

Rise of the SOA Machine for Enterprise Web 2.0 and SOA

Enterprise Web 2.0 >>> FAST

White Paper – September 2007

Why struggle and build code to process and manipulate XML when you can get a machine to do it for you!

Abstract

Enterprise Web 2.0 and SOA applications have massive amounts of XML information within different layers of application architectures, which can range from UI definitions, business logic, data storage and remote service invocation requests and responses. To accommodate this range of XML documents, traditional 3GL OO-based applications are forced to introduce many application code layers to manage the mappings between XML and OO structures and ensure service-oriented loose coupling. Traditional infrastructure software components are being adapted to handle this type of information, but this leads to costly technical architectures and increased lead times for new system deployments.

So, if adapting traditional tools, infrastructure components and development practices is not ideal, is there a better, more dedicated architecture component that can be used in different application scenarios to avoid the need to perform too many translations between different technologies. Enter **The SOA Machine**. Designed for purpose, its native language is XML, which means it 'understands' and 'thinks' in XML - essential for efficient SOA component development. It can perform myriad tasks, often negating the need for expensive enterprise architecture components. The SOA Machine allows Enterprise Application Developers to harness XML in a highly productive and completely service-oriented manner.

Introduction

Enterprise Web 2.0 and SOA applications contain XML documents in massive quantities, ranging from UI data to XML messages that flow between local and remote web services. As a result there is a need to send, receive, manipulate, transform and present XML information in different scenarios and different formats. In order to visualise and understand this information, an XML Information Modelling step can be highly beneficial that models the different documents and their flow through enterprise applications. You may be interested in the paper *“Web Application Development for the SOA Age – ‘Thinking in XML’”*, which provides detailed information on XML Information flows within modern SOA applications.

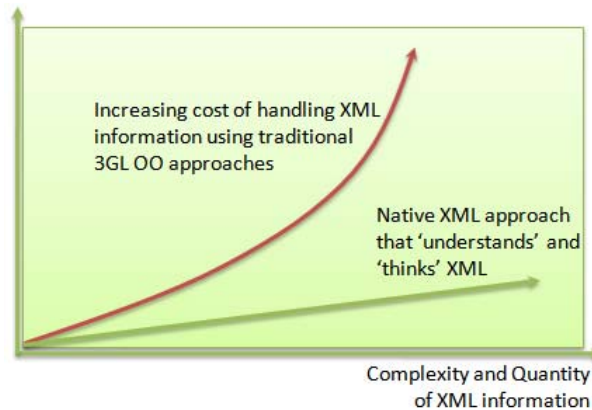


Figure 1. Value of the SOA Machine

Using a 3GL OO-based approach for handling this information within enterprise architectures can be inefficient and the context of the information can be easily lost. A document-centric processing model that ‘understands’ the native XML information models can be highly beneficial, providing better clarity, increased efficiency and faster time-to-value for enterprise Web 2.0 and SOA applications.

Some of the issues that can arise when using traditional 3GL approaches for processing XML information include:

- Development can be complex and time-consuming
- Developers can spend too much time dealing with technical plumbing
- Too much time spent trying to handle and manipulate XML documents within 3GL code. This can be due to the inability to visualise XML document structures as they are ‘hidden’ within object structures.
- Problems with mismatches between native XML formats and 3GL OO code implementations
- As a result deliverables are often late
- Resulting applications are harder to understand and maintain due to loss of the information models, hidden within object structures

Is there a better way to handle this information within Enterprise Web 2.0 and SOA applications?

The SOA Machine is a web-enabled software machine that is capable of receiving and processing XML information natively and orchestrating remote web services to compose composite documents and services using a declarative codeless approach. Unlike traditional approaches, the SOA machine is designed-for-purpose and 'understands' and 'Thinks' in XML rather than having to use traditional programming to constantly map back and forth between objects and XML.

