MVC Overview

Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

1 INTRODUCTION 3

1.1 Logical architecture 4
1.2 Physical architecture 4
1.3 Schema-based development 4
1.4 Web page architecture 5
1.5 MVC Architecture Layers 6

2 VISUALISATION 7

2.1 Development steps 7
2.2 FormMaker 8
2.2.1 Application Map - Pages, Actions and Handlers 8
2.2.2 Page Information 8
2.2.3 Action Information 10
2.2.4 Handler Information 10
2.2.5 Section Information 11
2.2.6 Field Information 12
2.2.7 Group Information 14
2.2.8 Repeat Information 15
2.2.9 Data Binding Information 16

3 ORCHESTRATION 17

3.1 Information Flow 17
3.1.1 Flow Objects 18
3.1.2 Flow Messages 19
3.1.3 Mapping File 21
3.1.4 Message Wrapper 21
3.2 XDE 22
3.2.1 Projects 22
3.2.2 Patterns 23
3.2.3 Nodes 24
3.2.4 Rules 26
3.2.5 Asset Pools 28
3.2.6 XGen 29
3.3 Tracking information flow 30

4 INTEGRATION 31

4.1 Integration Adaptors 31

5 NEXT STEPS 32

Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

1 Introduction

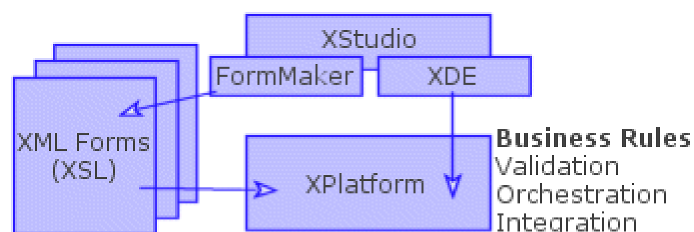
Hyfinity's MVC is a RAD environment for the design and deployment of enterprise-strength dynamic web applications. Based on a native service-oriented platform, MVC provides all the productivity benefits of web application development and leverages a powerful web services platform for page orchestration and integration. MVC can be used to automate most of the time consuming elements associated with web application development and integration. Some of the key features of MVC include:

- Web Based Graphical IDEs
- Schema Based Development
- Auto Gen Validation Logic
- Automatic Data Binding
- Automatic Pre and Post-Population
- Dynamic, Content Based Page Delivery
- Full Web Service interoperability
- Service-Oriented Integration

This is an overview document, which will introduce the necessary concepts to enable development of MVC applications. You should consult the FormMaker and XDE User Guides, together with the rest of the XStudio and XPlatform documentation for full details. The complete MVC documentation set is available at:

http://www.hyfinity.net

MVC uses XStudio for developing XML applications and XPlatform for hosting the resulting applications. MVC is primarily concerned with the development of interactive self-service applications and employs a tool within XStudio known as FormMaker for building web pages.



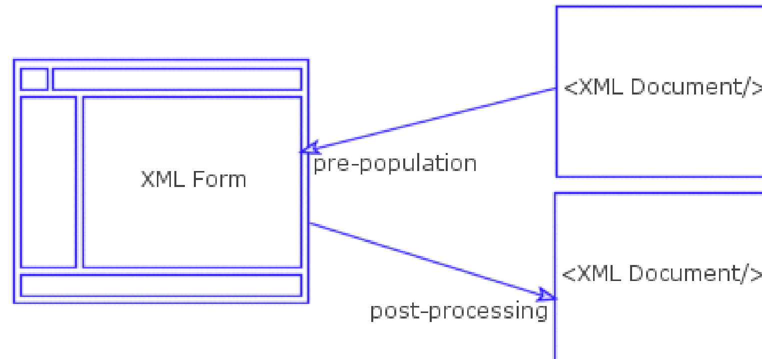
An additional MVC component is present within XPlatform known as SXForms. This is a plug-in for delivery and post-processing of XML forms from a Browser. Additional information is provided in the following sections.

You can use this overview document in conjunction with the tutorials. Once you are comfortable with various concepts you can start the tutorials and refer back to this document as required.

Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

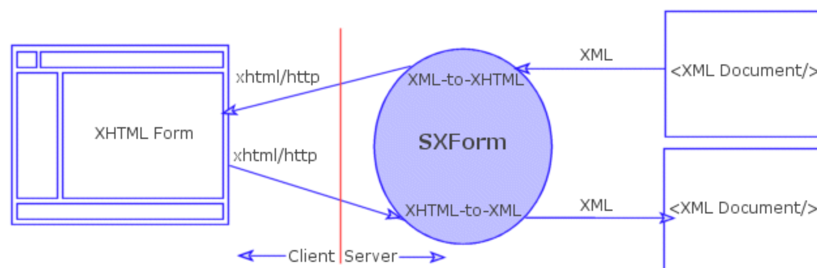
1.1 Logical architecture

Logically, MVC applications can be viewed as having the capability of editing XML documents in-place. This provides the ability to render an XML document as a form on the browser. The captured information is then validated and mapped back to an XML document.



1.2 Physical architecture

Physically, browsers do not support XML forms in a uniform way. XHTML is still the primary language for browsers. MVC applications don't require any downloads on the browser, instead, a series of innovative steps are used to provide logical XML-based development to enable the in-place XML editing paradigm. To achieve this, the XML information is transformed to XHTML on the server using XSL. The information that is captured on the browser is posted and transformed from XHTML name-value pairs to XML on the server-side. This provides the ability to logically view the XML forms as editing an XML document in-place.



1.3 Schema-based development

Each form in an MVC application is derived from an XML-schema where possible. Unlike most technologies, MVC uses an information modeling approach to ensure better-engineering and cohesion between different elements of a distributed application. Rather than providing a form painter and starting with fields on a canvas, followed by defining their attributes, MVC prefers information to be defined, independent of the eventual display format of such information.

For example, if a person's surname, forename and age fields require capture then this information can to be defined as an XML schema. The amount of detail in the schema will dictate the level of automation that will be achieved in the resulting application. For example, if all three fields are defined as alphanumeric then the information will simply be captured and submitted by default unless specific validation rules are explicitly specified. However, if the age

Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

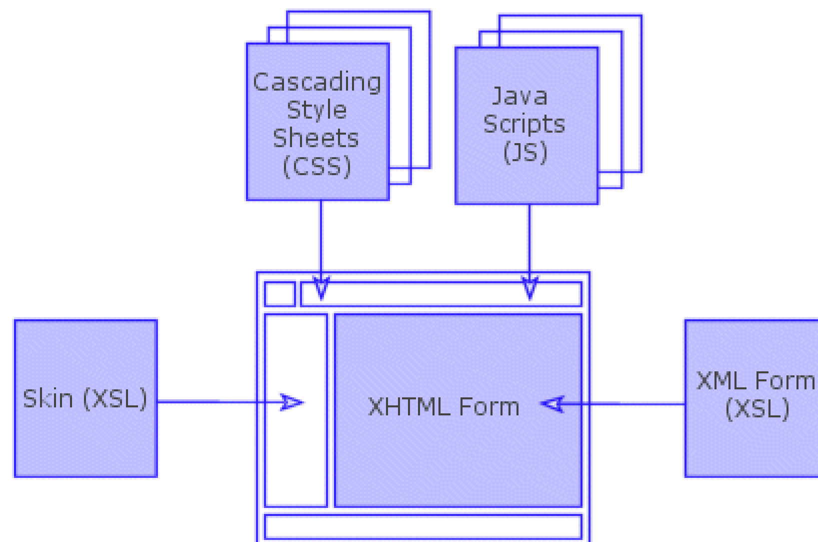
field is defined as integer then various validation steps will be performed automatically. Further, if the age is defined as being between 18 and 60 for example, then scripting will be automatically generated and activated to ensure proper validation.

Schema-driven development requires a different approach to traditional 'form-painting' tools, providing many advantages.

