An Architectural Model for Application Integration in Open Hypermedia Environments

E. James Whitehead, Jr.
Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425
Tel: +1 714 824-4121
E-mail: ejw@ics.uci.edu

ABSTRACT

This paper provides an architectural framework for modeling third-party application integrations with open hypermedia systems, which collects and extends the integration experience of the open hypermedia community. The framework is used to characterize applications prior to integration, and describe the qualities of a complete integration. Elements of the architectural model are artists, which are used to manipulate anchors, links, and native application objects; communicators, which manage information flow to and from the open hypermedia system; and containers which group the other elements. Prior integration experience is collected in a standard way using the model. Guidance in selecting the final integration architecture is provided by this prior integration experience, in conjunction with the degree of difficulty of an integration, which is related to the integration architecture.

KEYWORDS: Open hypermedia systems, third-party applications, integration, software architecture

1. INTRODUCTION

Open hypermedia systems [15], emphasize delivery of hypermedia functionality to the third-party applications populating a user's computing environment. Yet with the exception of the excellent paper by Hugh Davis, et al. describing application integration with the Microcosm system [5], the emphasis to date has been on describing the open hypermedia systems themselves, rather than the details of integrating third-party applications to work with the system. As a result, there is little guidance available for those who wish to perform such an integration, or who wish to design their applications or hypermedia systems to be easier to integrate. Since each open hypermedia system has integrated many external applications with their system, there is currently a large body of experience within the Open Hypermedia community on how to perform these integrations. This paper gathers this community knowledge, collecting application integration experience from Microcosm [7] [4], DeVise Hypermedia [8] [9], HyperDisco [21], Hyperform [20], Chimera [2], SPn/HBn [18][11], HOSS [14], Multicard [17], and Sun's Link Service [16].

This paper introduces a method for creating architectural models of application integrations with open hypermedia systems, and demonstrates how this modeling framework can be used to:

- Model applications prior to integration
- Describe the characteristics of a *complete* integration
- Provide guidance in selecting the architecture of a finished integration
- Provide a rough estimate of the degree of effort required to perform an integration
- Categorize existing integrations.

In the next section, two example application integrations are described, motivating the architectural modeling method presented afterwards. The characteristics of an application prior to integration are then described, followed by a description of a complete application integration using the model, and common integration architectures. A table providing guidance in the selection of an integration architecture is given, along with discussion of its implications. Rounding out the paper is a discussion of related work, including a discussion of the World Wide Web.

2. MOTIVATING EXAMPLES

To highlight the architectural similarity between external application integrations, and motivate the elements of the simple architectural model, this section describes an integration of the XEmacs text editor with the Chimera system, and an integration of Microsoft Calendar with Microcosm using the Universal Viewer.

2.1 XEmacs/Chimera Integration

XEmacs is a widely used variant of the ubiquitous Emacs text editor which features a built-in customization language, Emacs Lisp, in which many of the functions of the text editor are defined. Emacs Lisp allows communication to external applications either via a socket, or via a text stream to a "captive" process started and managed by XEmacs. XEmacs offers excellent services for creating regions of text,

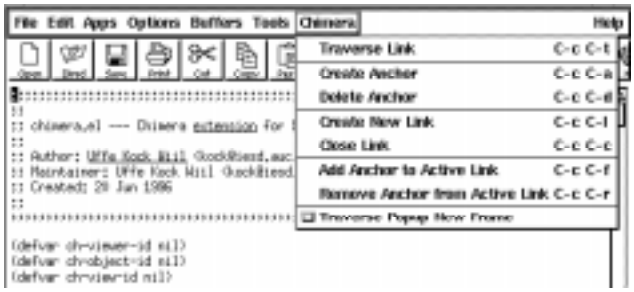

Figure 1: Screen shot of XEmacs/Chimera integration

regions using faces, a presentation description such as underlined boldface. XEmacs also allows the creation of custom user menus, and the ability to add a user defined routine (a hook) into the processing of a key or mouse press. The XEmacs/Chimera integration, shown in Figure 1, makes use of these XEmacs features.

The XEmacs/Chimera integration allows the addition of hypermedia anchors and links to a text document without modifying it with markup codes. By either using the Chimera menu or control key sequences, anchors can be created and deleted on ranges of text, a new link may be created and made "active," anchors may be added or removed from the "active" link, and a link traversal can be initiated. Within Chimera each user has an active link, to which anchors can be added or removed without all applications knowing its link identifier. With an active link, a user does not have to enter a link identifier to modify a link, and applications are freed from having a "create link" mode in their interface, resulting in simpler, non-modal link user interfaces. The XEmacs/Chimera integration also provides some control over the presentation of link traversals, allowing selection between popping up a new window or using an existing window for display of the destination anchor.

The XEmacs/Chimera integration architecture is shown in Figure 2. Since XEmacs does not support remote procedure calls (RPC), Chimera's native communications format, the integration uses the text stream I/O of XEmacs to communicate with the Chimera Shell, which converts text commands into Chimera RPC messages. The Chimera menu and control key sequences are created by custom Emacs Lisp code running within XEmacs. For anchors, custom Emacs Lisp functions manage their creation, deletion, display, and selection, while for links other custom functions manage their creation, the addition and removal of anchors, initiating a link traversal, and display of a link traversal. These custom functions interact with XEmacs through the Emacs Lisp library, and with Chimera by sending text commands to the Chimera Shell.

The Chimera Shell is started as a captive process from within XEmacs, using specialized Emacs Lisp library functions. When a hypermedia operation is required, a text command is sent to the Chimera Shell, which parses it, formats the call into RPC format, and invokes an entry point in the Chimera Server. The Chimera Server acts on the command, then the Chimera Shell formats the RPC return values into text before printing them. The text return values are read by custom Emacs Lisp code, which converts them into native types.