So, technically this is very interesting, but is there some business value associated with the SOA Machine? Using the SOA Machine you can:

- **Improve Time to Value** – Automate manual development stages and processes. Delive to the business faster.
- **Reduce Software Expenditure** - Conduct most of the processing required for Enterprise Web 2.0 and SOA applications using the SOA Machine, often negating the need for expensive ESBs, BPMs and traditional middleware products.
- **Increase Agility** – Create adaptable business applications that can change rapidly due to XML foundations.
- **Reduce Skills Requirements** - Use a uniform higher-level machine, requiring only XML skills. This provides the same development model for different parts of the architecture.
- **Increase Cost Effectiveness** - Requiring less effort and less programming skills.
- **Obtain Value for Money** - Single machine, many uses.

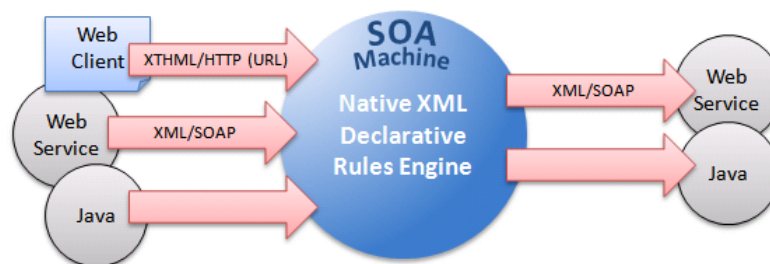


Figure 2. The SOA Machine

This paper will detail the SOA Machine and provide application scenarios within the Enterprise Web 2.0 and SOA domain where the SOA Machine can be used to accelerate development and reduce complexity. An example will be used to illustrate the different stages of an enterprise application and the different roles the SOA Machine can play within such application architectures, but with the same metadata powered software. In essence a 'Morphing' software machine.

The role of the SOA Machine within Enterprise Web 2.0 and SOA

The architecture diagram in Figure 3 is reproduced from the paper “Web Application Development for the SOA Age – Thinking in XML”. Enterprise architecture layers, especially the Business Logic Layer, can be quite complex. Implemented in traditional 3GL code, this layer can disrupt the document flow and act as a ‘non-XML buffer’, leading to clarity and future maintenance issues. The screens used to capture the information required by remote web service interfaces, described using WSDLs, often have a very close mapping. Both the UI and the remote services adhere to XML and a native XML approach can be highly beneficial, providing faster development through reuse of XML assets that result during the modelling phase. Future maintenance can also be much easier due to the self-documenting nature of the XML metadata produced by the Information Modelling phase.

The SOA Machine forms an ideal processing model for this type of application due to its native XML processing capabilities. In Figure 3, the SOA Machine can be used in different parts of the architecture to perform different tasks. The following sections will detail the internals of the SOA Machine and how it performs its tasks. This will be followed by a worked example scenario that will provide some idea of the declarative processing-model used by the SOA Machine that makes it very different to traditional 3GL procedural approaches, which ultimately result in the advantages of this declarative document-centric approach.