- Well engineered applications, with good cohesion throughout the document flow route.
- Reuse of existing schemas.
- Once defined, schemas can be reused across multiple forms, providing massive reusability.
- Ensures web applications validation and data structures are consistent with web services they may utilise.
- Schemas can be used for purposes other than defining XML forms. For example, messaging.
- Provides a display-independent representation of form information.

1.4 Web page architecture

As well as the XML forms, each MVC application uses additional separation of key web application components to enable better engineering and easier future maintenance. Some components apply application-wide with page-specific overrides. All MVC applications require the following:



- Common Cascading Style Sheet - This is used to provide a generic application-wide look-and-feel to the application and provides separation of content and presentation.
- A common application Skin - This is in the form of an XSL and provides a template for the overall application. The skin will provide placeholders for menus, logos, headers, footers, etc. The actual content will override this template as necessary for each page, but the parts that are not overridden will remain the same.

Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

The CSS and Skin require definition only once and provide an overall look-and-feel and layout for the web application. The main purpose of MVC however is to enable the creation of XML forms. This is achieved by using a component of MVC known as FormMaker. FormMaker is part of XStudio and provides a web-based IDE for creating XML forms. FormMaker can be used to produce the following elements of a web application:

- XML forms encapsulated in XSL. These XSLs are transformed to XHTML during execution to enable wide browser coverage without downloads.
- Page-specific CSSs to override or complement the application-wide CSS if necessary.
- Javascripts. A common set of javascript files are present for the application, which contain generic scripting for client-side validation. A large part of this scripting is generic and activated based on the schema information that is present for each form. Additional server-side logic can be included to compliment the client-side scripting.
- XML Data Mapping information. Similar to the scripting, this information is distributed between the client and the server and enables the data to be mapped bi-directionally between the browser and the underlying XML documents. Once defined, this element is automated and very transparent to the designer.

1.5 MVC architecture layers

MVC applications consist of three key stages:

1. Visualisation – This stage enables the design of the web screens, including data-binding and client-side validation. The main development tool used during this stage is FormMaker. This stage produces XSLs that will be served by the View layer of the MVC pattern.
2. Orchestration – This stage enables the physical deployment of the web screens and enables orchestration between screens, using a web server. Most of the orchestration functionality is automatically generated during stage 1, but additional tailoring and application deployment is performed during this stage using the eXtensible Design Environment (XDE). This stage involves layers View and Controller of the MVC pattern.
3. Integration – This stage provides the Model layer of the MVC architecture and, together with the Controller layer, provides integration capabilities. This can be designed using the XEngine operations such as the Service operator (see later) to interoperate with remote web services or generate proxy services from WSDL files.

These three key stages, Visualisation, Orchestration and Integration are discussed later in the document.

Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

2 Visualisation

Visualisation is the first stage of development and primarily concerned with the creation of the web pages. In later sections the orchestration of these pages will be discussed in more detail, followed by the Integration stage.

MVC enables complete web pages to be formed during the design stage of any project. Due to the XML foundations the pages can be extended very easily in an incremental and iterative manner.

It is very easy to design web pages that deliver variable content based on the XML information flows. This is very powerful for requirements such as various reporting screens.

2.1 Development steps

Traditionally, dynamic web applications need to compose XHTML for the browser and decompose the information posted from the browser in server-side logic. MVC uses innovative XML techniques to eliminate the bulk of the effort required to pre-populate and validate information to and from the browser. MVC uses XPath to enable information to be mapped between the XML form and the underlying XML document from which the data is derived. This is a two-way mapping that provides pre-population of information to the browser and the subsequent capture, after the information has been posted from the browser. This enables the automation of a large part of the server-side scripting role, typically associated with web application development. The key steps for developing MVC applications include the following:

- Design screen navigation model for complex applications.
- Design or reuse a CSS and application Skin.
- For each page, define or reuse a schema as a starting point, if necessary, to accelerate development.
- The generic application or specific page(s) can be generated at this stage for initial view. Subsequent steps enable individual fields or groups of fields to be selected, styled, positioned, validated and mapped.
- Indicate fields that will be part of each form. Include additional fields here for buttons, etc., which may not be part of schemas.
- 'Fine-tune' display and validation for each field if required.
- Bind fields to XML elements in both directions using XPath.
- Design and adapt server-side nodes for page delivery and data-binding. All the above steps can be achieved using FormMaker. This final step requires configuration of the MVC server architecture and is covered in more detail in later sections on Orchestration and Integration.

The diagrams in the following pages lead through the main MVC screens used to build Web pages.

Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

2.2 FormMaker

FormMaker is the main user interface for creating web pages. It handles all visual aspects, including page navigation, form structures, field details and data mappings, including page handlers and actions.

2.2.1 Application Map - Pages, Actions and Handlers

The Application Map screen can use the palette on the top left to create a navigational model of the overall application. The View layer of web applications primarily consists of pages that can generate a set of actions, which are handled by server-side handlers. The action links typically signify posting of web forms containing information.

2.2.2 Page Information

The first screen is the Application Map screen used to create the overall design of related sets of pages. This screen is also used to setup the overall theme of the forms application, including any schema standards that may be applicable.

The top right hand pane is context-sensitive, depending on whether a page, action link or handler is selected. Each action link and handler created will automatically generate server-side functionality that can be configured and deployed during the Orchestration stage discussed later. These handlers represent nodes in the Controller layer of the MVC pattern. This visualisation stage will continue with the creation and tailoring or web pages, including the selection of actions and data-binding.

When you select a page the right hand panel will display all page-specific information. All other areas of FormMaker deal with page information and you should consult the FormMaker User Guide for additional detail.

Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

2.2.3 Action Information

When you select a link between a page and a handler it will be highlighted. The right hand side panel will change context to display action information.

Action Details

From Page: contacts
To Handler: manage_contact
Action Name: view_contact

Hide Advanced Options

Select Page Painter: Page_Painter

Create New Painter

2.2.4 Handler Information

When you select a handler the right hand side panel will change context to display handler information. In this section you can define controllers and also select any pre-defined services that act as proxies to remote services. Please see later section on Integration for more information on generating services from WSDL files.

Handler Details

Create New Controller

Select Existing Controller

manage_contact-Controller

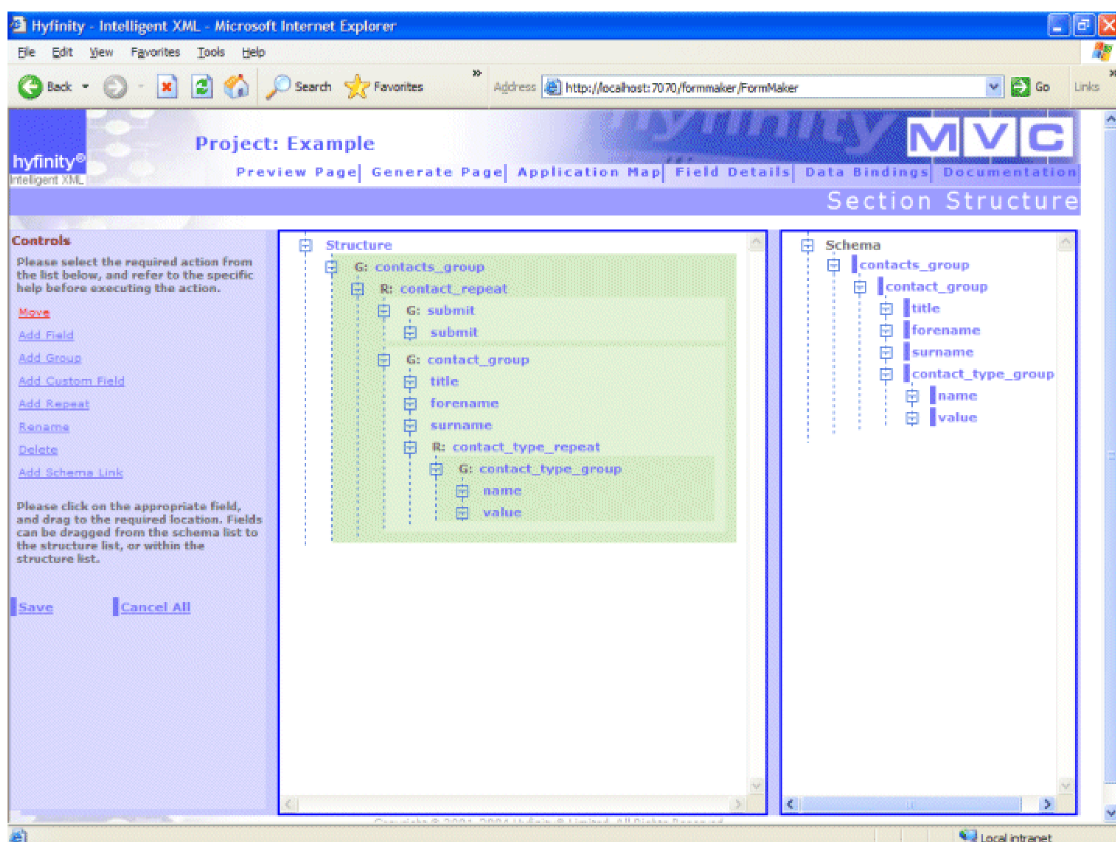
Please select a node.
manage_contact
manage_contact-Controller
No nodes available

Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

2.2.5 Section Information

For each page that is defined in the Application Map screen, a Section Structure screen is used to define groups of information containing individual fields. The information groups can contain nested repeating groups of information as appropriate. If schemas are used here then the information in the schemas can be dragged-and-dropped on to the form.

Link Information - Additional consideration should be given here for the action links defined in the previous step. XML information schemas are unlikely to define actions and some schemas will not have been designed for use within web pages. The Section Structure screen enables the definition of additional fields, which can be used to represent action buttons. The example below shows a submit group containing a submit field.

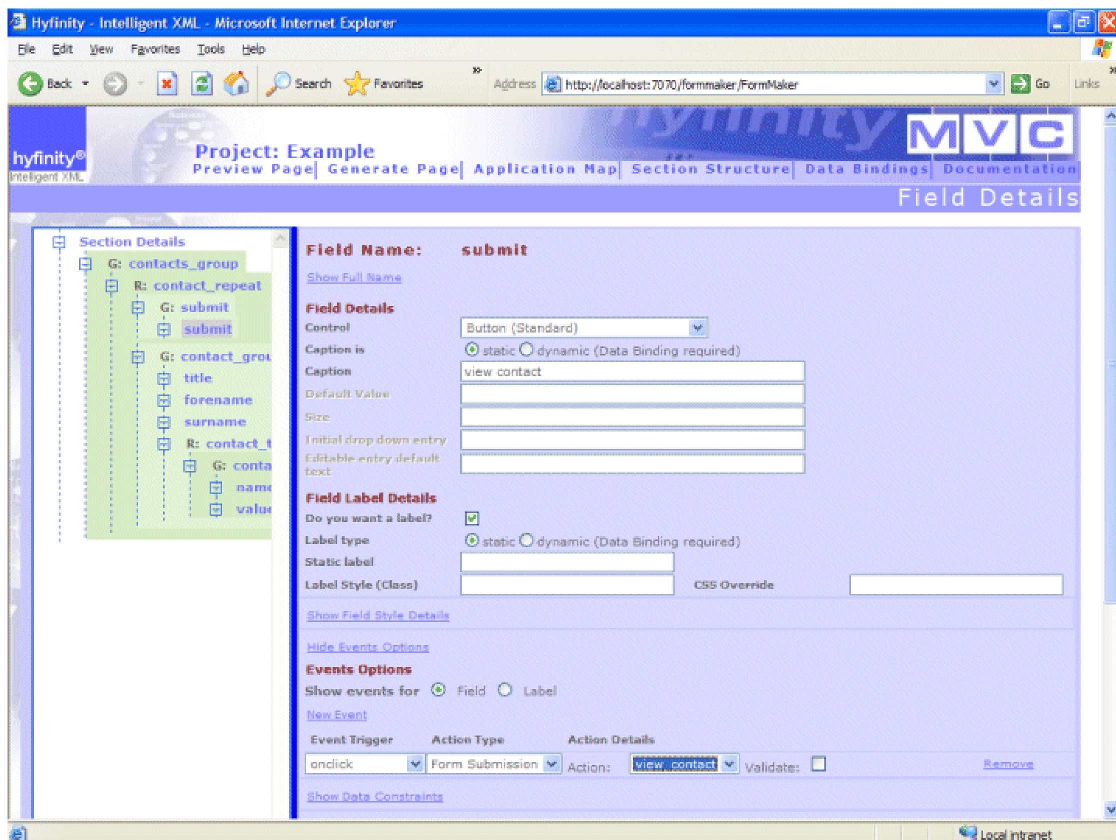


Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

2.2.6 Field Information

Once a section structure has been defined the page can be generated and previewed. Following this preview, each repeat, group or individual field can be selected and defined in more detail. For example, the type of control used to represent the field on the form, field labels, styling, etc. All this information will be initially defaulted using a combination of the CSS, application skin and the schema information if defined. This information can be changed or supplemented as required.

Link Information - Action links defined in the previous step on the Application Map screen can be defined as actions triggered by browser events, e.g. clicking a button. For these browser events, it should be possible to select actions defined in the Application Map screen. These actions should be available under the Events Options section of the screen.



Each field can have data constraints. If a schema was used then these constraints will be automatically generated and can be appended to or removed from manually if desired.

Hide Data Constraints

Data Constraints

Data Type	string	Mandatory	☑
Enumerations	Data Value: Mr, Mrs, Ms, Dr, Sir	Display Text: Mr, Mrs, Ms, Dr, Sir	Up / Down / Save / Remove
	Use these enumeration values for output display? ☐		
Min Inclusive		Max Inclusive	
Min Exclusive		Max Exclusive	
Min Length		Max Length	
Length			
Pattern			

For certain field types there are sophisticated value formatting and conversion routines available. For example, dates can have different display and storage formats and these transformations can be automatically handled.