2.2 Calendar/Microcosm Integration

Microsoft Calendar is a calendar management system available for Windows, which allows the entry of appointments and special days, and multiple views of the calendar information, such as by month or by day. Calendar does not have a built-in customization language, or a built-in external application program interface. As a result, the integration of Microsoft Calendar with Microcosm, shown in Figure 3, and initially described in [5], employs the Microcosm Universal Viewer. The Universal Viewer, unbeknownst to its host application, attaches a Microcosm icon to the application window's title bar. Selecting this icon causes a Microcosm menu to be displayed.

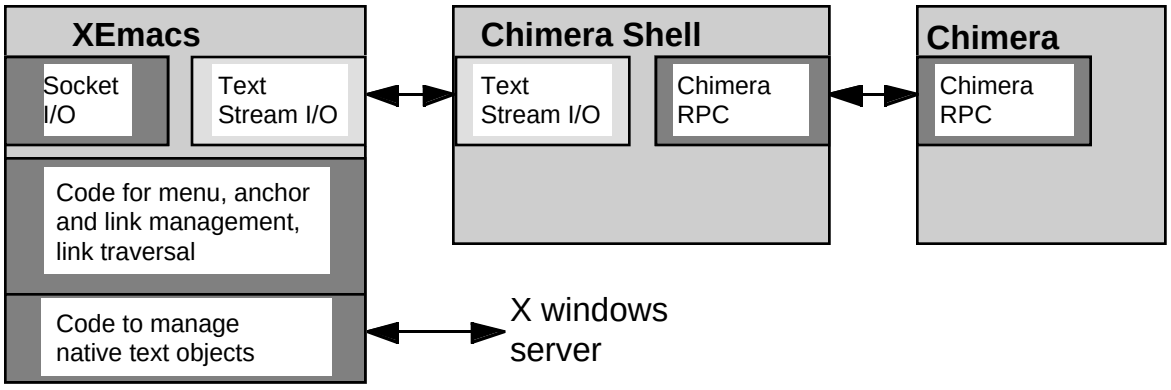

Figure 2: Boxes and arrows architecture diagram of the XEmacs/Chimera integration architecture.

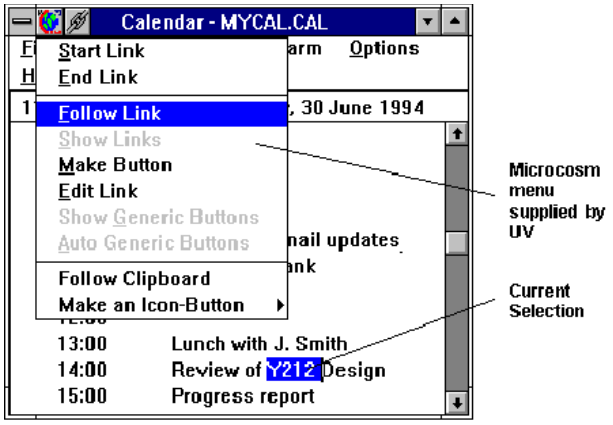

Figure 3: Screen shot of Calendar/Microcosm integration. Reused, with permission, from [5].

The Calendar/Microcosm integration, like all integrations which employ the Universal Viewer, provides the capability to create, edit and follow (traverse) links, and to create "follow link" buttons which are displayed in the title bar and traverse a single link.

Universal Viewer integrations employ the application's facilities for highlighting regions of text to select anchors. When a region of text is highlighted and Follow Link (or some other action) has been selected from the Universal Viewer menu, the application is instructed via macros, Microsoft Windows DDE, or a Windows Recorder File to pick up the selection and communicate it back to the Universal Viewer via either the clipboard, DDE, or a named swap file. In the case of the Calendar/Microcosm integration, a macro is used to copy the highlighted selection to the clipboard.

The architecture of a generic integration of a Windows application with Microcosm via the Universal Viewer is shown in Figure 4. While the application contains code for managing its native object type and the highlighting of anchors, the Universal Viewer contains code for managing links and link traversals. Because the application does not have a built-in customization language, it cannot be modified to send native Microcosm DDE messages. As a result, the Universal Viewer acts as a mediator between the application and Microcosm, employing one of several communications paths to retrieve anchor information from the application, which it combines with the user request and forwards along using the native Microcosm DDE communications protocol.

2.3 Similarities

The XEmacs/Chimera and Calendar/Microcosm integrations have several similarities. In both integrations, because the application could not communicate using the native protocol of the hypermedia system, a communications intermediary was needed. For the XEmacs/Chimera integration, this intermediary was the Chimera Shell, while for the Calendar/Microcosm integration, it was the Universal Viewer. Since neither XEmacs nor Calendar had built-in facilities for manipulating anchors or links, these capabilities had to be provided by the integration, along with a user interface to access them.

3. ARCHITECTURAL MODEL OF INTEGRATIONS

The architecture of an application integration may be characterized based on its user interface support for manipulating anchors and links, and the communication path between the application and the hypermedia system. As a result, the architectural framework for integrations consists of three types of elements: *artists*, which model user interface aspects, *communicators*, which model communications, and *containers*, which are used to group the other elements. This architectural framework does not claim to accurately model all aspects of an application. Instead, the framework emphasizes the most important aspects of an application integration: user interface support for anchors and links, and communications with the open hypermedia system. All other aspects of the integration are not part of the abstractions presented in this framework. This emphasis focuses attention on critical architectural elements, while abstracting away other elements which do not impact the integration.

Artists. An artist is any code which is responsible for

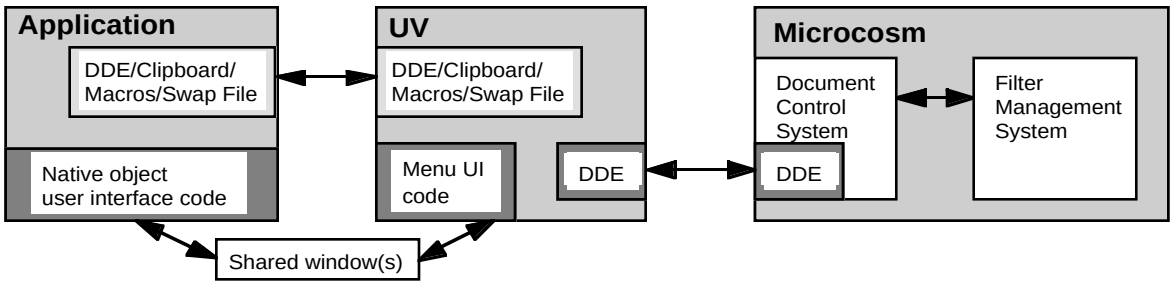

Figure 4: Boxes and arrows diagram of generic integration of an application with Microcosm using the Universal Viewer.

