Automating Rich Internet Application Development for Enterprise Web 2.0 and SOA

Enterprise Web 2.0 >>> FAST

White Paper – November 2006

Abstract

Modern Rich Internet Applications for SOA have to cope with very complex, multi-layered peer-to-peer architectures and ever-increasing technologies, ranging from XHTML, AJAX, Java, XML, HTTP SOAP and all the transformations in-between different layers of the architecture. Traditional development tools and languages were not designed for this type or volume of native XML information, which can lead to applications that are costly to implement, rigid and difficult to maintain. Using a native XML approach and tools that are 'designed-for-purpose', it is possible to accelerate the development of Rich Internet Applications for SOA by at least 60% through automation and reuse of readily available XML assets.

Copyright (c) 2006 Hyfinity Limited. All rights reserved

Introduction

In today's rapidly changing business environment, the ability to develop and change applications to keep pace with agile business models can be a challenge. The pressures on development teams for development and maintenance of applications can be compounded by numerous factors, including lack of resources, cost of development and the heterogeneous nature of enterprise application environments.

With the move towards SOA, more and more enterprise systems are wrapped as services. In real world scenarios, these 'headless' services and the information they provide often need to be interpreted by humans. The need for user interfaces therefore has not diminished, indeed the demands of end users is increasing. More and more users are now demanding desktop-style functionality within web applications, providing access to enterprise information.

Business application users want to see results quickly, sometimes within days, but not always at the expense of 'throw-away' pilot applications. The challenge for Rich Internet Application (RIA) development tools is to operate at the speed of RAD tools, but produce rich, highly functional and scalable enterprise applications based on existing investments in SOA.

This paper covers some of the key advantages and disadvantages of traditional development tools as well as packages and frameworks for building RIAs for SOA and compares and contrasts this with the new generation of tools and a different, designed-for-purpose, approach for building RIAs for SOA. This approach can reduce the development time of RIAs for SOA by at least 60%.

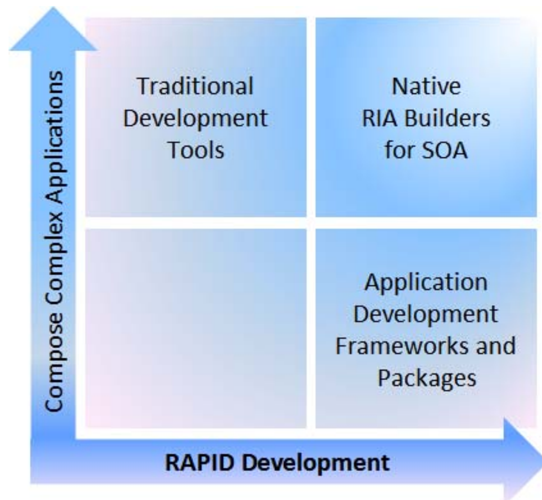
Complexity of RIAs for SOA

Rich Internet Applications that are assembled based on Service-Oriented Architectures can be technically complex, with increased levels of distribution and the need to address significantly more technologies during development. This can lead to increased development timescales, greater project risks and maintenance overheads. The increased development overheads for such projects can be difficult to address using traditional development tools. Traditional tools and methods were not designed for assembling RIAs for SOA. These RIAs have a very different architecture and information model, which can pose difficulties for traditional tools but, at the same time, provide valuable assets that can significantly accelerate development if the information assets are used correctly by the right tools.

Why do we need a new approach for developing RIAs for SOA?

Enterprise applications have traditionally been centralised. This has typically meant a centralised database and business logic that have been accessed using 'thin-clients' of little intelligence. With the advent of client-server, implementations largely separated the UI from the data and business logic, although a large number of UIs have been constructed using direct mappings to the underlying data model with the resulting systems having very 'fat-clients'. Web based applications on the other hand have largely been static, with a web server serving pages that perform limited integration on the server, either with other enterprise systems or providing access to a local database. These systems have predominantly been built using 3GLs such as Java or 4GL tools to accelerate development with resulting loss in overall fine-level control of the underlying application.

Copyright (c) 2006 Hyfinity Limited. All rights reserved.



The architecture of new generation RIAs for SOA is much more complex due to the location independence and the peer-to-peer nature of services. A typical application may use significantly more technologies and architectural layers compared to centralised or client-server enterprise systems. Therefore, as the complexity and the distributed nature of an application increases, the speed of development decreases, resulting in missed deadlines and increased overall project costs.

For this reason it is vital to choose the right approach for this type of application development task.