Hide Value Conversions

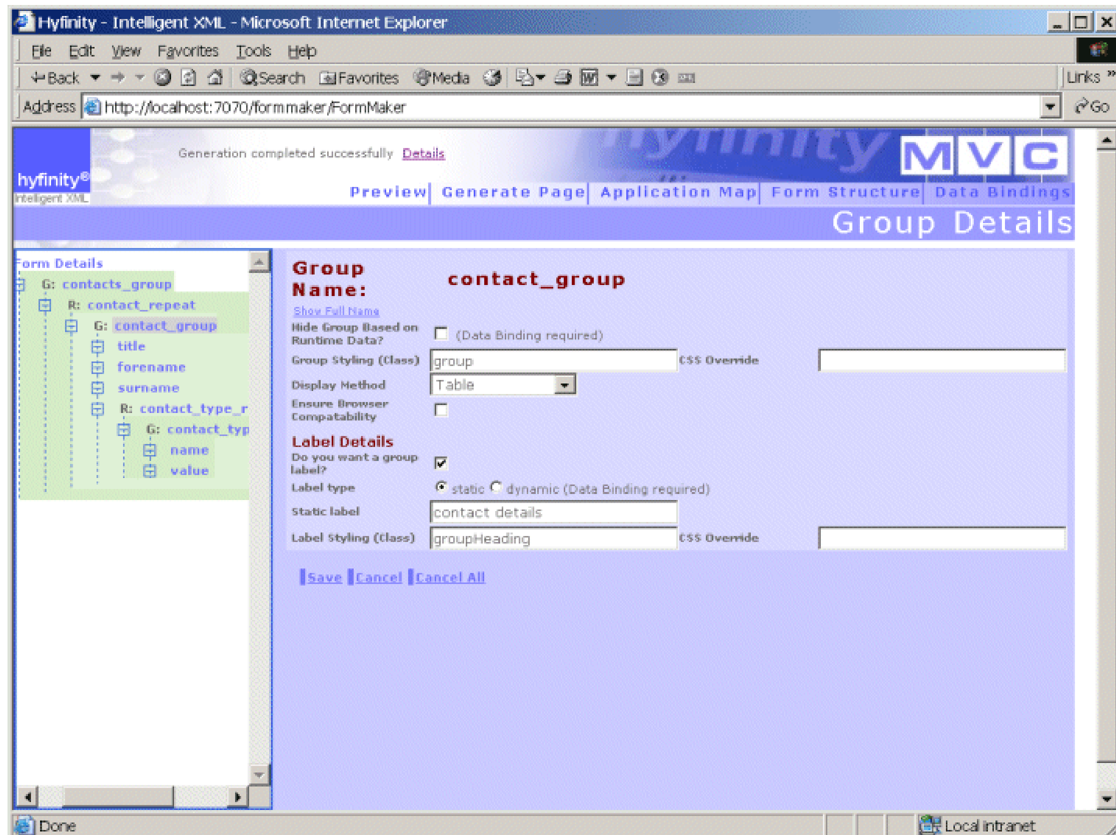
Value Conversions

	Data	Display
Date Formats	yyyy-MM-dd	dd/MM/yyyy
Case	Preserve	Preserve
Whitespace	Preserve	

Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

2.2.7 Group Information

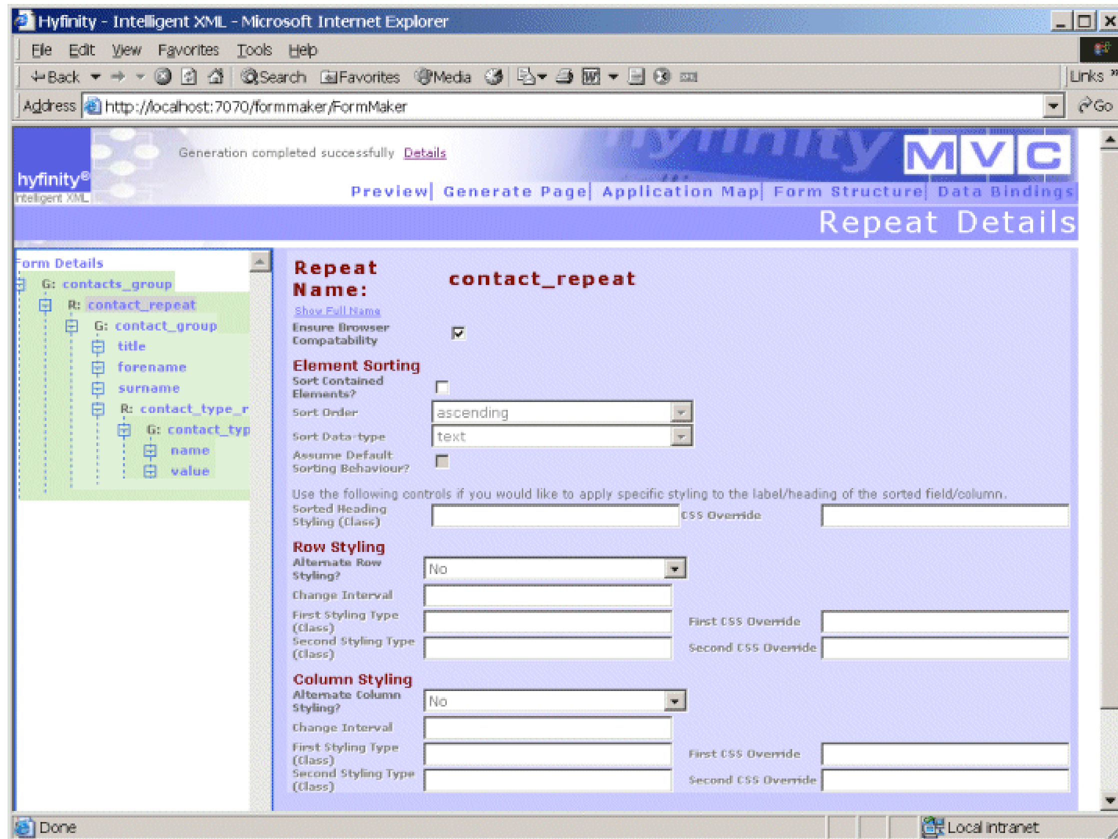
Information on sections can be grouped for easier styling and manipulation. For example, complete groups of information may be hidden, highlighted, etc., based on other values on the form. There are also sophisticated features available for different display formats for information groups, such as horizontal, vertical, tabular, etc.



Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

2.2.8 Repeat Information

Information groups can be repeated. There are various options available for the display and styling of such repeating groups of information, including sorting.



Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

2.2.9 Data Binding Information

The pages can be repeatedly previewed until the desired effect has been achieved. Once the individual repeats, groups and fields have been fine-tuned then each field can be mapped from a data source and back to a data target after completion. If XML instance documents were specified in the Application Map, then these can be used here to drag-and-drop fields onto the binding links. The binding information is automatically completed wherever possible.

Link Information – Each section will have an Instance Location XPath. This enables the isolation of data that needs to be captured and automatically pre and post-populated. This will be discussed in more detail in the Orchestration section and enables the isolation of the information to be captured from a larger document.

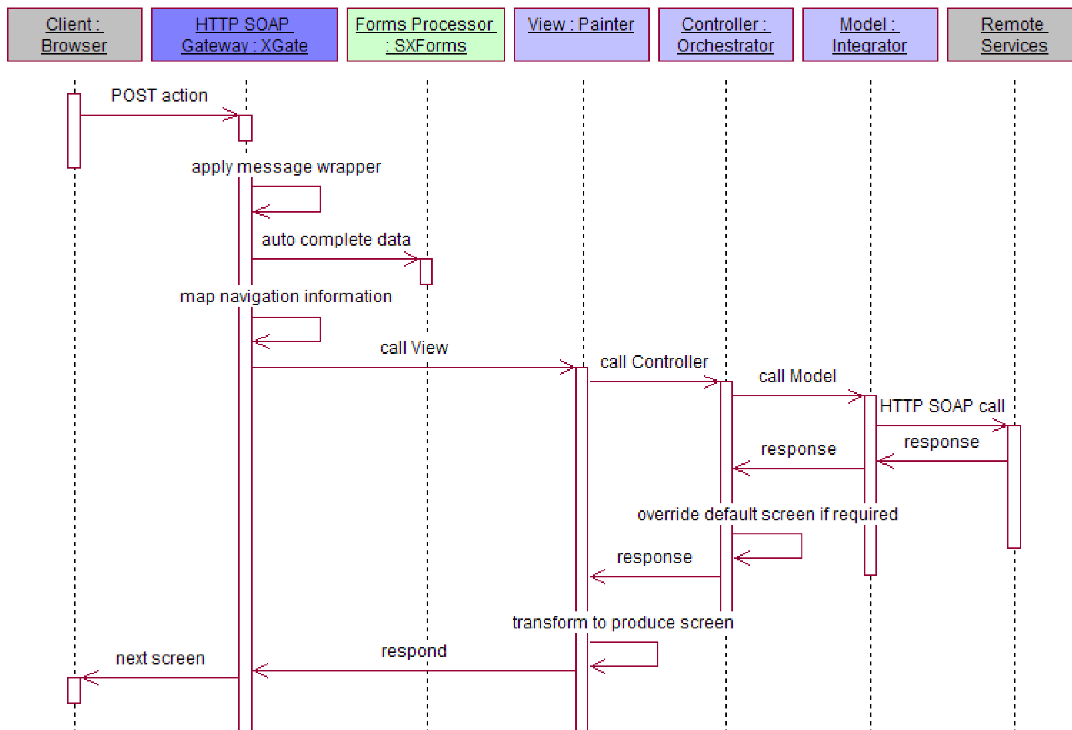
Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

3 Orchestration

The visualisation stage enables the creation of web pages. By default these pages are generated as XSLs. The orchestration stage enables the hosting and orchestration of these pages and integration to services providing business data. This is achieved using the XML rules engine (XEngine) to manipulate and orchestrate XML documents. This section will provide a description of the complete information flow model from the browser, through the middle-tier and integration to remote services.