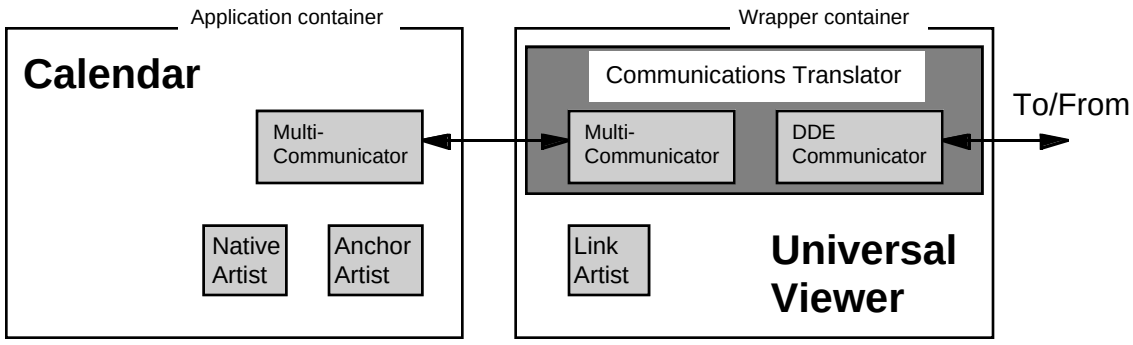

Figure 5: Software architectural model of the Calendar/Microcosm integration using the Universal Viewer.

managing a user interface which depicts and mediates the manipulation of a particular object [12]. For modeling integrations, there are three artists of interest: a *native artist*, which depicts and mediates operations on an application's native data type (e.g., a word processor is considered an artist for its native file format); an *anchor artist*, which depicts and mediates operations on anchors; and a *link artist*, which depicts and mediates operations on links. The subdivision of user interface capability into these three artists is motivated by noting that the user interface for hypermedia-aware applications contains affordances for the manipulation of anchors and links, and for the manipulation of the application's native object type. Anchor and link artists are separate from the rest of the user interface since non-hypermedia aware applications do not have facilities for anchor and link manipulation, and hence must be provided during an integration.

Communicators. A communicator is any code which is responsible for establishing and handling communications between programs using a specific protocol. For example, in the Chimera system, the code which manages communications using the Chimera RPC protocol would be modeled as a Chimera Communicator. A *communications translator* consists of two communicators within a *translator container* which holds the two communicators and the code which translates between the two communications protocols.

Containers. A container is used to group together the other architectural elements (artists, communicators, and translators). There are three container types used in the model: an *application container*, which represents the application being integrated with the hypermedia system, a *wrapper container*, which represents any wrapper (or shim) processes used in an integration, and a *translator container*, described above. A container frequently maps to an operating system process, and can be assumed to contain all code not explicitly modeled by other architectural elements.

There are two rules which govern the use of containers. All artists and communicators must belong to a container. A container may contain other containers, but must fully contain them.

Example. As an example, the architectural modeling elements will be used to model the integration of Calendar with Microcosm, using the Universal Viewer. Figure 5 shows a diagram of the final integration.

In this integration, the native artist and the anchor artist both reside within the Calendar application container. The native artist represents functionality within Calendar, such as adding and removing entries on a date, for manipulating its native calendar objects. Since Calendar does not have a built-in notion of hypertext anchor, the anchor artist represents the highlighting of text within Calendar, which the Universal Viewer copies to the clipboard before reading and transmitting as anchor text to Microcosm. Despite only describing the ability to highlight text, the anchor artist does model the code within Calendar which manages the anchor user interface. The link artist, located within the Universal Viewer wrapper container, represents the operations supplied by the title bar pull-down menu, and the management of link buttons on the title bar.

The multi-communicator models the information flow of selected text from Calendar, onto the clipboard, and then into the Universal Viewer, along with the selection of destination anchors using a search macro. The communications translator in the Universal Viewer converts between Microcosm's native DDE communications, and the more limited communications options available to interact with Calendar. To reduce visual clutter, the translator container is not explicitly labeled in Figure 5.

4. CHARACTERIZING AN APPLICATION PRIOR TO INTEGRATION

The architectural model allows those characteristics of an application which are relevant to its integration with an open hypermedia system to be modeled. This same model is

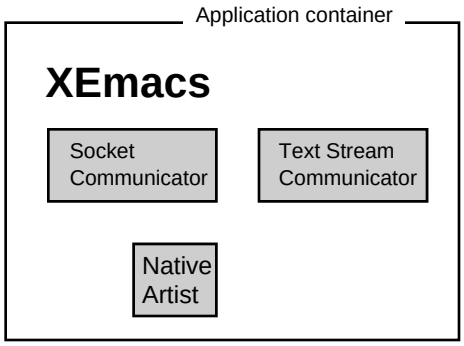

Figure 6: Architectural model of XEmacs prior to integration.

also used to describe the characteristics of a complete integration. The difference between these before-and-after models provides a high-level description of the work required to integrate the application.

