

# *Asperon AppProjector™ Presentation Server*

## **Technical Overview**

### **Introduction:**

The Asperon™ AppProjector™ is a complete system for delivering enterprise business applications over the web. It significantly reduces the time and cost of developing web applications and provides a rich user interface to those applications. Applications developed with the AppProjector™ can run natively in Internet Explorer and most other browser without a plug-in. The AppProjector™ can be thought of as an alternative to Java Server Pages (JSPs) or Struts that enables fine-grained user interaction, while still requiring only server-side application development. Because the user interface and client-server communication is handled entirely by the AppProjector™, the application developer can be freed from the concern of client-side compatibility while benefiting from the richness and interactivity of a desktop-like web application.

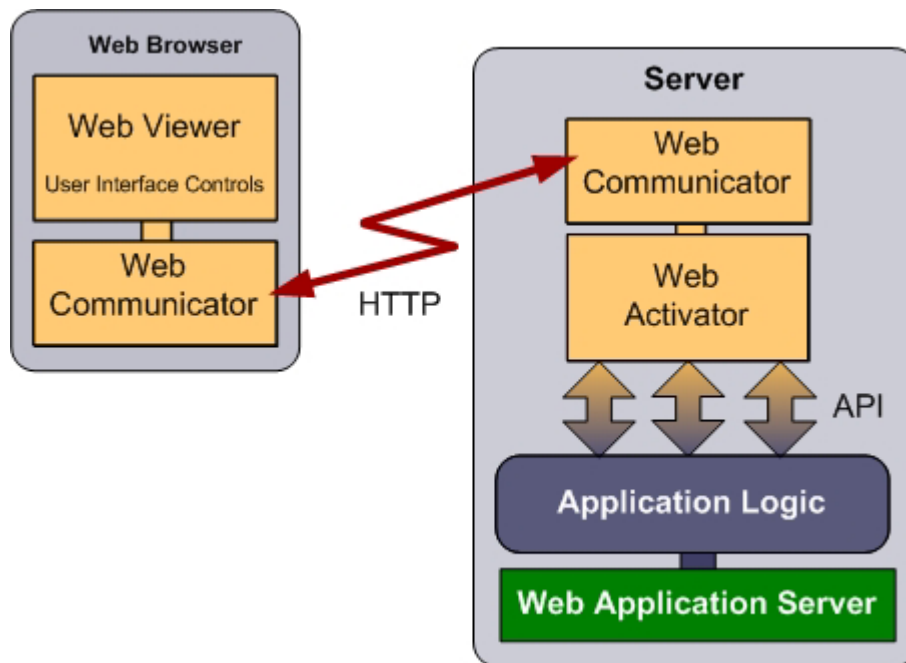


Fig. 1. Asperon™ AppProjector™ architecture.

### **AppProjector™ Architecture:**

Figure 1 shows the AppProjector™ architecture. The AppProjector™ consists of a client portion for displaying the user interface called the **Web Viewer™**, a server portion for running the business logic called the **Web Activator™**, and a communications layer called the **Web Communicator™** that exists partially on the client and partially on the server. The **Web Viewer™** has a set of user interface controls, a rendering engine that interprets

descriptions of the screens that are provided in XML by the application developer, and a client-side data buffer for caching data on the client. The client portion of the Web Communicator™ is a communications mechanism for connecting the Web Viewer™ client to the Web Activator™ server. The entire client is downloaded to the end user's browser automatically when the user browses to the web page that contains the application. The client is designed to support user interface controls rendered using the Asperon KWT, which is a highly compatible user interface component set that eliminates the need for a plug-in for Internet Explorer and most other browsers.

The server consists of the part of the Web Communicator™ that talks to the client, and the Web Activator™. The Web Activator™ has a controller for handling user inputs, and an application model that serves as the interface to the application logic, and manages all of the user sessions as remote users log in to the server. In all, the AppProjector™ components form a model-view-controller system with the application as the model, and the AppProjector™ acting as the view and controller.