3.1 Information Flow

When MVC applications are deployed they undergo a series of steps to enable full orchestration. These are typical of most well-engineered systems. The diagram below shows the key information flows and each object and message is discussed in more detail in tables later in this section.

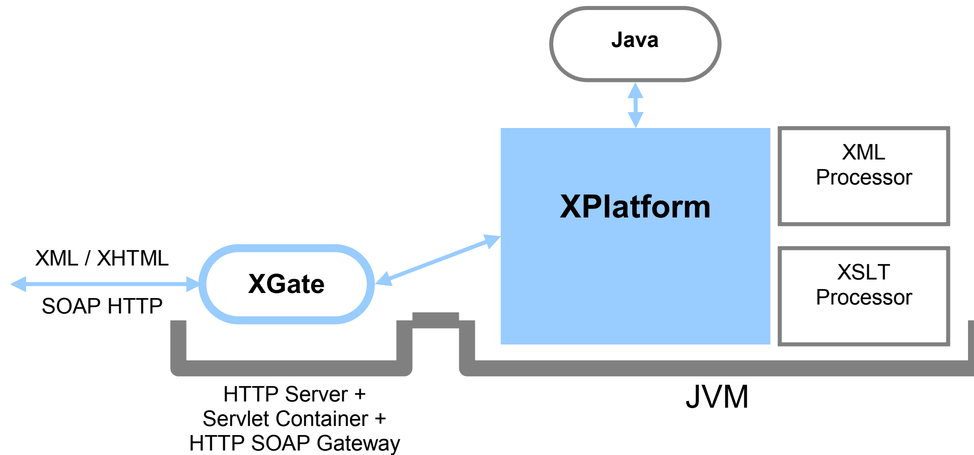


Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

3.1.1 Flow Objects

Key system objects	
XGate	In native format XPlatform can host nodes that exchange XML messages using in-memory Java objects. Often there is requirement to exchange messages over HTTP SOAP for web services orchestration. XGate is the generic HTTP SOAP Gateway used by XPlatform to achieve bi-directional HTTP SOAP communications. XGate can receive information in XML and XHTML formats and use plug-ins and XSL filters to post process this information before submission to XPlatform nodes. An example plug-in is SXForms for forms processing in MVC. For MVC, the default access method is HTTP. XGate is an integral component of XPlatform, but it’s loosely-coupled to enable deployment flexibility. This relationship is illustrated in the diagram following this table. XGate also provides facilities for outgoing calls to remote services using HTTP SOAP. The outgoing calls are achieved using the Service operation mentioned later.
SXForms	SXForms is an XGate plug-in that enables the bridging between XML and XHTML. During the incoming message flow from the browser, SXForms will auto-populate the submitted information into XML format for onward processing. During the outgoing message flow, SXForms identifies the form instance information that requires capturing.
View	The View layer shown is a native XPlatform layer and may contain any number of patterns and nodes. For MVC applications, a single View pattern is created by default with a single node called Page_Painter. During the incoming message flow, The Page_Painter node routes the message according to the specified controller in the message header. During the outgoing message flow, Page_Painter performs an XSL transform to produce the next web page.
Controller	The controller acts as the orchestration layer and can have any number of patterns and nodes. By default a single controller is created for each page handler that is created in FormMaker. Controllers, typically perform the role of splitting and aggregating messages using the Model layer and other controllers. For example, a web page may require various services to be used to dynamically pre-populate drop-down lists. These lists can be cached for an individual user or all users if required.
Model	This layer represents the information that is required by the application. This information may be retrieved from a database, the filesystem or, more commonly in composite applications, from remote services. Nodes in the Model layer can be created to act as proxies to remote services or created using the WSDL import plug-in within XDE. Once created, these nodes appear as services within FormMaker that can be used by controllers. More information on this layer is provided in the Integration section later.

Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential



3.1.2 Flow Messages

Key system message flows	
Incoming	
POST action	Browser requests a Page based on a browser triggered event, e.g. a button or a URL request. If this is the start page of the application then this fact can be indicated under Advanced Options on the top right hand pane of the Application Map screen. The request triggers an action via the HTTP Servlet Container.
Apply message wrapper	XGate uses an optional message wrapper to structure the incoming message into structured XML. The wrapped message will include additional information such as the action and controller that will handle this request and the default transformation to use to display the next page. This information is obtained from the mapping file (see later).
Auto-complete data	XGate then invokes any plug-ins to post process the received information. In case of MVC, XGate uses the SXForms plug-in to automatically populate the captured information into the desired locations within the XForms Instance.
Map navigation information	The incoming event triggered action is mapped to a controller node via the XGate mapping.xml file. Please note that this information is defined by the graphical Application Map editor when Links are added. This information is inserted in the header of the wrapped message. Later sections provide examples of the message wrapper and mapping files. An additional element is also inserted to indicate the default next screen to display if everything is successful.
Call View	XGate then forwards the XML message to the Page_Painter node in the View layer. During the incoming flow the Page_Painter node will forward the message to the target Controller in the next layer as specified in the header of the wrapped message.
Call Controller	This is a Service operation from the Page_Painter to a Controller that will handle this particular request.
Call Model	This is a Service operation from the Controller to particular Models to obtain or store application information. Many of the model

Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

Key system message flows	
	nodes will be proxies to remote web services.
HTTP SOAP call	This is a Service call to remote web services using HTTP SOAP or HTTP XML as appropriate. This enables the support of SOAP or REST based architectures.
Outgoing	
Response from remote service	The remote service returns control back to the Model layer.
Response from Model	The model returns control back to the controller after processing
Override default screen if required	The default presentation XSL will have been populated by XGate during the incoming message flow. At this current stage it is possible to override this default page setting to indicate a different page. For example, the display of an error page if an error occurred, or to re-direct based on the XML content required. This is very useful where search results from a service may mean you wish to return to different pages based on zero, one or many entries returned.
Response from Controller	The controller returns control to the Page_Painter after processing.
Transform to produce screen	The view then applies a XSL transform using the pages developed during the Visualisation stage using FormMaker. The XSL to use can be set within the Controller, but the default transform will be applied if not overridden. This default is stated during construction of the navigation model. For example, if an error occurred during a remote call then a fault screen may need to be rendered instead of the default screen.
Response from View	The transformed information is returned to XGate.
Next screen	XGate returns the page back to the browser.

Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

3.1.3 Mapping File

An example mapping file is shown below. This file is automatically generated and maintained by FormMaker and XDE and it should not be necessary for any manual amendments in most application scenarios.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <mapping xmlns:fo="http://www.w3.org/1999/XSL/Format" xmlns:xf="http://www.hyfinity.com/xfactory" xmlns:xg="http://www.hyfinity.com/xgate" xmlns="">
3    <action desc="" name="process">
4      <destination action="requestInboundService" desc="Generated by XGen" name="mvc-Example-view-sxforms"
5                   runtime_instance="com.hyfinity.xagent.XAgent" type="xagent" wrapper="to_header_wrapper.xsl">
6        <param desc="" name="" type=""></param>
7      </destination>
8    </action>
9    <action desc="Please Enter a Description Here!" name="submitContact">
10     <destination action="requestInboundService" desc="Generated by XGen" name="mvc-Example-view-sxforms"
11                  runtime_instance="com.hyfinity.xagent.XAgent" type="xagent" wrapper="to_header_wrapper.xsl">
12       <param desc="" name="" type=""></param>
13     </destination>
14   </action>
15 </mapping>
```

3.1.4 Message Wrapper

Below is an example wrapper document used by XGate to wrap the incoming message. The incoming information is contained in the Data element. The Control element contains the action and Controller as defined in FormMaker to indicate the action to invoke on the stated Controller. The Page_Painter uses this information to make the Service call to the appropriate Controller. During the outgoing information flow, the Page_Painter uses the Page element set by the returning Controller to perform the transform to render the next page.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <eForm xmlns="http://www.hyfinity.com/mvc" xmlns:fo="http://www.w3.org/1999/XSL/Format" xmlns:xg="http://www.hyfinity.com/xgate">
3    <Control>
4      <action>view_contact</action>
5      <Page>contacts.xsl</Page>
6      <Controller>mvc-Example-manage_contacts-Controller</Controller>
7    </Control>
8    <Data>
9      <contact action="Get" id="3" successful="false" xmlns="http://www.hyfinity.com/schemas"/>
10   </Data>
11 </eForm>
12
```