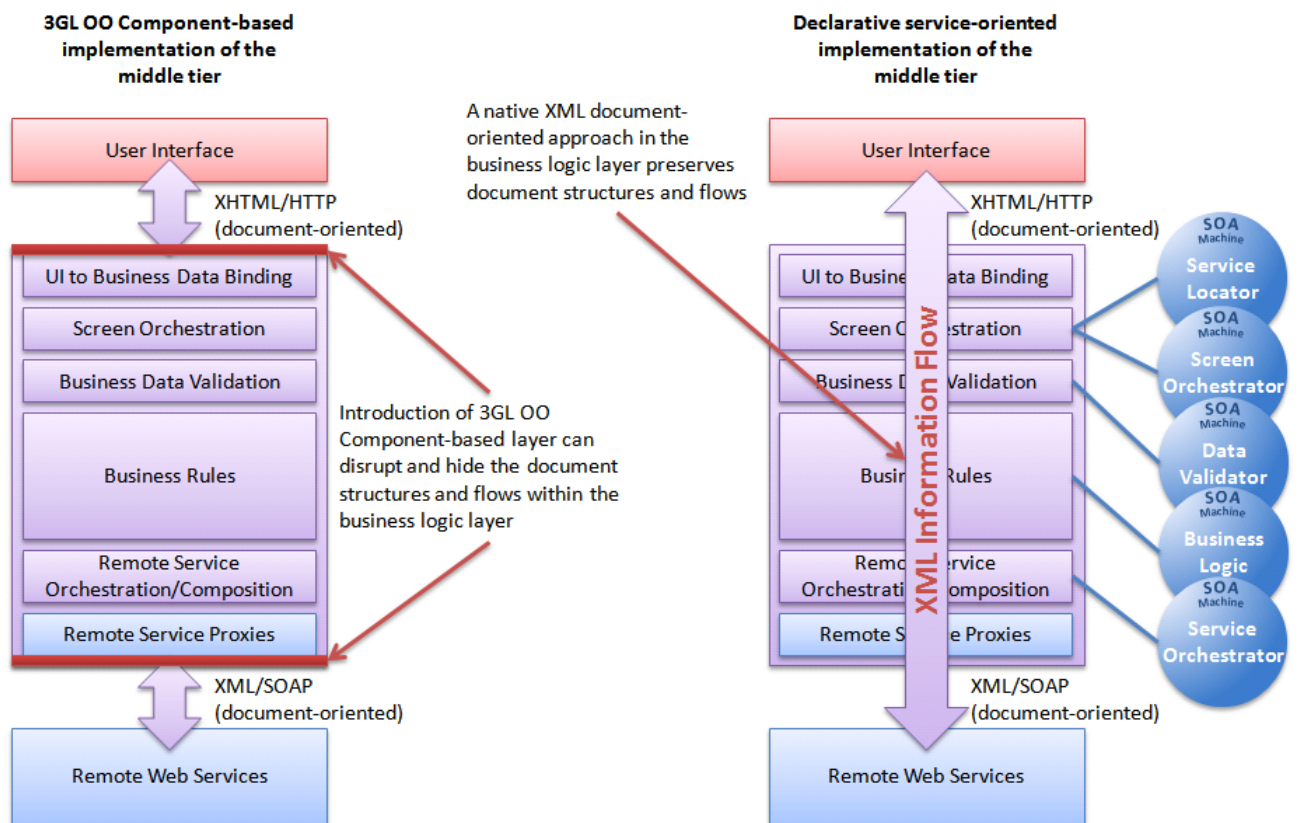


Figure 3. 3GL vs. Declarative implementation of the middle-tier – the role of the SOA Machine

Internals of the SOA Machine

The SOA Machine can receive XML requests from Web Clients, SOAP Services and Java Classes. The Machine can execute a set of declarative rules against these requests, which may result in requests to other SOA Machines and third party web services. Once the SOA Machine has finished its execution it returns a response, which may be XML or non-XML to the original client that made the request.

Figure 4 provides internal details of the SOA machine. SOA Machine instances will typically work together to build larger applications. The SOA Machine is activated when an XML document is received and loaded into the **Factbase**. The Factbase acts as the primary working memory of the SOA Machine. The **Workspaces** shown can be used as secondary working memory to hold any number of XML documents or fragments. Workspaces can be useful in organising XML documents better and preventing the Factbase from growing too large. The Factbase and Workspaces are capable of holding any well-formed XML document, which can be manipulated during execution of the SOA Machine.