Where does all the development time go? The anatomy of RIAs for SOA

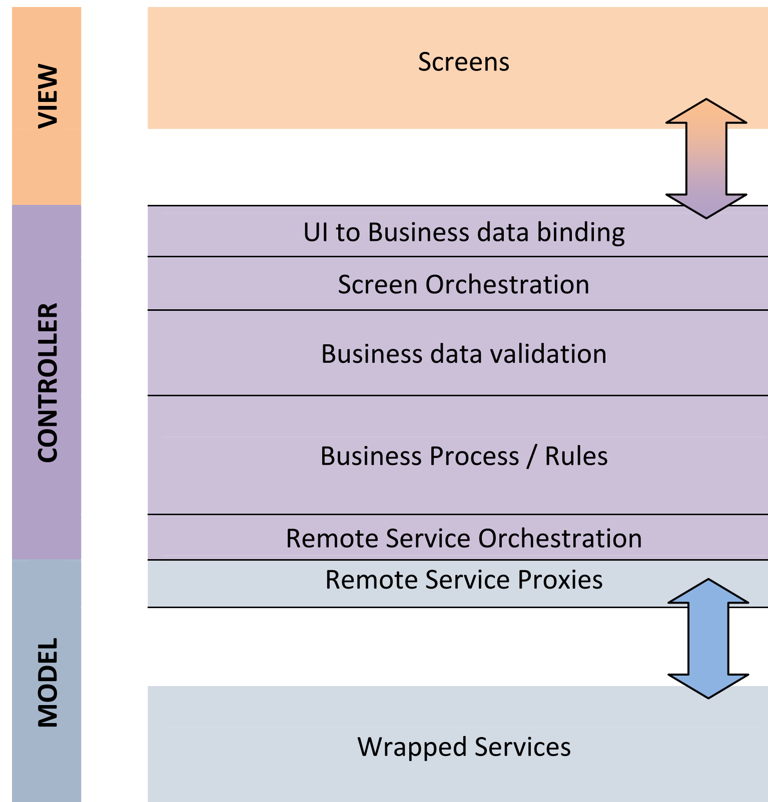
Let's have a high-level look at the design and implementation of a typical RIA. In order to build a properly architected RIA for SOA, with the agility businesses demand, it is important to consider a layered approach to development. Often, applications are built with incomplete designs due to time, resource and cost pressures. These applications may also omit layers quite simply due to lack of knowledge or governance.

Please note that the layers discussed here are not meant to be concrete, but provide an example of components to consider during development of agile rich business applications. The exact architecture of RIA applications is determined by the type of application being constructed. Let's consider the following as examples:

- Legacy application modernisation
- High-touch transactional web applications
- Enterprise Mashups
- Contact Centre CRM systems and line of business systems integration
- Accessible Self-Service Websites

For illustration purposes we will use the legacy modernisation scenario to illustrate the differences in approach that may be undertaken using traditional development and one using the RIA for SOA approach. The architecture diagram below splits a RIA for SOA architecture into three main layers (MODEL, VIEW and CONTROLLER). The layers are more representative of a deployment architecture, but during development each layer will contain sub-layers which, implemented together, provide a complete RIA for SOA.

Copyright (c) 2006 Hyfinity Limited. All rights reserved.



Whereas the diagram above contains a deployment architecture, the table below contains further detail, containing the additional resources that need to be considered during development. The same layering approach is used for clarity and reference.

	Component/Layer	Component Details and Notes
1.0	**MODEL Layer - Wrapping Legacy Services**	
1.1	MODEL - WSDL Design	This is the process of designing and documenting the WSDL files for the services to be wrapped.
1.2	MODEL - Schema and Message Design	This is the schema and message information that will typically be 'included' or 'embedded' in the WSDLs and provides information about how the middle (business logic) layer will interact with the wrapped legacy services.
1.3	MODEL - Actions	These are the operations that will be supported by the wrapped services.
1.4	MODEL - Service Implementation	This is the effort to actually implement the web service and wrapper functionality. In this scenario, the bulk of these services are likely to already exist in a SOA environment and the actual effort to implement these services will not be analysed further.
1.5	MODEL - Service Integration Layer (Proxies for the Wrapped services)	

Copyright (c) 2006 Hyfinity Limited. All rights reserved.