The AppProjector™ client is completely generic, in that the same client applet is used for any application. This makes it easy to deploy changes to the user interface without having to update the client. The user interface is specified as a set of screens that are represented in XML, and read in by the Web Activator™ server. These screens are automatically delivered to the client through the Web Communicator™ communications layer at run time.

The Web Communicator™ sends messages back and forth between the client and the server using a binary encrypted form XML. It uses HTTP tunneling with the browser's proxy settings so that the Web Communicator™ can be used through corporate firewalls without forcing the user to set up a new proxy server configuration.

### **Application Model:**

Figure 2 shows the AppProjector™ application model. An AppProjector™ application is a Java class that implements the *AppProjectorApplication* interface. This interface defines a number of other Java interfaces that represent a business application. It consists of the following parts:

- 1 Screens** – These are a set of XML files that define the screens, with one screen per file. Screens describe the types and positions of all user interface controls, and define the way these controls bind to the runtime data. The XML files that represent screens are stored on the server and are automatically downloaded to the client by the AppProjector™ as they are needed.
- 2 Appearances** – These are a set of XML files that define sets of common styles used to represent various user interface controls. At any time, one of the appearances is designated as the current appearance, and this appearance controls the appearance of all the screens in the user interface. If no appearance is specified the "default" appearance will be used.
- 3 Data Buffers** – These are a set of server-side buffers that cache the runtime data that is to remain persistent when the user exits the application. They are normally used to represent the business objects (entities) in the application, and are implemented by the application developer. Data Buffers come in two types: Table data buffers for tabular data, and Tree data buffers for hierarchical data.
- 4 Control Blocks** – These are a set of associated sets of parameters that are used to control the runtime state of the application. Control blocks typically represent transient (session) information that is lost when the client exits.

- 5 **Commands** – These are a set of application specific commands that are defined mostly by the application developer. They implement the application business logic, and are bound to buttons and menu items, so that when a user presses a button or selects a menu item, the associated command is executed on the server.

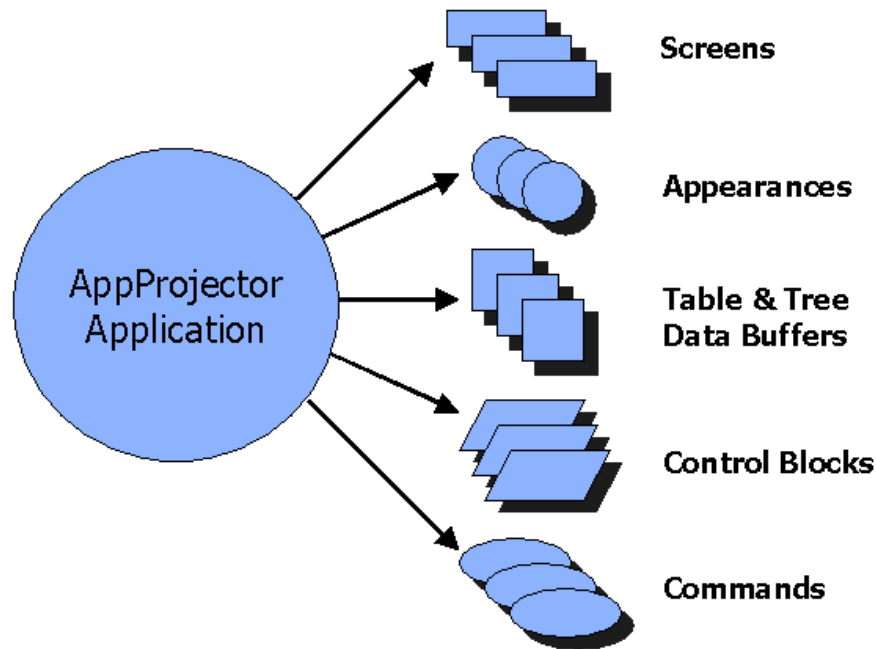


Fig. 2. The AppProjector application model.

