An Introduction to the Web Dynpro Protocol

Dipak Chopra
Standards Architect
SAP NetWeaver Industry Standards
Table of Contents

Overview ........................................................................................................ 4
Introduction and Approach ........................................................................... 4
Evolution of SAP GUI .................................................................................... 4
Lessons Learned ............................................................................................ 5
  Issues ........................................................................................................... 5
Industry Initiatives .......................................................................................... 6
  W3C XForms 1.0 ........................................................................................... 6
  Other Technologies ....................................................................................... 7
Web Dynpro Protocol: A New GUI Approach ................................................ 7
Technical Overview ......................................................................................... 8
  UI Object Model .......................................................................................... 8
  Session Lifecycle ......................................................................................... 10
Future Direction ............................................................................................. 11
References ....................................................................................................... 11
Overview

Since the first GUI (Graphical User Interface) demonstration by Douglas Englebart in 1968, GUI technologies have always evolved from Smalltalk/Alto simple screens to Windows and Mac feature-rich screens barring one disruption, the Internet! The rich GUI interface got eclipsed by an impoverished and quirky client known as browser. Although it created new business opportunities, it scaled down user experience and productivity. Business users were compelled to move from rich desktop clients to browser-based clients. In last couple of years, the industry has been working on new initiatives to enable rich web-enabled clients. Being the leading business application vendor, SAP has developed Web Dynpro technology to enhance the user experience by enabling rich clients.

This article will outline issues with HTTP-based clients, highlight salient features of the Web Dynpro Protocol, and compare it with other industry initiatives.

Introduction and Approach

It is assumed that the reader has basic knowledge of GUI and HTTP technologies. The intended audience for this article is GUI tool vendors and developers interested in the details of Web Dynpro runtime behavior. This article does not deal with design-time aspects of Web Dynpro.

The Web Dynpro Protocol is an enabling technology that provides efficient and adaptable HTTP-based exchange of UI (User Interface) instances between client and server. Enabling such exchange between different types of clients and servers poses a number of challenges.

This paper outlines issues with HTTP-based clients, presents solutions provided by current industry initiatives and explains how the SAP Web Dynpro Protocol addresses these issues. This article should give a better understanding through which the reader will appreciate Web Dynpro technology even more!

Evolution of SAP GUI

SAP R/2 (which was based on mainframe system) introduced the first UI for SAP applications in 1979. This very first UI (Figure 1) was based on terminal screens. The second step in the GUI evolution was SAP R/3 1.0, when the first GUI for SAP applications was introduced. This initial GUI was based on OSF/Motif.

A major milestone was SAP R/3 2.0, which introduced Windows GUI. With this version, the Windows GUI started to dominate the SAP user interface. Many users believed that this GUI was a "real" Windows application; however, it’s really a "clever" terminal simulation. The next major milestone was R/3 4.6, when the GUI was "reinvented". In the evolution of SAP GUI, the next big change came when R/3
became web-enabled with mySAP, which introduced mySAP Workplace (Picture 2). The mySAP.com Workplace is the predecessor to SAP Portals UI.

Lessons Learned
Throughout the GUI evolution, there were many valuable lessons learned. With the inevitable HTTP enabling of clients, new issues came into the picture, which eventually lead to a new GUI technology, Web Dynpro.

Issues
Multiple Clients: With the advent of HTTP based clients, business applications have become accessible from multiple devices, e.g. desktops, PDAs, mobile, browsers. Although this presents a great business opportunity, it poses a challenge to cater to the needs to different clients. Should one create multiple web end points to cater to multiple devices? Should the web application identify the client and respond accordingly?

Performance: In a typical web application, a page refresh is required to get even a small piece of new data. This puts an unnecessary load on the network traffic and client/server processing. For example, a user makes a selection in one combo box, which results in new entries for another combo box. Instead of getting just the data that needs to be displayed in the second combo box, one needs to retrieve the whole page. To make matters worse, if the set of values (for the second combo box) is too large, and the UI would display only a very small subset (due to UI limitations), an unnecessary load would be put on the system to retrieve the complete set when only a fraction of it would actually be used.

UI and Data Mashing: In HTML model, presentation and data are mingled, which is a bad design as it necessitates the exchange of both, even if only one has changed. This requires unnecessary development efforts to handle multiple clients. Ideally one should be able to specify the data separately from different UIs and bind the UI to the data and vice-versa.