	Component/Layer	Component Details and Notes
1.5.1	MODEL - Fault Design and Error Handling	This layer defines the messages and structures for handling errors between the business logic layer and the remote web services. In this illustration, it is assumed that a generic method is adopted for handling all faults, including design of fault message structures, style and method of display, rather than handling individual errors. If specific code is required for specific error codes then this is treated as an exception rather than the norm.
1.5.2	MODEL - Implementation of proxies	These proxies provide the capability to wrap messages, make calls to the wrapped services, typically over HTTP SOAP, and handle responses.
2.0	**VIEW Layer - User Interface**	
2.1	VIEW - CSS Design	CSS for the overall application and additional stylesheets for specific pages if necessary.
2.2	VIEW - Site and Page Structure Design	This task provides design information and XML assets that define the structure and layout of information within application screens. Location of menus, headers and footers for example.
2.3	VIEW - Page implementation	This is the main task of actually creating ('painting') each of the application screens and typically represents a major part of the effort, especially when you consider that the rendered page has to provide a focal point for other aspects, including CSS, Javascript, dynamic presentation, accessibility considerations and more.
2.4	VIEW - UI Data validation	This is the client-side validation logic, typically JavaScript.
2.5	VIEW - UI Integration Layer	
2.5.1	VIEW - UI Message Wrapper Design	This is an agreed ('consistent') message structure designed to communicate between the screens and the server-side handlers of the POSTed information.
2.5.2	VIEW - UI to Business data binding	This layer provides the binding capabilities between the screen information and the server-side handlers for processing the POSTed information. For example, XHTML to XML.
3.0	**CONTROLLER Layer - Screen and Service Orchestration**	
3.1	CONTROLLER - Business data validation	Additional data validation can be undertaken in this layer. For example, if certain items of information are not available within the client then this information can be used to perform validation on the server.

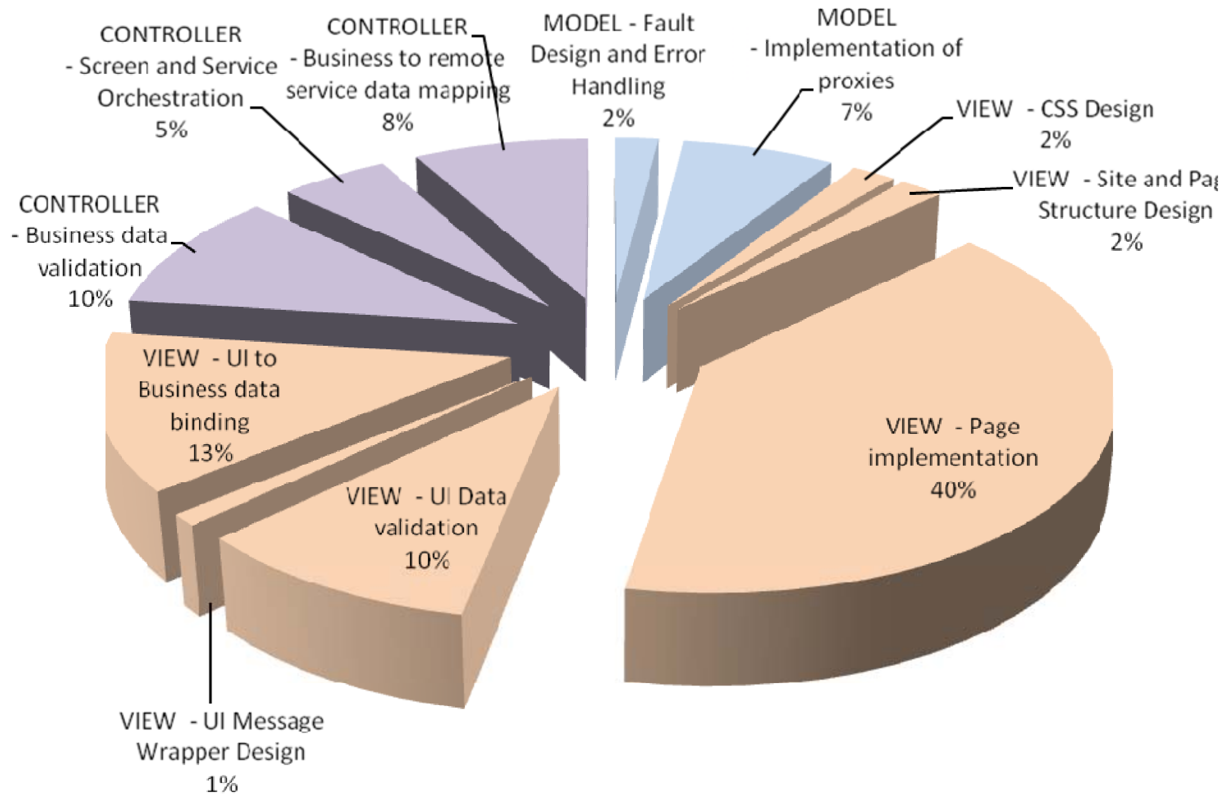
Copyright (c) 2006 Hyfinity Limited. All rights reserved.