In FormMaker there is an *XPath to Instance Location* available for each form. This XPath defines the data portion of the message on the server and enables SXForms to auto-populate the information returned by the browser. In this wrapper example the XPath to Instance Location would be /mvc:eForm/mvc:Data/*, which essentially indicates the data portion of the larger XML document. By default this will not require population unless the location and structure of the instance request and response documents need to be different to the default position within the wrapper.

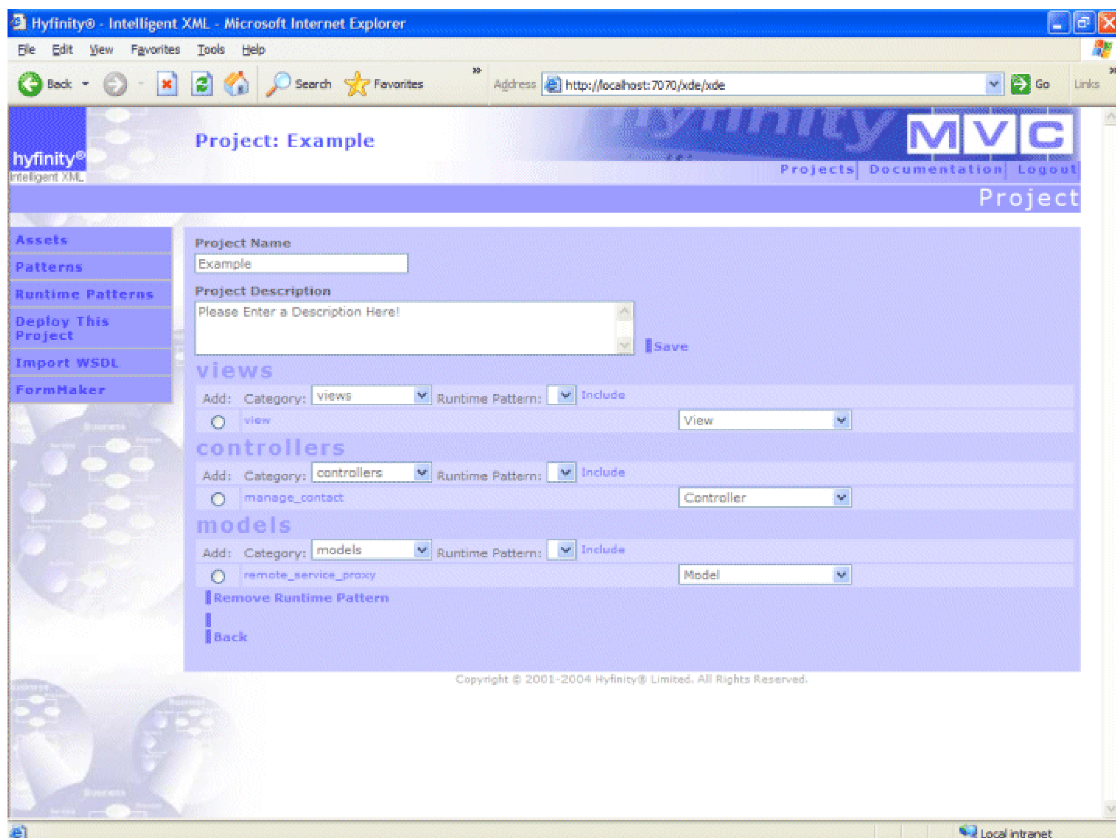
Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

3.2 XDE

Most of the steps highlighted above are automated during the Visualisation stage as new pages, actions and handlers are created. This information can be viewed on the server-side using XDE. This section provides necessary information to enable a high-level understanding of the orchestration steps. Application designers and developers should consult detailed XDE and RuleMaker documentation.

3.2.1 Projects

After logging onto XDE you should be presented with the Project screen. From here you can start FormMaker. You should also see the MVC layers and the patterns available in each layer. The View will have a single pattern, which should not require any modification. The Controller layer will expand as more handlers are created in FormMaker. The Model layer will also expand as more remote service proxies are defined.

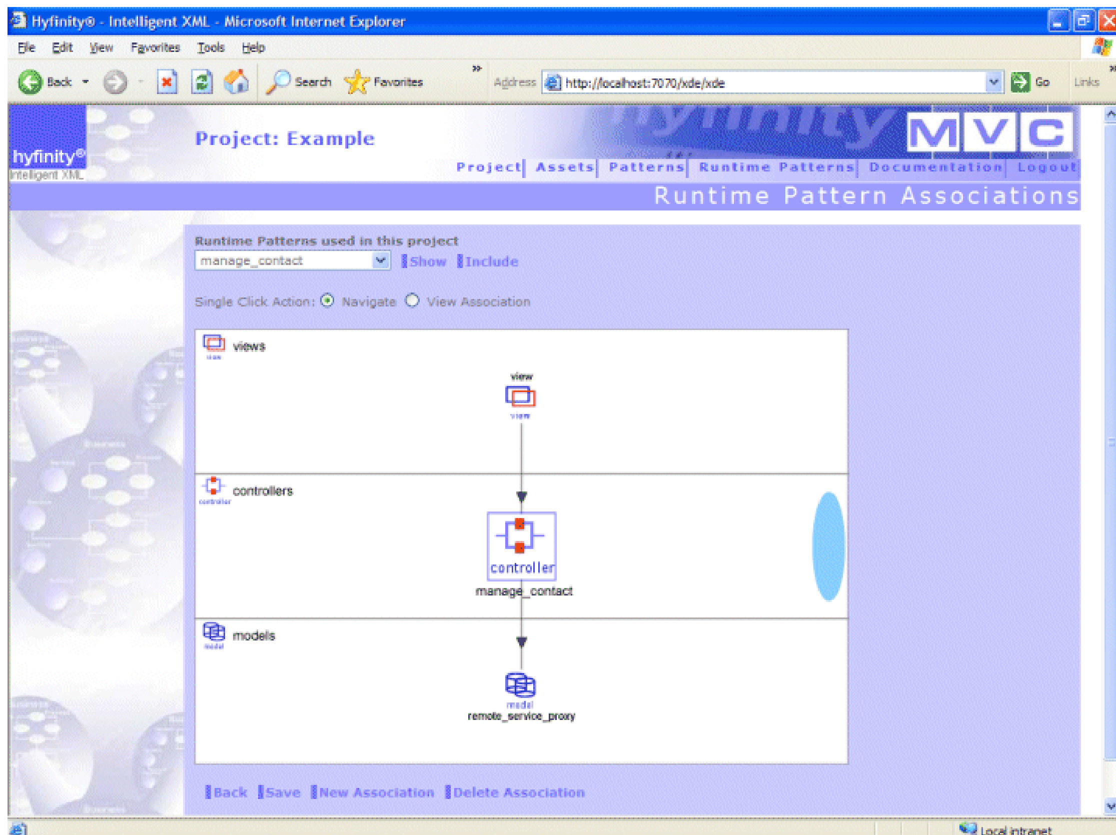


Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

3.2.2 Patterns

If you click through one of the Controller patterns you should be presented with the specific Pattern information. Using XDE it is possible to construct very sophisticated pattern and node interactions, but for MVC, most patterns are simple single node patterns. A node is an autonomous service-oriented agent.

