We are also very grateful for the contributions and for the heated (but object-ive) discussions with friends and partners from many HP's entities. Nothing, however, would have happened without the quick and thorough feedback from our customers, the technical consultants in the field. The Software Solution Broker is dedicated to them.

References

1. F.P. Brooks, *The Mythical Man-Month: Essays in Software Engineering*, Yourdon Press, Englewood Cliffs, 1982.

2. D. Coleman, et al, *Object-Oriented Development—The Fusion Method*, Prentice Hall, 1994.