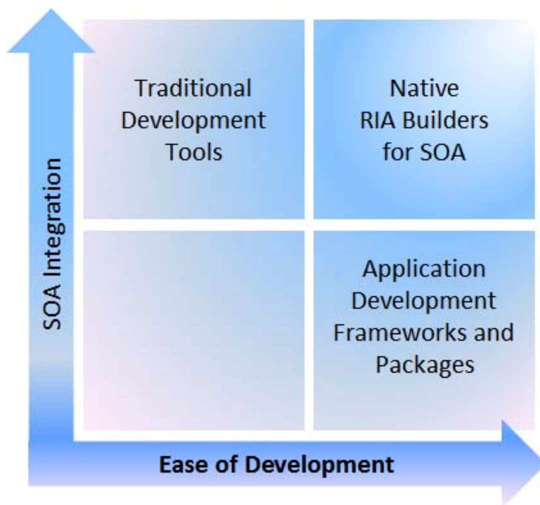
	Component/Layer	Component Details and Notes
3.2	CONTROLLER - Screen and Service Orchestration	Also within this layer, rules are required to sequence the screens and the orchestration of the calls to wrapped remote services within the SOA for composition of screens and also sending captured information back to the wrapped services.
3.3	CONTROLLER - Business Rules	This element can contain business rules to define service logic in the middle-tier or enhance the business logic in the wrapped legacy services. For the purposes of this discussion it is assumed that additional logic is not required in this layer.
3.4	CONTROLLER - Business Processes	Additional business logic and flow components can be incorporated in this layer, such as BPM components. This is not considered for the legacy modernisation example discussed here.
3.5	CONTROLLER - Business to remote service data mapping	This task provides mapping/binding capabilities between message structures within the middle-tier and the message structures within the wrapped legacy services.

As you can see, even at a high level, there are a significant number of layers that require modelling, design and implementation, which adds complexity and introduces a range of technical challenges for the average RIA for SOA developer.

In order to further analyse the development effort, the following chart provides a breakdown of time for an example system, together with the effort to implement various parts of the application and architecture. Since our focus here is on the UI, we will assume that the legacy applications have been wrapped and the SOA services are available for development (layers 1.1 to 1.4 in the above table). Also, we will assume that the business logic layer is ‘thin’ or supplied by another technology such as BPM (layers 3.3 to 3.4 in the above table). In reality, within legacy modernisation scenarios the business logic is primarily resident within the legacy applications that are being wrapped.

Copyright (c) 2006 Hyfinity Limited. All rights reserved.





It is interesting to analyse the areas where most of the time is spent during development. Often, you may find that the 'technical aspects', which do not have a direct contribution to the business value of an application, take up much of the effort within large parts of traditional development. This is often due to lack of separation of certain parts of the application or the right tools are not used for this type of application. Later in this paper we will explore how the effort in certain areas can be significantly reduced using a native XML development approach.

What is the word 'agility' I keep hearing about?

You may be wondering why there are so many layers. These layers add flexibility to the overall application architecture. Unfortunately, during many developments you may be lucky to find even the three major layers (MODEL, VIEW, CONTROLLER). For example, all the steps within the VIEW layer (layers 2.1 to 2.5.2 in the above table) are often implemented in the same step. This completely

Copyright (c) 2006 Hyfinity Limited. All rights reserved.

removes the ability to independently change different aspects of the UI. For example, the ability to change look-and-feel without impacting the validation logic, the screen layout or the data bindings.

The separation of these layers is vital to provide the level of agility required by modern business applications. For this reason tools that are designed to handle multi-layered peer-to-peer applications are more adept at handling such changes.

The role of XML in RIAs for SOA

As you may be aware XML and the transportation of XML information over SOAP plays a vital role within RIAs for SOA. The communications between many of the layers is conducted using XML. Consider the role of XML in the following table.