Any application which is being considered for integration may be characterized prior to integration both in terms of the existence of anchor and link artists, and based on its level of communication. This uses the architectural model in a descriptive sense, describing the initial state of the application. By definition, an application always contains a native artist for manipulating its native object type, and in the worst case this artist may also serve as the anchor artist to achieve a "large-chunk" hypermedia effect. If the application has a built-in notion of hypermedia, then it will already have provisions in its user interface for links, and probably anchors. For example, FrameMaker has an internal notion of hypermedia, and contains user interface elements for manipulating anchors and links. As a result, FrameMaker contains both anchor and link artists prior to integration. Other applications which are non-hypermedia-aware lack a link artist, and typically also lack a user-interface facility which can be used to represent and manipulate anchors of smaller chunk size than a complete object. The GIF viewer integrated with Chimera exemplifies this, as it had no internal notion of hypermedia, and hence no link artist, while also possessing no object smaller than a complete image which could be used as an anchor.

The initial state of an application's communications ability can be described (from the perspective of the open hypermedia system) as being either "native," non-native," or "non-communicative." An application which is a *native* communicator fully understands the communications protocol used by the open hypermedia system when communicating with its clients. For example, the native communications of Microcosm and DHM (Windows version) are different protocols layered on top of DDE, while the native Hyperform and HyperDisco protocol is an ASCII linearization of Lisp and Scheme data structures, sent through sockets, and Chimera uses RPC messages. Typically only an application written from the ground-up to use a specific hypermedia system's protocol is considered to be a native communicator. A *non-native* application has an external application program interface (API) which may be used to achieve hypermedia functionality, especially a link traversal operation, but which differs from the native communications protocol of the open hypermedia system. FrameMaker is an example of a non-native communicator, since its hypermedia functionality is accessible via its external API, a collection of RPC entry points which differs from the protocols of all current open hypermedia systems. A non-native application is modeled as having a communicator for the application-specific non-native protocol. A *non-communicative* application is closed to the outside world except via its user interface, and does not have an external API. These applications are modeled as having no communicators.

Example. The XEmacs application, modeled prior to integration, is shown in Figure 6. Since XEmacs contains the ability to communicate via either a network socket connection, or via a text stream, its communications ability is modeled using two communicators, a Socket Communicator and a Text Stream Communicator. XEmacs does have the ability to highlight regions of text with an underline character using its built-in extents and faces features, thus providing one means of visually depicting an anchor, however, it does not have the ability to directly create or destroy anchors from its user interface, or to automatically maintain an association of a screen anchor with a hypermedia system anchor handle, and hence XEmacs is modeled without an anchor artist. XEmacs also does not contain link manipulation facilities, and hence is modeled without a link artist.

5. ARCHITECTURAL PROPERTIES OF A COMPLETE INTEGRATION

The architectural model makes it possible to generalize the experience of many integrations of applications with open hypermedia systems and concisely describe the properties of a *complete* integration:

- Existence of all three artists: native, anchor, and link.
- Existence of a communicator for the hypermedia system's native protocol.
- If the application has a communicator for a protocol which is different from the hypermedia system's protocol, then a communications translator must be present which translates into the hypermedia system's protocol.

In essence, these properties state that a complete integration consists of a user interface for manipulating anchors and links, and an unbroken path for communication between the application and the open hypermedia system. Performing an integration of an external application with an open

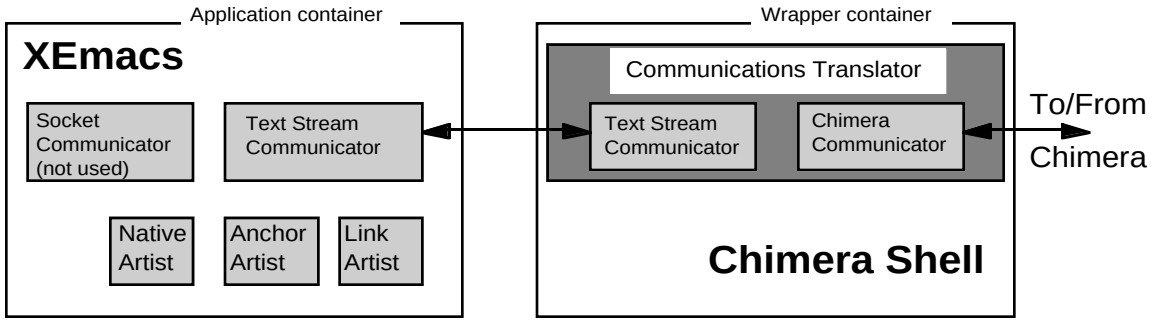

Figure 7: Architectural model of XEmacs/Chimera integration.

hypermedia system can be seen as involving the modeling of the initial state of the application using the architectural model, comparing this initial state model with the properties of a complete integration, and then supplying the missing elements. This uses the architectural model in a normative sense, describing what should be present once the integration is complete.

Example: XEmacs/Chimera. Continuing the example from the previous section, the task of performing an integration of XEmacs with Chimera can be described as creating anchor and link artists within XEmacs, and creating a communications translator to manage communications between either the Socket Communicator or the Text Stream Communicator and the native Chimera Communicator. For the final integration the Text Stream Communicator was used, and the Chimera Shell transforms text commands into Chimera RPC messages. Figure 7 shows the final integration architecture which is a depiction of the architecture of Figure 2 using the architectural model.

Example: XEmacs/HyperDisco. The integration of XEmacs with HyperDisco highlights the architectural variation which is possible when integrating even the same application. Shown in Figure 8, this integration is similar to the XEmacs/Chimera integration in that menus and control key sequences are used to access custom Emacs Lisp code which creates anchors and manipulates links. (In fact, there is some code reuse between these two integrations.) Hence anchor and link artists are created within XEmacs. However since the native protocol of HyperDisco is layered on top of sockets, it was easier to convert the XEmacs Socket Communicator into a native HyperDisco Communicator by writing custom Emacs Lisp code. This removed the need for a communications translator, while still satisfying the properties of a complete integration. Interestingly, the integration of XEmacs with DHM has the same architecture as the XEmacs/HyperDisco integration.