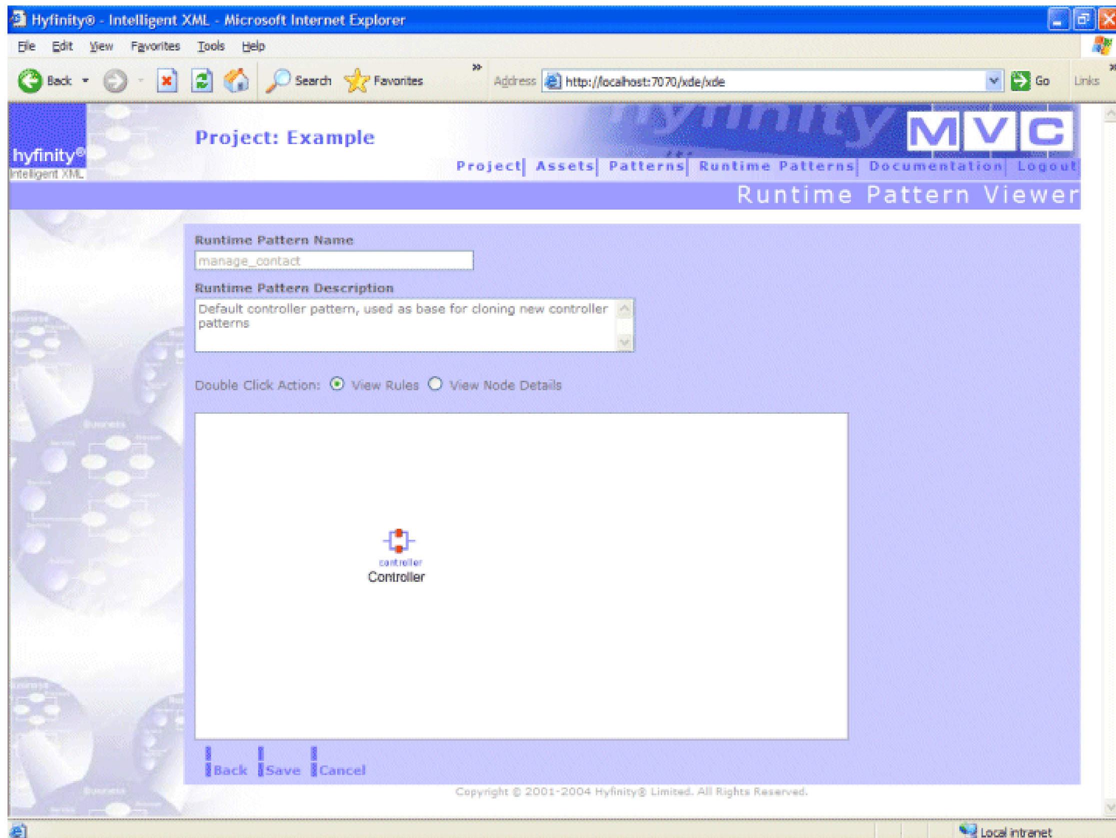
The Controllers typically represent one-to-one mappings between a Page Handler and a Controller node.



Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

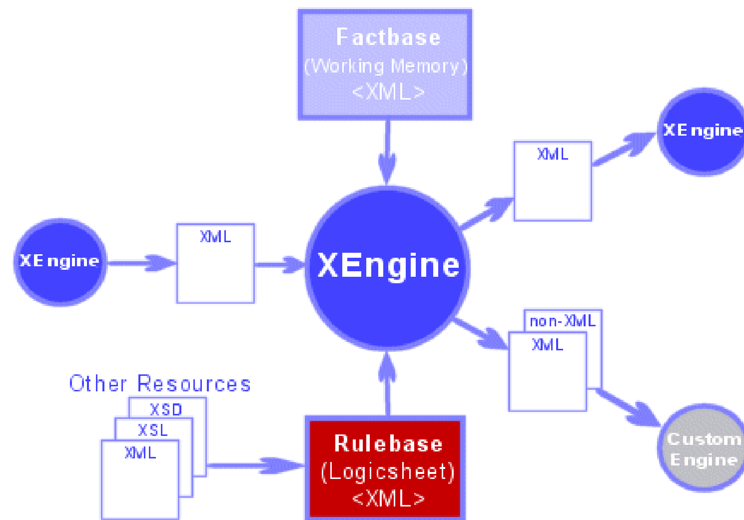
3.2.3 Nodes

If you click through the pattern you should see the Controller node. For detailed information you should consult the XDE user and developer guides.



Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

The following diagram illustrates the rules processing capability available within each node. The key business rules processing component within the MVC architecture is a node, which contains an Engine. When a node is created, an XEngine is assigned by default.



Each node contains a *Logicsheet*, which contains a set of *Rules*. The engine also has a *FactBase*, which is its working memory. The FactBase is an XML document, which is manipulated as the engine executes the rules to derive new information. A FactBase can have sub documents, which are known as *Workspaces*. Workspaces are useful for partitioning information. For example, $Request and $Response may store request and response XML fragments respectively.

Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

3.2.4 Rules

If you click through the node you should see the list of generated rules. You can click through any of the rules and view the details. Each rule will act on the incoming message and fire various actions if necessary. Typically, there will be various calls to nodes in the Model layer to compose information from different remote services. After such invocations, the presentation XSL will be set to enable the Page_Painter to decide which XSL to use for transforming the next page. For more detailed documentation you should consult the RuleMaker user and developer guides.

Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

The following is a list of the standard XEngine operations that are available for XML document manipulation. All operations are applied to the working memory (FactBase). All operations use XPaths to locate XML information within the FactBase.

Operation	Description
AddAttribute	Enables the addition of attributes against elements.
Cache	Caches named XML fragment either in User Session or Application scope.
ClearCache	Clears cached XML information.
Copy	Copies XML fragments from one location to another.
Delete	Deletes an XML fragment.
Log	Enables the logging of XML information to the underlying XPlatform logging system.
Parse	Parses an XML document or fragment into the FactBase. E.g. a document template, an XML fragment or content of another XPath into the FactBase.
Retrieve	Retrieves named XML information from the cache.
Return	A 'hard' return during the execution of a Logicsheet. The response is specified by an XPath.
Save	Saves specified FactBase information to a file on the file system.
Service	This is the main orchestration operation. A Service call can be made to any other node within XPlatform. The target nodes may be native XPlatform nodes or proxies to remote web services. The Service operation always assumes it is calling a named native XPlatform node. The physical manifestation of this node can be specified during deployment of the application.
Transform	This is an XSLT operation on specified FactBase content using a specified XSL file.
Validate	This validates specified FactBase information using an XSD file.
Write	This sets the value of an element or attribute within the FactBase.

Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

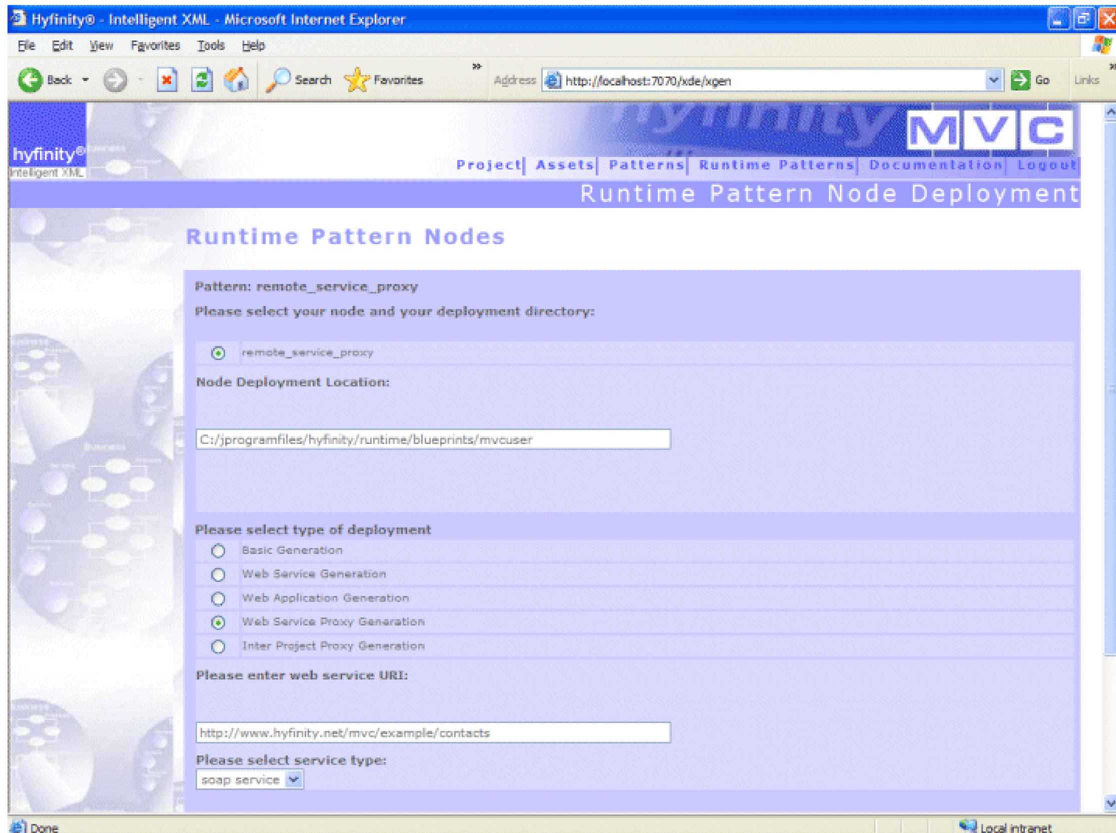
3.2.5 Asset Pools

XDE has the concept of XML Asset pools. All XML assets used within any project are catalogued in these pools. These pools are shared by XDE and FormMaker. The *XSL pool* will contain all the pages created using FormMaker. The XSDs and WSDLs used will be available in the *XSD pool*. Instance documents used for data binding will be available in the *messages pool* or the *other resources pool*.

Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

3.2.6 XGen

As you may be aware, all Morphyc applications such as PxP and MVC adopt a design and deploy concept. Everything discussed so far, including the use of FormMaker and XDE, will result in the production of numerous XML assets, including XSLs, XSDs and logicsheets. These assets need to be deployed on the runtime platform to enable hosting and execution. XGen is the facility that enables deployment and can be accessed from the main XDE Project screen.



By default you can deploy the complete application, which will be hosted on XPlatform. MVC however contains three layers and each layer is slightly different in its communication requirements. You can drill-down and deploy individual patterns and nodes to fine-tune deployment information. The complete application is held in specification form and only physicalised during deployment.

For MVC, we need to deploy the View layer Page_Painter node as a *Web Application* to enable invocation from Browser and other HTTP clients. This deployment mode generates the necessary files, including the XGate mapping file to enable remote invocation of the Painter node via XGate.

The Controller nodes will typically be deployed as native XPlatform nodes and will not require any communication wrappers.

The Model nodes will typically act as remote service proxies to enable interaction with other web services. To enable this we must use the *Web Service Proxy* option. This will require us to state the remote service location and the method of communication. This deployment setting will be automatically configured if the WSDL Import process has been used to generate the model nodes. In this

Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

case, this deployment option should only be used if the service locations defined in the WSDL file need to be overriden. As the deployment is separate from the web application developed, it is possible to develop an application using a local “dummy” service for early testing before linking directly to a published web service. This enables development of the web application, independent of other web services that require development and invocation.

Once the deployment is complete, the application is ready to be invoked from remote clients.

3.3 Tracking information flow

During development, the information flow can be tracked, including all messages, nodes and the rules that are applied to the messages as they flow through XPlatform. This is achieved using the standard logging and tracking system. An illustrative example is shown below.

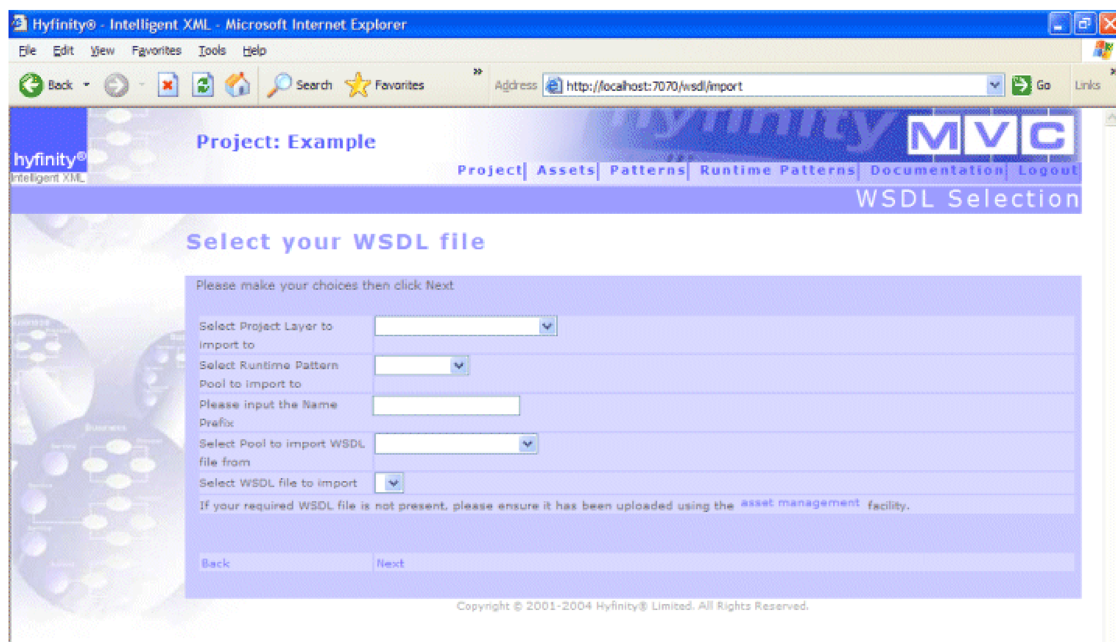
For full details of how to use this facility, please see the Morphyc Installation and Administration Guides.

Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

4 Integration

XPlatform is a native XML and service-oriented platform for processing XML messages. Using XGate, any node can interact with external web services using the Service operation mentioned earlier. Within MVC, the Model layer is used to integrate to remote information. This is typically achieved by invoking remote web services using HTTP SOAP.

These remote invocations can be set up by creating specific Model layer nodes using XDE and then defining orchestration logic in the controllers to call these remote service proxies. If remote services are described using WSDL files then the Model layer can be generated by importing the WSDL files. This option is available from the project screen.



All nodes generated in the Model layer will appear as services in FormMaker that can be used by handlers to obtain remote information.

4.1 Integration Adaptors

All SOAP web services or REST based interfaces can be utilised without the use of integration adaptors.

Additional adaptors are available for integration to non-XML services such as SQL databases, Email Servers and Java objects. You can find more information within the PxP documentation under XPlatform Custom Engines. Software vendors are rapidly ensuring legacy integration technologies, application packages and SQL databases offer web service interfaces, which ensure easy integration.

Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential

5 Next Steps

This overview should have provided the high-level information necessary to understand all MVC concepts. This should enable the construction of applications within the MVC environment. You can access the complete documentation at www.hyfinity.net for more detailed information.

You may now wish to complete the MVC tutorials and refer back to this document as required.

If you require any information or assistance, please contact the Hyfinity team using out contact information available at www.hyfinity.com and www.hyfinity.net

Copyright © 2001-2005 Hyfinity Limited. All rights reserved.
Company Confidential