	Component/Layer	XML Asset
1	**MODEL Layer - Wrapping Legacy Services**	
1.1	MODEL - WSDL Design	WSDL
1.2	MODEL - Schema and Message Design	XSD, XML (part of WSDL)
1.3	MODEL - Actions	part of WSDL
1.4	MODEL - Service Implementation	Traditionally achieved using coding and/or configuration
1.5	MODEL - Service Integration Layer (Proxies for the Wrapped services)	
1.5.1	MODEL - Fault Design and Error Handling	XSD, XML. May also include XSL for transformations
1.5.2	MODEL - Implementation of proxies	Traditionally achieved using coding and/or configuration
2	**VIEW Layer - User Interface**	
2.1	VIEW - CSS Design	CSS
2.2	VIEW - Site and Page Structure Design	XHTML, XSL
2.3	VIEW - Page implementation	XHTML, XSL
2.4	VIEW - UI Data validation	JavaScript
2.5	VIEW - UI Integration Layer	
2.5.1	VIEW - UI Message Wrapper Design	XSD, XML
2.5.2	VIEW - UI to Business data binding	XPath

Copyright (c) 2006 Hyfinity Limited. All rights reserved.

	Component/Layer	XML Asset
3	**CONTROLLER Layer - Screen and Service Orchestration**	
3.1	CONTROLLER - Business data validation	Traditionally achieved using coding and/or configuration
3.2	CONTROLLER - Screen and Service Orchestration	Traditionally achieved using coding and/or configuration
3.3	CONTROLLER - Business Rules	Traditionally achieved using coding and/or configuration
3.4	CONTROLLER - Business Processes	Traditionally achieved using coding and/or configuration
3.5	CONTROLLER - Business to remote service data mapping	Traditionally achieved using coding and/or configuration

The XML information flow and the native processing of this information between the layers provides the greatest opportunity to accelerate development and, at the same time, provides a more agile and maintainable RIA. Traditional approaches encounter great difficulties processing such vast amounts of XML information and the transformations between the different layers. To illustrate this point let's revisit the implementation steps again, but this time assume we have a choice on how we implement each of the stages in the different layers, without being constrained by a particular 3GL or some other traditional implementation tool.

Model Driven Rich Application Features

Now that we have discussed the importance of XML, let's translate this into speed and how this helps us achieve our goals more rapidly. The following table provides details of possible time savings:

	Component/Layer	Native XML approach for building RIAs for SOA
1	**MODEL Layer - Wrapping Legacy Services**	
1.1	MODEL - WSDL Design	Not applicable for this example scenario. It is assumed that the remote legacy services have been implemented/wrapped.
1.2	MODEL - Schema and Message Design	Not applicable for this example scenario
1.3	MODEL - Actions	Not applicable for this example scenario
1.4	MODEL - Service Implementation	Not applicable for this example scenario
1.5	MODEL - Service Integration Layer (Proxies for the Wrapped services)	

Copyright (c) 2006 Hyfinity Limited. All rights reserved.

	Component/Layer	Native XML approach for building RIAs for SOA
1.5.1	MODEL - Fault Design and Error Handling	In RIA for SOA environments the error and fault information will typically be returned via an XML fragment. Using native XML handling and orchestration capabilities it is possible to handle, format and display error information without coding.
1.5.2	MODEL - Implementation of proxies	Since WSDL files contain information about the locations of services, the operations they support and also the messages they can send and receive, it should be a simple and intuitive process to use this information to generate proxy services that can interact with actual SOA services. There should be no need to implement significant amounts of code because all information required for interaction already exists.
2	**VIEW Layer - User Interface**	
2.1	VIEW - CSS Design	
2.2	VIEW - Site and Page Structure Design	
2.3	VIEW - Page implementation	Similar to the implementation of proxies, much of the information required for data capture is actually present in the WSDL files or other W3C Schemas within the overall design of the RIA for SOA. If this information can be used as a starting point for the creation of screens then it should significantly accelerate the development process. This provides a more intuitive and reusable process of screen development rather than 'painting' screens and then trying to figure out how to map this information to the various layers of the overall application architecture.
2.4	VIEW - UI Data validation	Using the model-driven approach, the same schema information can be used to perform validation. Schemas often contain rich data constraint information, which can be leveraged to automatically create validation logic, including dynamic data dependencies. This approach provides better reusability and cohesion between the layers of the application, rather than writing lots of code, which doesn't have inbuilt knowledge of the relationships between the data and between the different layers. Using the model-driven approach, all aspects of data validation, including cross-field validation, dynamic visibility and other rich application features, such as tabbing, is possible based on the data grouping structures within the information model.
2.5	VIEW - UI Integration Layer	

Copyright (c) 2006 Hyfinity Limited. All rights reserved.