6. COMMON INTEGRATION ARCHITECTURES

A consideration of the many applications integrated with Microcosm, DeVise Hypermedia, HyperDisco, Chimera, and Sun's Link Service finds that application integration architectures fall into three categories, known as launch-only, wrapper, and custom. These architectural categories range from being very easy to implement (launch-only) to very time-consuming to implement (custom), with wrapper architectures spanning the range in between. For example, a Microcosm Universal Viewer integration is easy to implement, while writing a new wrapper is very time consuming. The three architectural categories of integrations are described below:

Launch-only. In a launch-only integration architecture, a non-communicative application is made to participate in the open hypermedia system. The main advantage is the ability to quickly integrate any application without modifying its binary image. The downside is being limited to a "large chunk" hypermedia effect, and only having link traversals which end at the application, but cannot be initiated from the application.

In this architecture, when a link traversal event is received, the application is invoked and instructed to open a specific file. In this case, an anchor is defined to be an entire file, resulting in the native artist being identical to the anchor artist. Link artist facilities are provided by either command-line options or by a separate tool. Since the application is non-communicative, its communications with the

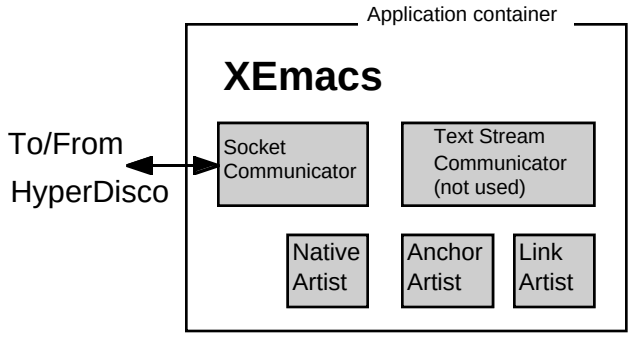

Figure 8: Architectural model of XEmacs/HyperDisco integration.

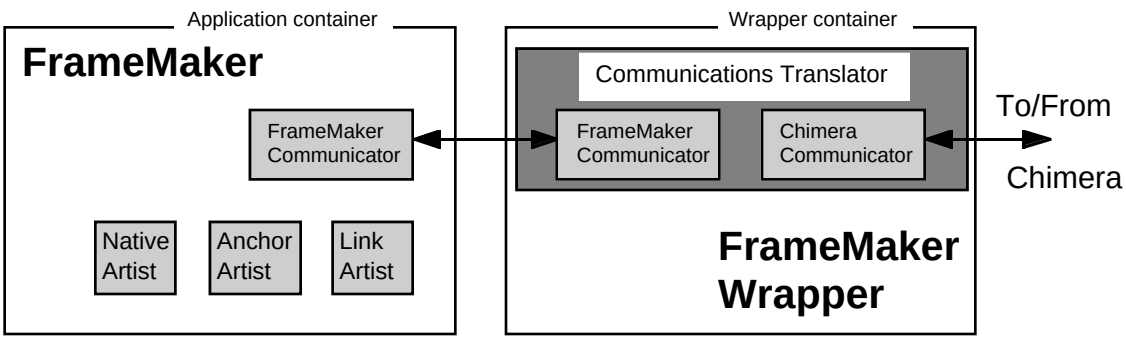

Figure 9: Architectural model of FrameMaker/Chimera integration.

hypermedia system must be handled by some other program.

Example. The HyperDisco system has integrated many non-communicative applications using a launch-only architecture. HyperDisco requires that the application be able to start on a particular file, and this file is the entire anchor. The HyperDisco XEmacs client has special menu options which are used to create links to launch-only applications, and hence acts as a communications mediator for them.

Wrapper. A wrapper (or shim) integration architecture contains a separate computational element, often a separate operating system process, which acts as an intermediary between the application and the hypermedia system. The main advantage of the wrapper architecture is it does not require modification to the application's binary image, since it uses the application's built-in hypermedia capability. By using the application's external API, the integration is isolated from changes to the application. Disadvantages are the translation processing, and potential extra process, hence communications process hop, associated with this architecture.

There are two common cases of wrapper architectures. In the first, a wrapper only contains a communications translator, which converts between the hypermedia functions offered by the application's external API, and the native functionality of the hypermedia system. In the second, a wrapper may contain a communications translator plus one or more artists when these artists are not present in the application. Applications integrated with the Microcosm Universal Viewer are an example of this approach.

Example: FrameMaker/Chimera. The FrameMaker/Chimera integration, shown in Figure 9, contains a communications translator which mediates between FrameMaker's native hypermedia functionality, accessed via its external API, and the hypermedia functionality accessed via Chimera's API. Since FrameMaker supported an internal notion of hypermedia, and an API which could manipulate it, the wrapper approach made it possible to integrate FrameMaker without modifying its binary image, and without having access to its source code. The wrapper approach has been similarly used to integrate Chimera with Mosaic using its Common Client Interface.

Custom. A custom integration consists of either modifying the source code of the application or writing code in an application's customization language, such as Emacs Lisp. Due to the large degree of control this provides over the application, a custom integration has the advantage of providing a very fine-grain level of hypermedia functionality, with anchors which can be smaller than a file. A custom integration also gives more control over the presentation of a link traversal, supporting such abilities as going directly to an anchor, or popping up a new window. The disadvantage of a custom integration is the possibility a new revision of the application will break the integration, requiring an update of the integration code.

When the application does not have any built-in hypermedia functionality in its user interface, the integration involves writing an anchor and a link artist. Since an application typically does not have a native communicator for the hypermedia system, it is also necessary to either create a new communicator, or tailor one of the application's existing communicators to become a hypermedia system native communicator.

Example: Word/DHM. The integration of Microsoft Word with the DeVise Hypermedia system is shown in Figure 10. Word's customization language, Visual Basic for Applications, was used to create a new toolbar for creating and manipulating new anchors and links, thus implementing the anchor and link artists. The native communications within DHM is based upon DDE, and Word has built-in functions for sending messages using DDE. As a result, it was easy to customize the DDE communicator as a DHM communicator. The DHM/Excel and Word/Microcosm integrations also have the same architecture.