### Implementation Procedure:

To develop an AppProjector™ application, the application developer needs to define and implement a set of data buffers in Java to represent the business objects of the application, define a set of screens in XML to represent the user interface, and define a set of application-specific commands and control blocks in Java that implement the work done by the application.

#### 1. Define Data Buffers:

Data buffers store the persistent data in the application. The most common type of data buffer is the Table Data Buffer. This represents data as a set of rows and columns. A table data buffer maintains a "current" row and must inform its "current row listeners" whenever the current row changes. The current row is distinguished from the other rows in that only the values in the columns of the current row can be changed, although values of any row can be displayed. When a scalar control such as a text field is bound to a table data buffer it is bound only to the value of the column at the current row. However, table-aware controls such as a UI table can be bound to the entire table at once so long as they indicate the current row.

A table data buffer is defined by defining its columns. Each table column has a name, a type of data to be stored in the column (e.g. Integer , Boolean, or String), an optional default value, and a "validator". A validator is a Java interface that is implemented by the application developer. It is passed a value to validate, and determines if that value is "valid" for the associated column of the current row. Validators may also cast the value that was entered into a format that is consistent with the type of data to be stored in the associated column, and may perform additional functions when the value of a column is about to change. This allows the user to define actions that will occur whenever a new value is set. Listing 1 shows the *Validator* interface.

```
public interface Validator
{
    /** Validates the indicated value.
     * The process of validation may involve coercing the value to another
     * format or to the nearest value in some set of values.
     * Therefore the object returned should be used as the valid value.
     * @param val The value to validate.
     * @return The possibly coerced valid value.
     * @exception com.asperon.util.ValidationException if the value is invalid.
     */
    public Object validate(Object val) throws ValidationException;
}
```

Listing 1, the validator interface.

Asperon provides abstract implementations of the different types of data buffers that can be subclassed to speed up development of the application.

## 2. Define Screens:

Screens are represented in XML in a format that is very similar to HTML. Each screen is uniquely named and consists of a set of controls (represented as XML entities), each of which has a set of attributes. The attributes of each control define the component's appearance, where it is positioned on the screen, and to which data the control is bound. Once a control is bound to a data buffer, the data in the control will be automatically synchronized with its data buffer by the AppProjector™. Component positioning is performed using a "table-style" layout that lays out components in a 2-dimensional array of rows and columns on the screen. Listing 2 shows an example of a screen that contains a table of contacts that is bound to the "contactsTable" data buffer.

```
<!DOCTYPE screen SYSTEM "../dtd/screen.dtd" >

<screen name="tableScreen" title="Table test screen">
  <components>
    <label name="space1" gridx="0" gridy="0" text=" " />
    <label name="space2" gridx="2" gridy="2" text=" " />

    <table name="table1" gridx="1" gridy="1" dataSource="contactsTable"
           style="greenBorderTableStyle">
      <columns>
        <tableColumn name="col1" dataField="firstName"
                     headerText="First Name"
                     enterable="true" editable="true"
                     resizable="false" selectable="true" sortable="true" />
        <tableColumn name="col2" dataField="lastName"
                     headerText="Last Name"
                     enterable="true" editable="false"
                     resizable="true" selectable="true" sortable="true" />
        <tableColumn name="col3" dataField="tel" headerText="Phone"
                     enterable="true" editable="true"
                     resizable="true" selectable="false" sortable="false" />
        <tableColumn name="col4" dataField="email" headerText="email"
                     enterable="false" editable="true" width="150"
```

```

resizable="true" selectable="true" sortable="false" />
</columns>
</table>
</components>
</screen>

```

Listing 2, an example screen with a table control.

Figure 2 shows the screen that results from the table screen defined in Listing 2. The data that populates the rows of this table is automatically fetched from the contacts table by the AppProjector™ at run time. It is also cached on the client to permit faster scrolling of data when there are too many rows to display on the screen at once.