	Component/Layer	Native XML approach for building RIAs for SOA
2.5.1	VIEW - UI Message Wrapper Design	HTML forms post information as name/value pairs. Although there are many different techniques for capturing this information on the server, the information ultimately ends up as an XML fragment somewhere within the overall application architecture. It is highly beneficial to have the ability to treat this information in a consistent manner, similar to SOAP wrappers for web service interactions. It should also be possible to map XHTML information to XML fragments between client and server, saving lots of bespoke code to handle the receipt and conversion of fields in an ad-hoc manner.
2.5.2	VIEW - UI to Business data binding	One of the most mundane and time consuming tasks during RIA for SOA developments is the need to map the XHTML information posted by browsers to underlying XML information models. This often entails the decomposition of the data into structures that programming languages understand. Given the native XML nature of this information this layer presents one the greatest opportunities to save time. It should be possible to intuitively map the information bi-directionally to and from the screens. XPath was created for the manipulation of native XML information and using this approach for data mapping is the obvious choice rather than writing lots of programming code, which is difficult to write, maintain and understand.
3	**CONTROLLER Layer - Screen and Service Orchestration**	

Copyright (c) 2006 Hyfinity Limited. All rights reserved.

	Component/Layer	Native XML approach for building RIAs for SOA
3.1	CONTROLLER - Business data validation	Similar to the arguments for data mapping in the previous paragraph, it should be possible to natively process information that is in XML format. Given the large amounts of XML information present in RIA for SOA applications, the ability to process such information natively provides an opportunity to save time and make the development of business logic and orchestration easier. Rules of the form ***'If Condition Then Action...'*** can be invoked based on data and document structures. The resulting Actions should be able to process the XML information natively rather than relying on the breakdown of the XML information into code structures. For example, when a quotation request for a loan is made, it would be ideal to be able to specify: ***'If an applicant is between the age of 18 and 60 then call the Quotation Service, forwarding the loan application data and capturing the response from the Quotation service, ready for further processing'***. It would be extremely useful if the actual logic looked like the specification in the above statement. Unfortunately, code created using most traditional development methods will look nothing like the specification statement. Quite simply because a lot of code is required to decompose the XML documents, process the information contained within the document, consider the SOAP transportation mechanism and other technical plumbing aspects, losing the business context somewhere in the middle. Therefore, having a native rules environment that 'understands' XML information and SOA architectures can save a lot of time during development and also provide simpler more intuitive steps to implement complex RIAs for SOA .
3.2	CONTROLLER - Screen and Service Orchestration	Using a similar approach to the business data validation rules it is possible to orchestrate the delivery of screens and the interactions with remote services based on a data-driven model. Again, the ability to write orchestration rules based on native XML information is important, remaining focused on the business problem rather than the plumbing.
3.3	CONTROLLER - Business Rules	Not applicable for this example scenario. It is assumed that the Business Rules are primarily provided by the wrapped services.
3.4	CONTROLLER - Business Processes	Not applicable for this example scenario. If required then business process functionality will be provided by a suitable BPM product.
3.5	CONTROLLER - Business to remote service data mapping	Once the data has been captured and validated and ready for transmission to the remote services, it is useful to agree standard wrapping capabilities for the information based on HTTP SOAP. The business rules orchestration mentioned earlier, together with the wrapped messages, can be used to provide a consistent and codeless mechanism for mapping XML information natively between the middle tier and the remote legacy services.

Copyright (c) 2006 Hyfinity Limited. All rights reserved.

To illustrate how the development of these layers can be simplified using a native XML approach, it is worth remembering that RIAs for SOA are much more document-centric than code-centric. The ability to process XML information natively using a declarative approach therefore provides accelerated development and also provides better visibility of the application flow because the documents do not have to be decomposed into program code. This ensures that business and technical people can communicate better during the design and development processes.

Why do we need new development tools for building RIAs for SOA?

Using some of the labour-saving steps discussed above it should be possible to save vast amounts of time during the construction of certain layers of RIAs for SOA. The following table provides an example model of the possible savings that can be achieved.

The 1000 man day project below is based on some real life examples, with effort distributed across the layers based on the percentage distribution model discussed earlier. Please note that all parts will vary depending on the type of application being constructed.