User Experience and Productivity Degradation: User experience on the internet may be quite different for a casual reader and a business application user. A business application user expects the same user experience on the internet as with their rich desktop clients; instant loading, no flickering, rich and intuitive screens and less navigation to achieve the same results. First generation UIs based on HTTP are limited; visualization is not as rich as needed in some cases, and performance suffers due to the need for multiple round trips to the server to execute simple tasks. This slows down productivity as users wait for each command to be executed.
Incompatible Clients and Server Applications: Due to the relatively longer life of business applications (which is linked to the higher cost on the server side) and constant evolution of GUI clients, there are instances when clients and server applications do not support the same UI functionality. For example, a new GUI client is developed to target a new business application server with latest GUI features but this new client should also be able to access older business application servers. There is no standard way one can set the GUI features for an HTTP session between a client and server.

Dynamic Behavior of HTTP Channel: As one can imagine, clients and servers are communicating over a crowded HTTP channel. Depending upon runtime changes, the HTTP communication needs to be tuned. The client and server may also develop heuristics depending on the speed of connection (LAN or WAN), or on the complexity of the application. A server could send more, and a client could request more information than what is currently visible (where the user may navigate to) without the need for a roundtrip. For example, a data collection may have a large number of data elements. A client can request for a small number or a larger number of data elements depending upon the network speed.

Industry Initiatives
In the industry, there are many GUI initiatives going on over last couple of years.

- W3C XForms 1.0
- Mozilla XUL (XML User Interface Language)
- AJAX (Asynchronous JavaScript and XML)

W3C XForms 1.0
XForms 1.0 is a W3C Recommendation. XForms are the successor to HTML forms, and benefit from the lessons learned from HTML forms. XForms are designed to cater to the needs of rich internet applications although it can also be used for static web pages. XForms provide a richer, reusable, and device independent way of handling web input.

Figure 3: XForms Framework

XForms is not a free-standing document type, but is intended to be integrated into other markup languages, such as XHTML or SVG. XForms use XML for data definition and HTML or XHTML for data display. Separating data from presentation helps makes XForms device independent. The presentation can be customized for different user interfaces, like mobile phones, handheld devices, and Braille readers for the blind.

XForms enables a rich GUI client. XForms supports a large number of GUI components (more than thirty components in XForms 1.0). These components are abstract and they can be rendered as appropriate. For example, a user interface can have a \(<select1>\) component instead of hard coding it to a radio
XForms specifies a model that defines XML instance data (which is exchanged with the server in the HTTP Request). The UI controls are bound to the XForms model via the XForms binding mechanism. XForms allow defining actions (similar to event handlers) that occur whenever the event is fired for a particular UI component. One can also bind events to the model so that when a given value in the model changes, it triggers an event that can be caught. The only difference is that the event handlers, the event notifiers, and everything else are built with XML; one can handle processing with XPath expressions, describe the model using XML Schema, and fire events using the XML events architecture. For example, “xforms-value-changed” notification event is triggered when the value of a form control is changed.

Once the XForms is submitted using the XForms Submit Protocol, it is delivered to the processing application as XML instance data. This means that the data that has been entered into the form is packaged as an XML document representing that particular instance of the form. This allows the processing application to look at the form data as an XML document, rather than a group of unrelated data. This obviates the need for custom server-side logic to marshal the submitted data to the application back-end. The received XML instance document can be directly validated and processed by the application back-end.

Although XForms 1.0 became W3C Recommendation in 2003, usage has not increased significantly.

Other Technologies

There are two other GUI technologies that, although they are not standards have gained some attention in the industry.

**XUL** (XML User Interface Language) is another technology in the GUI domain. XUL (pronounced "zool") is Mozilla's XML-based user interface language that lets the developers build feature-rich cross platform applications. The goal of XUL is to build cross platform applications, in contrast to DHTML which is intended for developing web pages. For this reason, XUL is oriented toward application artifacts such as windows, labels and buttons instead of pages, heading levels, and hypertext links. Since XUL provides an abstraction of user interface components, it delivers on the promise of write-once, run-anywhere. XUL specifies the UI model in XML. It specifies a relatively rich set of UI controls.

**AJAX** (Asynchronous JavaScript and XML) is another technology that has received lot of attention from the industry. There are some very interesting examples of AJAX applications, e.g. Google Maps, Google Suggest. In reality AJAX is not a new technology but a combination of existing technologies (JavaScript, DOM, XML). It enables the creation of custom, rich clients that also reduce network traffic and server usage. AJAX effectively moves some of the computing from server to client, which may not work for light clients like mobile devices. One major difference between AJAX and XForms/XUL is that AJAX does not specify or refer to any standard UI and data model. Clients and servers can have their own proprietary object model, which they can exchange via XML over HTTP and render using JavaScript and DOM.