	First Name	Last Name	Phone	email
1	John	Doerr	650 233-3353	jdoerr@kpcb.com
2	Gus	Tai	650 854-9500	gus@trinityventures.com
3	Michael	Samols	650 696.1475	samols@amicusweb.com
4	Jim	Marshall	650 854-7399	jim@selbyventures.com

Fig. 2. Screen resulting from Listing 2.

### 3. Define Control Blocks:

Control blocks are named and consist of a set of named "parameters". These parameters are typically defined to hold temporary information relevant to the display state of the application. For example, a control block may contain a Boolean parameter that represents the "editable" state of a text field on the display. In addition, a control block may also hold temporary session data such as the name and role of the current user. Control block parameters are referred to by the combination of their unique control block name, and parameter name.

There is usually one control block defined per screen, and additional non-screen control blocks defined to hold other information. Control block parameters are defined by specifying their name, data type, validator, and default value. Control block parameter validators are the same as data buffer validators in that they are used to coerce and validate the value to be put in the parameter, and can be used as triggers to alter the state of the application based on runtime conditions.

### 4. Define Commands:

Commands are defined by writing Java classes that implement the *ApplicationCommand* interface. They are named, have an enabled state, and have an "execute()" method that

performs the action of the command. Commands do the work behind the application. They may be used to validate users, populate or delete items from data buffers, or to interface to databases or other applications. There are some built-in commands such as those used to set the current screen and to advance to the next or previous rows in a table data buffer.

### **Deployment:**

After the application developer has finished developing the Java classes that implement the *AppProjectorApplication*, those classes are deployed as a component library in a standard J2EE web application server. The AppProjector™ server is a Java Servlet that manages all applications. It is configured to locate the *AppProjectorApplication* to run based on the application name specified as a parameter to the client applet, and to start the application once the client has started. Screens and appearances are then read from the application and are distributed to the clients at run time.

A single Web Activator™ server can manage many different users and applications, so long as the server knows where to find the requested *AppProjectorApplication*. When a new client connects to the Web Activator™, it is initialized with a new copy of the application, and a new session is started. This session stores the runtime state of the application for one user for the duration of the session. The server also manages the lifecycle of each session and discards the session when the client exits.

### **Integration:**

The Web Activator™ is integrated with an existing server-side application by representing that application in terms of the *AppProjectorApplication* interface, and defining screens in XML. The *AppProjectorApplication* interface can be easily adapted to existing enterprise Java web applications that use the J2EE framework.

### **Conclusion:**

The Asperon AppProjector™ is a powerful new application engine that can enable application developers to quickly create and deploy enterprise class based business applications that run over the web without having to worry about client-side compatibility. The resulting applications have unprecedented interactivity, and can run in most browsers (version 4.0 and above) without a plug-in. The AppProjector™ completely handles all aspects of the user interface and manages all user session information. It is based on the widely adopted standards of Java and XML, and it can be used in any J2EE compatible web application server.

## **About Asperon Corporation:**

Asperon Corporation is a provider of applications infrastructure software for developing and deploying highly interactive web based business applications. Our proprietary user interface and business object technologies provide critical infrastructure for high-end business software that is hosted for use over the web. It is ideal for software intended to be distributed using the application service provider (ASP) model. Our AppProjector™ client is being used as a mission-critical component of applications at several Fortune 500 companies, providing unprecedented interactive speed and significantly lower total cost of ownership.

Asperon was founded by John T. Kucera, Ph.D. (MIT) – formerly a director of technology development in the CRM division of Oracle, and Solon F. Blundell, Ph.D. an MIT graduate, former professor of mathematics at Boston University, and former developer of analytical, predictive software at Cambridge Systematics. Dr. Kucera had earlier been instrumental in converting Oracle's ERP applications to run over the internet, and has over eight years experience in database and user interface development.

## **More Information:**

For more information please contact:

***Asperon Corporation***  
***535 Everett Ave., Suite 306***  
***Palo Alto, CA 94301***  
***650-321-8700***  
***[sales@asperon.com](mailto:sales@asperon.com)***  
***<http://www.asperon.com>***

1. Asperon and AppProjector are trademarks of Asperon Corporation, Palo Alto, CA.