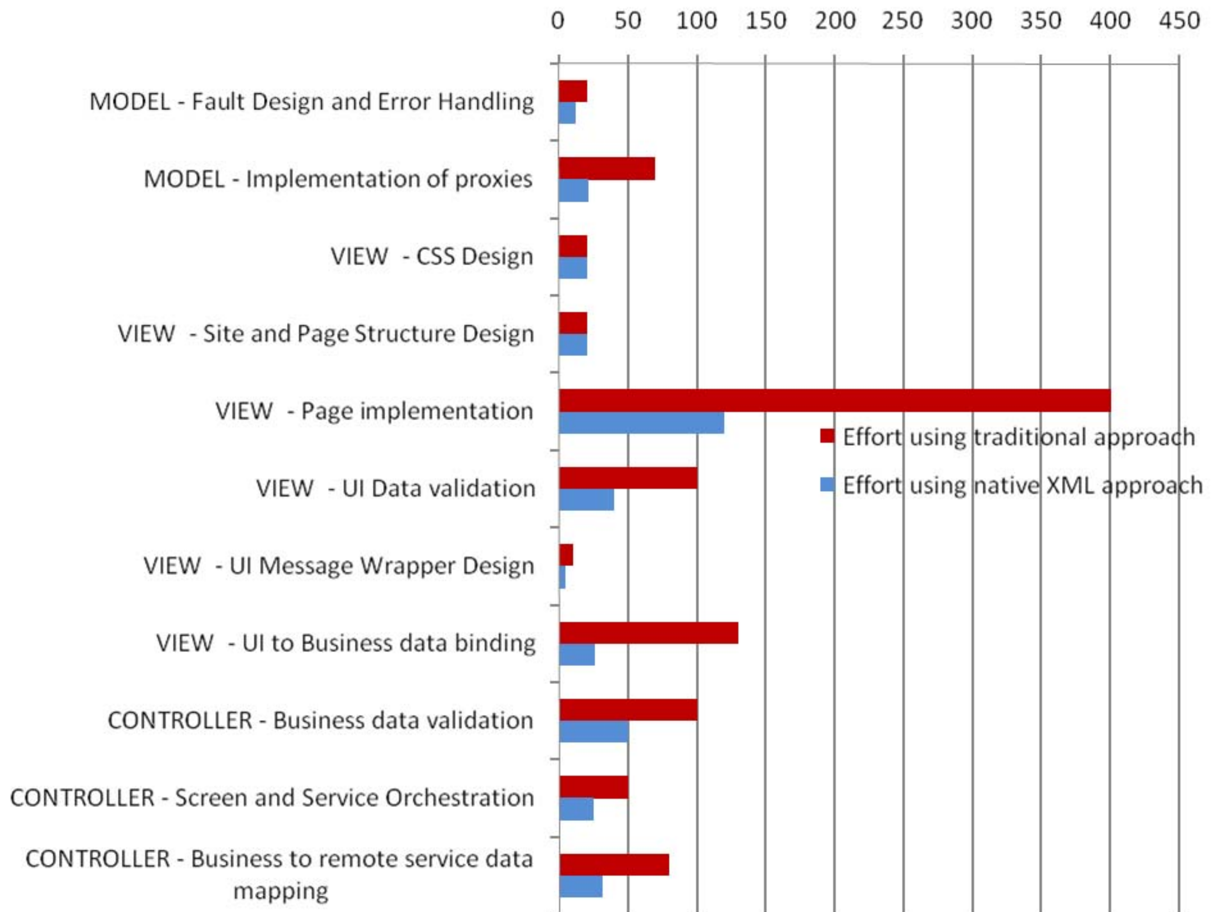
Component/ Layer	% Effort Distribution	Effort (days) using traditional approach	% Saving using XML approach	Saving (days) using XML approach	Effort (days) using XML approach
MODEL Layer - Wrapping Legacy Services					
MODEL - WSDL Design	Not applicable for this example. It is assumed that the remote legacy services have been implemented/wrapped.				
MODEL - Schema and Message Design					
MODEL - Actions					
MODEL - Service Implementation					
MODEL - Service Integration Layer (Proxies for the Wrapped services)					
MODEL - Fault Design and Error Handling	2%	20	40%	8	12
MODEL - Implementation of proxies	7%	70	70%	49	21
VIEW Layer - User Interface					
VIEW - CSS Design	2%	20	0%	0	20
VIEW - Site and Page Structure Design	2%	20	0%	0	20
VIEW - Page implementation	40%	400	70%	280	120

Copyright (c) 2006 Hyfinity Limited. All rights reserved.