Many other custom integrations have also been performed, such as Microstation (a computer aided design application) and XEmacs with DHM; xvi (a public domain vi-clone), MPEG movie player, xgif (a GIF viewer), and xmh (an email reader) with Chimera; Textedit (a text editor) with

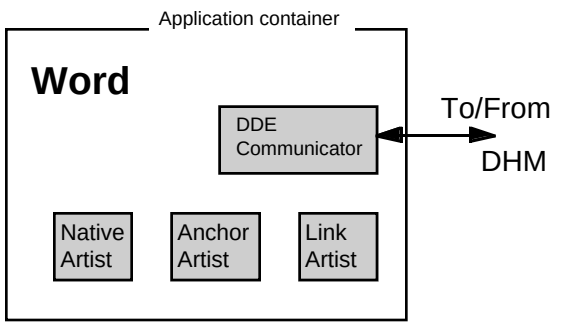

Figure 10: Architectural model of Word/DHM integration.

Sun's Link Service, and Emacs, XEmacs, and Epoch with HyperDisco.

Combination Architectures. Some integrations combine two architecture categories to gain the advantages of both, while mitigating their drawbacks. Instances of these combination architectures are not common. Combinations which have occurred to date are launch-only and wrapper, and wrapper and custom architectures. There have been no instances of a combination of launch-only and custom, probably because a custom integration implies either modifying source code or writing new capabilities in a customization language, which removes the key advantage of the launch-only approach, the rapid integration of an unmodified binary.

The XEmacs/Chimera integration is an example of a combination wrapper and custom integration. The Chimera Shell, which is a communications translator between text and native Chimera RPC, is the wrapper portion of the integration. The Emacs Lisp code which implements the anchor and link artists, and which manages communications with the Chimera Shell, constitute the custom part of the integration. Using the Chimera Shell instead of implementing the Chimera RPC protocol in Emacs Lisp provides greater isolation to changes in XEmacs, and also results in a tool which has far greater utility than just the XEmacs integration. The custom Emacs Lisp code which was written provides a very fine grain hypermedia capability which would have been unachievable with just a pure wrapper approach. Thus the combination integration gains the advantages of the wrapper architecture, isolating the exposure to most changes to a single, known communications protocol (the text commands sent to the Chimera Shell), while also providing fine-grain hypermedia capability, the advantage of the custom architecture.

7. ARCHITECTURE SELECTION GUIDANCE

When performing a new application integration, a pertinent question is what architecture should be chosen for the final integration, based on the characteristics of the application prior to integration. An analysis of existing integrations shows that the same final architecture is often chosen for a particular initial application architecture. Table 1 lists the integration architecture which results from an application's initial characteristics. This table has the application's initial communications capability on the columns, and existing artists on the rows. The table cells contain the final integration architectures which have been observed for applications with those initial characteristics. The table exhibits a trend of having applications with little initial knowledge of hypermedia appearing towards the lower right, and applications with full hypermedia knowledge appearing in the upper left corner.

The utility of this table varies. In the best case, when an application initially has only an anchor artist, along with a non-native communicator, it returns the unambiguous selection of a wrapper architecture. In the worst situations, such as the common no artists, no communicators case, the table shows that all architectures, and even some combinations have been used under these conditions. For this case, the table clearly doesn't offer much guidance.

Interestingly, there are many entries which do not describe any existing integration. There appear to be several reasons for this. First, the row representing the case where the application only has a link artist, but no anchor artist, is most likely unachievable, since a link artist is difficult to create without an anchor depiction to manipulate. Because of this, link artists universally come bundled with anchor artists. However, it is possible to conceive of an application which implemented computed links without having an anchor artist, and hence this row is not entirely impossible. The blank entries in the native communications column result from the unlikelihood of an application being fully communicative in a hypermedia system's native protocol without also having anchor and link artists. Native communications is always accompanied by anchor and link artists.

However, other blank table entries are not as pathological. An application possessing an internal notion of hypermedia but lacking an external API is descriptive of many closed hypermedia systems, such as KMS [1]. It is quite conceivable that a closed hypermedia system could have a custom integration with an open hypermedia system, but to the author's knowledge, no such integration has been performed to date. Interestingly, there are existing integrations with closed hypermedia systems containing an external API. Examples of these integrations are Microcosm with Toolbook, and Hyperform and HyperDisco with EHTS [19].

	NATIVE	NON-NATIVE	NON-COMMUNICATIVE
ANCHOR AND LINK	**No integration required, use as-is** *Chimera Hotlist, Sound Player.* *Microcosm Text Viewer, Graphics Viewer.* *DHM text component, drawing editor.* *SP3/HB3 & HOSS HyperEd, Sangam, Scholia* *SP2/HB2 HyperDraw, V text editor* *Multicard MCEditor, Emerald, FCEditor*	**Custom** *Microcosm/Toolbook, EHTS/Hyperform, EHTS/HyperDisco* **Wrapper** *FrameMaker/Chimera*	None observed
LINK ONLY	None observed	None observed	None observed
ANCHOR ONLY	None observed	**Wrapper** *Mosaic/Chimera*	None observed
NO ARTISTS	None observed	**Custom** *XEmacs/DHM, XEmacs/HyperDisco, Emacs/HyperDisco, Emacs/Hyperform, Epoch/HyperForm, Epoch/HyperDisco, Word/Microcosm, Word/DHM, Word/Multicard, Word Perfect/Microcosm, Lotus 123/Microcosm, AMI Pro/Microcosm, Microstation/DHM, AutoCAD/Microcosm, Excel/Microcosm, Excel/DHM, Authorware Pro/Microcosm, Access/Microcosm, Project/Microcosm,* **Wrapper/Custom** *XEmacs/Chimera*	**Custom** *Textedit/SLS, xvi/Chimera, MPEG_play/Chimera, xgif/Chimera, xmh/Chimera, Athena text widget/ SP1/HB1, xedit/SP1/HB1* **Wrapper** *Microcosm Universal Viewer integrations, vi/DHM* **Wrapper/Launch-only** *Chimera XRay integrations* **Launch-only** *HyperDisco (Idraw, Xdvi, PageView), DHM file component*

Table 1: Integration architecture resulting from application's pre-integration characteristics.