Web Dynpro Protocol: A New GUI Approach

After evaluating the industry initiatives, it was quite evident that for demanding business web applications, existing GUI technologies were not enough, which lead to the next milestone in the evolution of SAP GUI: Web Dynpro. Web Dynpro caters to the needs of sophisticated, rich web applications. Web Dynpro Protocol is about the runtime aspect of Web Dynpro technology.

The first generation of the web was as a presenter of static information with links to more static information, which focused on one-directional (server to client) information flow. When the same GUI model of static web pages is taken to the business web applications, a number of issues (as described earlier) come into the picture. In order to address the issues with the first generation HTTP clients, a new GUI ap-
proach is required. The core objective is to realize the features of rich clients while keeping the advantages of web based applications.

One of the important lessons learned in last couple of years is the decoupling of different layers to respond quickly to changing business environment. The invention of database driver methodologies such as JDBC and ODBC led to applications being loosely coupled with their back end databases, allowing best-of-breed databases to be chosen—and then swapped out when necessary. At the UI level, there are three layers; presentation, data and eventing. In a typical HTML based web page, these layers, especially presentation and data, are intermingled. By decoupling these layers, one can modify the layers independently of each other to meet different scenarios (e.g. different client devices).

XML is the ideal choice for specifying the UI model. XML can be used to define presentation, data and eventing separately and bindings to link these layers. Also XML is inherently platform-independent. The same client can interact with applications deployed on ABAP and Java server, which support the same XML object model. One important feature of XML model is the ability to work at sub-document level, which opens new opportunities. Client and server can exchange only the required information, which may be a subset of the whole instance.

The new GUI model needs to add intelligence to clients and servers so that they can negotiate a common set of GUI features and adapt accordingly. A mechanism is required between clients and servers to identify the capabilities of each other and use it for the rest of HTTP session. This is important as new clients should be able to access old application servers. For example, SAP GUI for Windows still works with R/3 3.1 and even R/2 systems.

It is important to remove the data redundancy of the UI object model exchange between clients and servers. It should be possible to exchange delta updates. For example, if a client needs to get partial data (to be displayed in a pop-up menu) it should be able to request only that piece of data from the server.

In business applications, clients need to deal with large amounts of data. The dataset might be too large for a client that can only show partial data, for example in tree and table structures. The client should be able to request partial data in chunks depending upon the user action. For example, if a table displays only five line items, there is no need to retrieve all of the line items from the server. The client can retrieve more line items when the user scrolls down the table.

**Technical Overview**

The Web Dynpro Protocol is an HTTP based protocol, which sends XML as its body. The main purpose of the protocol is to synchronize a server instance of a UI (User Interface) object tree with the client instance. The client request and server response refer to the same UI object model.

**UI Object Model**

The UI object model uses an XML representation to specify data, view and eventing objects for HTTP request and responses. There is a clear separation between data, data types, view and eventing. The object model is defined in different XML schemas.

The root element of the UI object model is `<SCXML>`, which serves as a container for the complete UI instance.
<SCXML> consists of XML fragments for specifying client UI and interaction.

1. **Header** specifies the client’s information like date format, language and group separator. Besides that it may also provide statistics information for client (in HTTP request) and server (in HTTP response).
2. **Windows** specifies the client application’s windows. It is important to notice that a window created does not contain any UI elements except toolbar. One can specify a hierarchy of windows with their locations on the client side. For a client, there is always one root window known as an application window.
3. **Views** is a container for UI components. A view is linked to a window to display it on the client. Object model provides a rich UI functionality.
4. **Data** provides the data linked to UI elements. This provides a clear separation between the presentation (which is represented by Windows and Views) and the data.
5. **ContainerTypes** specify the complex type structure of the data containers within the Data fragment referring to simple types defined in the SimpleTypes fragment.
6. **SimpleTypes** specify simple types, e.g. value sets for enumerated types.
7. **Messages** specify the user messages that are displayed depending upon some user interaction with the client. Allowed message types are error, warning, fatal error and information.
8. **Actions** specifies the user actions on the client. An action is defined for a UI component. For example, an action can be defined for a button to submit the request.
9. **Automation** is required between client and server to handle invocations of different types, e.g. EventHandler, WindowHandler, DataHandler, ActionHandler, PageHandler etc. An object tree instance can have a queue of invocations targeted to handlers, which allows sending of multiple invocations in the same HTTP request or response. For example, if a user is viewing a tree structure, and she clicks on an expandable node that has un-retrieved child elements, the client sends an automation invocation to server. This invocation is processed by the server side PageHandler and the response is sent back to the client with the resultant child nodes.
Session Lifecycle

For the complete HTTP session between client and server, there are five stages.