Component/ Layer	% Effort Distribution	Effort (days) using traditional approach	% Saving using XML approach	Saving (days) using XML approach	Effort (days) using XML approach
VIEW - UI Data validation	10%	100	60%	60	40
VIEW - UI Integration Layer					
VIEW - UI Message Wrapper Design	1%	10	50%	5	5
VIEW - UI to Business data binding	13%	130	80%	104	26
CONTROLLER Layer - Screen and Service Orchestration					
CONTROLLER - Business data validation	10%	100	50%	50	50
CONTROLLER - Screen and Service Orchestration	5%	50	50%	25	25
CONTROLLER - Business Rules	Not applicable for this example scenario. It is assumed that the Business Rules are primarily provided by the wrapped services.				
CONTROLLER - Business Processes	Not applicable for this example scenario. If required then business process functionality will be provided by a suitable BPM product.				
CONTROLLER - Business to remote service data mapping	8%	80	60%	48	32
Total Effort	**100%**	**1000d**		**629d**	**371d**
Total Savings					**63%**

From the table above, it should be possible to save approximately 60% of the effort if the correct tools are used for developing RIAs for SOA. Much of this is achieved by reusing the information models that are readily available within RIA for SOA specifications and also having the intuitive features that can automate and handle the processing and distribution of XML information between the different layers of RIA for SOA natively. This approach becomes highly productive when using vertical industry XML standards, which are under the control of external bodies.

Copyright (c) 2006 Hyfinity Limited. All rights reserved.



Conclusion

Enterprise Web 2.0 applications have very complex architectures and, like all applications, will prove more complex than originally anticipated. It is therefore important to ensure you have considered all the layers of such applications, including 'hidden extras'. Using traditional approaches to build such applications can often be costly, with resulting applications being more rigid and difficult to maintain. If the speed of development is of the essence, it is important to ensure that this is not achieved at the expense of flexibility and future maintainability.

The new generation of RIA for SOA Builders are designed-for-purpose, providing better support for AJAX and integration requirements with SOA services. This can provide accelerated development with a reduced skillset, without losing context of the overall application flow hidden deep within program code.

Copyright (c) 2006 Hyfinity Limited. All rights reserved.

MVC – Rapid RIA Builder for SOA

MVC is a *Rapid* Rich Internet Application Builder for Service-Oriented Architectures. Unlike other tools, MVC has a complete design and deployment environment that can be used to build Enterprise Web 2.0 applications without any code, providing the *simplicity* of 4GL RAD environments. Our customers have built solutions, including:

- Legacy application modernisation
- High-touch transactional web applications
- Enterprise Mashups
- Contact Centre CRM systems and line of business systems integration
- Accessible Self-Service Websites

Affordable Quality

MVC is an *affordable* product and has provided savings of at least 60% during enterprise application development. MVC can be used to produce consistent and better engineered solutions that are standards-based and easier to maintain, with the agility to keep pace with rapidly changing business models.

Rapid automation through model-driven development

Using a model-driven development approach MVC can automate many of the mundane development tasks associated with traditional development tools and frameworks.

- Automatic generation of web service proxies (provides interactions with SOA services)
- Automatic generation of web pages
- Automatic data binding between XHTML and XML
- Automatic generation of validation logic
- Accelerated development through native XML and AJAX compatibility

To learn more you can download a free copy of MVC and learn how you can develop applications in a matter of minutes for yourself. Please contact us at www.hyfinity.com if you wish to discuss your specific project requirements.

Copyright (c) 2006 Hyfinity Limited. All rights reserved.

About Hyfinity

Hyfinity is a privately owned software company founded in 2001. Hyfinity's mission is to simplify the development of Enterprise Web 2.0 applications by automating mundane developments tasks and reusing readily available information models.

Hyfinity has customers worldwide and is known for its innovation in rapid development tools for building Rich Internet Applications for Service-Oriented Architectures.

We work with our customers and partners to ensure solutions are designed and developed rapidly without compromising quality and scalability. We have worldwide OEM agreements with partners and some of our customer solutions include:

- Dynamic data-driven self-service web applications and eForms
- Feature rich, AJAX-based high transaction e-Commerce and internal solutions
- Contact Centres - CRM integration with existing line of business systems
- Service-oriented web application development for COTS Package Developers

Please contact us if you wish to discuss your specific project requirements or learn more about Hyfinity's RIA for SOA technology and how it can help accelerate your next project development.

Hyfinity Limited
Innovation Centre, Central Boulevard
Blythe Valley Park, Solihull B90 8AJ, United Kingdom
(T) +44 (0)121 506 9111
(F) +44 (0)121 506 9112
(E) info@hyfinity.com
(W) www.hyfinity.com

Copyright (c) 2001-2006 Hyfinity Limited. All Rights Reserved.