8. RELATIVE INTEGRATION EFFORT

The effort required to perform an application integration is directly related to its architecture. Experience has shown that, as a general rule, launch-only integrations are easier to perform than wrapper integrations, which in turn are easier that custom integrations. A launch-only integration can usually be performed in under an hour by an expert user of the hypermedia system. Wrapper integrations vary widely, from Universal Viewer integrations which also only require an hour for integration, up to many weeks when a new wrapper must be written from scratch, such as with the Chimera Shell. Wrappers thus run the gamut from being as easy as a launch-only integration, to being as difficult as a custom integration. Custom integrations can run from weeks to months, depending on the degree of knowledge about the application, expertise with the hypermedia system, and the granularity of hypermedia functionality desired.

In those cases where there are multiple choices for which integration architecture may be chosen based on a given set of initial application characteristics, the effort required to perform the integration can provide an extra discriminator for determining the actual integration architecture. If a non-communicative application, with no knowledge of hypermedia needs to be immediately used, a launch-only architecture is ideal. However, if fine-grain hypermedia capability is desired, and programmer resources and source code are available, a custom integration architecture provides the best results.

9. RELATED WORK

This paper has described integration architectures as being either launch-only, wrapper, or custom. Experience integrating applications with the Microcosm system, reported in [5], resulted in an enumeration of six separate categories of integration levels, "tailor made viewers," "source code adaptation," "object-oriented reuse," "application interface level adaptation," "shim or proxy programs," and "launch-only viewers." The first four of these cases are considered custom integrations, since they all involve writing code that either modifies the application's executable image or is written in an application's customization language. The Microcosm "shim," or "proxy" is identical to our notion of wrapper, and the Microcosm notion of "launch-only viewer" is identical to our notion of a launch-only integration. While the work in this paper tends to confirm the Microcosm experience, the presentation in this paper differs markedly from the Microcosm approach by emphasizing the use of the novel architectural model when describing these common integration architectures.

Integrations with the HyperDisco system also fall within three categories, which are termed "Full Integration," "Partial Integration," and "No Integration." These integrations are characterized based on what hypermedia system facilities they use: full integration uses both storage and hypermedia capabilities, partial integration uses only hypermedia capabilities, and no hypermedia system capabilities are used by a "no integration" application. These facilities-used integration categories contrast with the architectural categories used in this paper, and also in [5].

The granularity of the architectural model presented in this paper is rather coarse. This contrasts with [8], where a more fine-grain, object-oriented architectural framework is used when performing custom application integrations with the DHM system. This picks up where this paper leaves off. Using the architectural model presented in this paper, a developer could decide which type of integration architecture they wish to perform. If they decide on a custom integration, they could then use the Dexter-based object-oriented architecture presented in [8].

The elements of the architectural model have been previously identified. Descriptions of issues encountered when integrating applications with Sun's Link Service cover identification of elements of an application which may serve as anchors, similar to the notion of an anchor artist presented in this paper. In [10] which also described integration with Microcosm, a distinction is made between the three communication cases, "fully aware," "partially aware," and "unaware," which correspond directly to the cases, "native," "non-native," and "non-communicative" used herein. [10] also discusses the need for anchor selection and highlighting, and the need for a menu to control linking, which corresponds to the notions of anchor and link artist. However, this paper is the first instance where anchor and link artists, along with communication ability are both used to characterize an application prior to integration, and to group final integration architectures.

World-Wide Web (WWW) [3] browsers fit the model presented in this paper. A browser such as Mosaic or Netscape can be viewed as a native application which contains both anchor and link artists, and is fully communicative in the hypermedia system's native protocol, the HyperText Transfer Protocol [6]. Early browsers shared the assumption of open hypermedia systems that a particular object type would be displayed by its native application if it was not displayable by the browser, and most browsers contained facilities for performing launch-only integrations of the requisite native applications. These launch-only integrations differ from those in open hypermedia systems, since the WWW client performs the application launching, rather than the open hypermedia system itself. This is similar to the Chimera system, where a special client for each user, known as the Process Invoker, invokes applications needed to complete a link traversal. More recent browsers have a plug-in architecture which are specialized for the display of a given object type, thus removing the need

for a launch-only integration of an application which can display objects of that type.

In the software development environment community, represented by the consensus view described in [13], application integration is also an important issue. In [13], five dimensions of integration are identified: data, control, presentation, process, and framework. Integration between an application and a linkbase system is primarily a control and presentation integration problem. Integrations with open hyperbase systems also introduce data integration aspects. In the architectural model, control integration is handled by communicators, and presentation integration is managed by the anchor and link artists.

10. FUTURE WORK

Integrations with link server style systems are well described by the architectural model. Less well described are integrations with open hyperbase systems, due to the data integration that must be performed in addition to the control and user interface integration concerns handled by the model. Thus, there is a need to extend the model to adequately describe the integration of an application with the data storage capabilities of an open hyperbase system, and to examine how data integration affects the final integration architecture.

While this paper has limited its scope to an examination of integrations with open hypermedia systems, future examinations will focus on extending the model to other domains with large amounts of integrations, such as software engineering environments, and document management systems. These extensions would involve extending the range of artist types, and providing a general facility for modeling data integration. A framework for describing application integrations across several domains would be extremely powerful, and would unify the integration experience of several disciplines.

11. CONCLUSIONS

An architectural framework for open hypermedia client integrations has been presented. This framework has utility in modeling architectural elements of an application, and thus characterizing it prior to integration. This same framework has been used to provide a normative description of architectural elements applications should contain after integration. The model allows past integration experience to be collected in a standard way, and used as guidance in selecting a final integration architecture. The degree of difficulty of an integration, related to the final integration architecture, may inform the selection decision for the integration architecture. While the model does not describe the data integration aspects of integrations with open hyperbase systems, the framework nevertheless provides a useful categorization for existing work in open hypermedia application integration.