**First stage “Server Identification”** is initiated explicitly by a user, which uses a client (e.g. browser, SAP GUI for Windows etc) to connect to a Web Dynpro application. The client needs to establish whether or not the web application supports the Web Dynpro Protocol. The client uses cheap HTTP HEAD request to check if the server supports the Web Dynpro Protocol.

**Second stage “Client Identification”** The objective for “Client Identification” is to find out the basic information about the client. For example, if the client identified is a pure browser, the rest of the exchange is based upon HTML and if the client is a rich client like SAP Smart Client for Windows (for Java), the rest of the exchange is based upon SCXML (to exchange object tree instances).

This allows the handling of multiple clients using the same protocol. The server can generate the object tree depending upon the client.

**Third stage “GUI Features Negotiation”** is about the setting the behavior between client and server. There are many optional GUI features that are enabled using Web Dynpro Object model extensibility. The extensions are managed by using “support bits” feature. For example, one feature is caching of whole objects on the client side. The setting of GUI features for the session is important because the clients can change more quickly than servers. Using support bits, clients and servers can negotiate what features can be used between a particular client and server. Support bits negotiation happens at the starting of the session, and the resultant GUI feature set remains the same for the rest of that session, with the exception of the dynamic GUI features which might be switched on or off within a session like client and server statistics.

**In the fourth stage “Request / Response Processing”,** client and server exchange partial object tree instances. A client sends a request to the server with a partial object tree instance, describing data changes because of user input or events because of user activities like pressing a button. The server processes the request and responds with a success or error code, followed by a partial object tree instance now containing changes of the UI object model caused by application logic like modified views or data. The life time of the object tree instance is a session, which starts with an appropriate initial client request and is ended by an appropriate client request or server response. During the session the object tree is maintained on both sides by client and server. While the client is only able to modify the content of the tree, the server in addition is in charge of managing the lifecycle of each object by adding new objects or removing the existing ones.

Sharing the same object model enables many features that are important for modern day UI clients. It enables the delta synchronization between client and server. Clients need only send that data to servers that has changed and vice-versa. This results in a significant performance gain.

In Web Dynpro version 2.0 with no additional support bits, the server will serialize and send the whole instance of its currently accessible runtime object model in each response. This behaviour is required for an eCATT client, which will record and later replay a session for testing purposes.
The client will always send back only those attributes which have been changed by a user. The changes are serialized in a delta format and merged back by the server into its instance.

To handle large data collections for tree and table structures, a paging feature is provided as an extension feature. To support the paging feature, both client and server need to support the corresponding support bits. The client requests for a “specific number” of elements. One of the ways to determine this number is by using statistics (provided by the Header segment in the object tree instance). For example, if network speed is slow, the client can request fewer elements.

Further support bits of the current version allow delta management, e.g. simple and container types. As long as types do not change, they are only sent once per session. Also the server notifies the client, whenever data types are no longer needed and can be removed from the session state.

**Fifth stage “Session Termination”** is triggered either by client or server (in the case of errors). This results into destroying the object tree instances for the session on both sides.

**Future Direction**

Web Dynpro is a major milestone in SAP’s GUI evolution. As happens with any technology, Web Dynpro will continue its evolution. The core framework of the Web Dynpro Protocol is sound. The protocol will be extended using support bits mechanism. Some of the potential extension features are persistent caching of whole objects like views, types or even data containers, which are uniquely identified by MD5 keys on the server side.

In the industry, the general consensus about the need to create a new rich-client GUI technology is growing. W3C is proposing to start a new “Rich Web Client Activity”, which will create a new standard for enabling rich web applications.

**References**

1. A History of the GUI
2. Introduction to Smalltalk/Alto
3. SAP Web Dynpro
4. Hypertext Transfer Protocol – HTTP/1.1
5. eCATT: extended Computer Aided Test Tool
6. W3C XForms
7. XForms Implementations
8. XML User Interface Language 1.0 (XUL)
9. AJAX: A New Approach to Web Applications
10. AJAX with J2EE