ACKNOWLEDGMENTS

My understanding of these issues has been an emergent process, based on discussions with many researchers, all of whom have my gratitude, some deserving special mention. Hugh Davis, Kaj Grønbæk, Peter Nürnberg, Antoine Rizk, Francis Malezieux and Uffe Wiil were extremely helpful answering my detailed questions about their integration experience. I also thank Hugh Davis for allowing the reuse of a figure from [5]. Discussions with Uffe Wiil helped refine these ideas, especially his insight into combination architectures. Ken Anderson reviewed an earlier draft of this paper. Greg Bolcer, Rebecca Grinter, Hadar Ziv, and Nancy Eickelmann all offered insightful early feedback. I am grateful to Richard Taylor for his discerning feedback and support during this research.

This material is based upon work sponsored by the Air Force Materiel Command and the Defense Advanced Research Projects Agency under Contract Number F30602-94-C-0218. This material is also based upon work sponsored by the State of California and the Northrop Corporation via the MICRO program. The content of the information does not necessarily reflect the position or the policy of any Government, or the Northrop Corporation, and no official endorsement should be inferred.

REFERENCES

[1] R. M. Akscyn, D. L. McCracken, E. A. Yoder. KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations. *Communications of the ACM*, 31(7):820-835, July, 1988.

[2] K. M. Anderson, R. N. Taylor, E. J. Whitehead, Jr., Chimera: Hypertext for Heterogeneous Software Environments. In *Proceedings of the European Conference on Hypermedia Technology 1994, ECHT94*, pages 94-107, Edinburgh, Scotland, September 1994.

[3] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, A. Secret. The World-Wide Web. *Communications of the ACM*, 37(8):76-82, August, 1994.

[4] H. Davis, W. Hall, I. Heath, G. Hill, and R. Wilkins. Towards an Integrated Information Environment with Open Hypermedia Systems. In *Proceedings of the European Conference on Hypermedia Technology 1992, ECHT92*, pages 181-190, Milano, Italy, November, 1992.

[5] H. Davis, S. Knight, and W. Hall. Light Hypermedia Link Services: A Study of Third Party Application Integration. In *Proceedings of the European Conference on Hypermedia Technology 1994, ECHT94*, pages 41-50, Edinburgh, Scotland, September 1994.

[6] R. Fielding, J. Gettys, J. C. Mogul, H. F. Nielsen, T. Berners-Lee. *Hypertext Transfer Protocol -- HTTP/1.1*, RFC 2068, U.C. Irvine, January, 1997.

[7] A. M. Fountain, W. Hall, I. Heath, H. C. Davis. MICROCOSM: An Open Model for Hypermedia with Dynamic Linking, *in* A. Rizk, N. Streitz and J. Andre *eds. Hypertext: Concepts, Systems and Applications. The Proceedings of the European Conference on Hypertext*, INRIA, France. Cambridge University Press. 1990.

[8] K. Grønbæk, J. Malhotra. Building Tailorable Hypermedia Systems: the embedded-interpreter approach. In *Proceedings of OOPSLA'94*, pages 85-101, Portland, Oregon, 1994.

[9] K. Grønbæk, R. Trigg. Design Issues for a Dexter-Based Hypermedia System. *Communications of the ACM*, 37(2):40-49, February, 1994.

[10] W. Hall, G. Hill, and H. Davis. The Microcosm Link Service. Technical Briefing. In *Proceedings of Hypertext'93*, pages 256-259, Seattle, Washington, November, 1993.

[11] J. J. Leggett, J. L. Schnase. Viewing Dexter with Open Eyes. *Communications of the ACM*, 37(2):76-86, February, 1994.

[12] B. A. Myers. Incense: A System for Displaying Data Structures. *Computer Graphics*, 17(3):115-125, July, 1983.

[13] Computer Systems Laboratory, National Institute of Standards and Technology (NIST), and European Computer Manufacturers Association (ECMA). *Reference Model for Frameworks of Software Engineering Environments*, 3rd ed. Technical Report NIST 500-211, ECMA TR/55, 1993.

[14] P. J. Nürnberg, J. J. Leggett, E. R. Schneider, J. L. Schnase. Hypermedia Operating Systems: A New Paradigm for Computing. In *Proceedings of Hypertext'96*, pages 194-202, Washington, DC, March, 1996.

[15] K. Østerbye, U. Wiil. The Flag Taxonomy of Open Hypermedia Systems. In *Proceedings of Hypertext'96*, pages 129-139, Washington, DC, March, 1996.

[16] A. Pearl. Sun's Link Service: A Protocol for Open Linking. In *Proceedings of Hypertext'89*, pages 137-146, Pittsburgh, PA, November, 1989.

[17] A. Rizk, L. Sauter. Multicard: An open hypermedia System. In *Proceedings of the European Conference on Hypermedia Technology 1992, ECHT92*, pages 4-10, Milano, Italy, November, 1992.

[18] J. L. Schnase, J. J. Leggett, D. L. Hicks, P. J. Nürnberg, J. A. Sanchez. Open Architectures for Integrated, Hypermedia-Based Information Systems. In *Proc. HICSS '94*, January, 1994.

[19] U. K. Wiil. Issues in the Design of EHTS: A Multiuser Hypertext System for Collaboration. In *Proceedings of HICSS-25*, pages 629-639, 1992.

[20] U. K. Wiil, J. J. Leggett. Hyperform: Using Extensibility to Develop Dynamic, Open, and Distributed Hypertext Systems. In *Proceedings of the European Conference on Hypermedia Technology 1992, ECHT92*, pages 251-261, Milano, Italy, November, 1992.

[21] U. K. Wiil, J. J. Leggett. The HyperDisco Approach to Open Hypermedia Systems. In *Proceedings of Hypertext'96*, pages 140-148, Washington, DC, March, 1996.