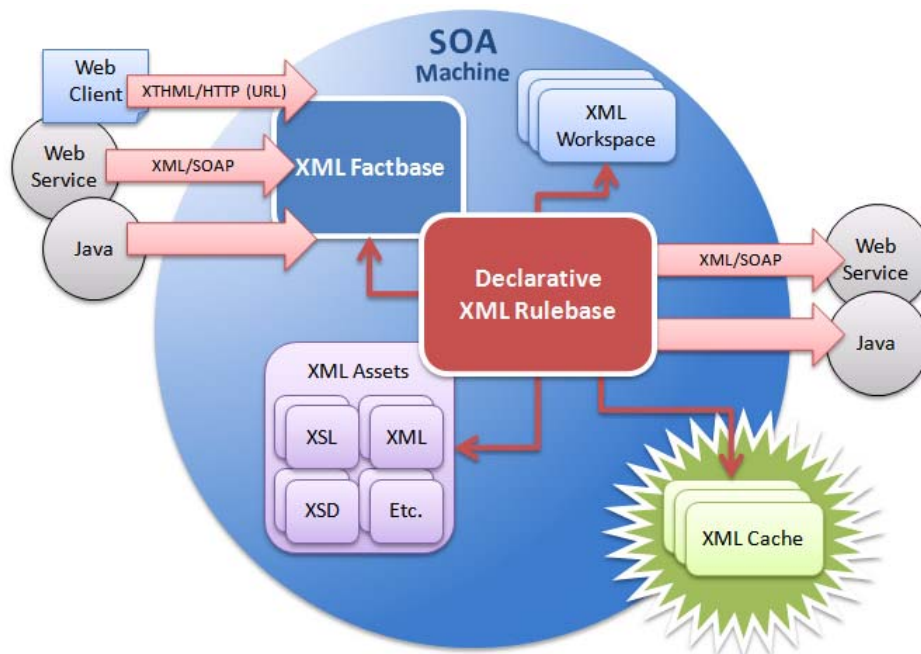


Figure 4. SOA Machine Internals

Once a document has been received and loaded into the Factbase, the rules contained in the **Rulebase** execute against the XML information in the Factbase. As the rules execute additional **XML Assets** (XSDs, XSLs, XML, etc.) may be loaded and used. For example, a schema may be used to validate the incoming request document; a separate schema may be used to validate the different responses from the remote service invocations. An XSL may be used to provide a SOAP Wrapper around the remote service requests. If any information requires caching then this can be stored in the **XML Cache**, capable of holding any number of XML documents or fragments. One of the operations that the SOA Machine can perform is the ability to invoke remote services. Once the responses are returned from the remote services they need to be aggregated to compose the response so it can be returned to the original client that made the request. In addition, we may need to consider the screen flow depending on the responses received from the remote services.

How the different XML documents (requests, responses, cached documents and intermediate working documents) are composed will depend on the actual document modelling stage. The following sections use some example document fragments to illustrate the typical SOA Machine operations and the document structures that may emerge as a result.

The SOA Machine in action

In this section we will work through a simple example that will cover the core principles of the SOA Machine.

Example Overview

A financial portal provides a comparative quotation service for life insurance products, provided by different financial organisations. The portal captures the basic customer details, validates this information and composes requests to different third party web services that return the requested quotations. The portal then aggregates this information and presents it back to the customer that requested the quotation. Figure 5 shows the quotation request screen, which captures some basic information to assess the initial suitability of an applicant for a particular product. Figure 5 also shows the response table containing possible suitable products for the applicant. The network diagram to the right of the diagram shows the request and response screens and the **QuotationEngine** that receives the Quotation request, calls three separate services and composes a response to present to the customer using the quotation response screen.

In this simple example, the quotation request is captured via a web page called **QuoteRequest** and submitted to the web server. Once the information is submitted the UI to Business Data Binding Layer maps the captured XHTML information to the desired XML document structure on the server. Based on the XML quotation document received we will need to locate a SOA Machine that can process the quotation. This is the **QuotationEngine** in this scenario. After Data Validation and application of any Business Logic that is required (for example person to be insured from company xyz needs to be between the ages of 18 and 60) we will need to compose quotation requests for the different remote quotation services provided by the different companies. After the responses have been received we will need to aggregate these responses and present them on a table of quotations.

QUOTATION REQUEST

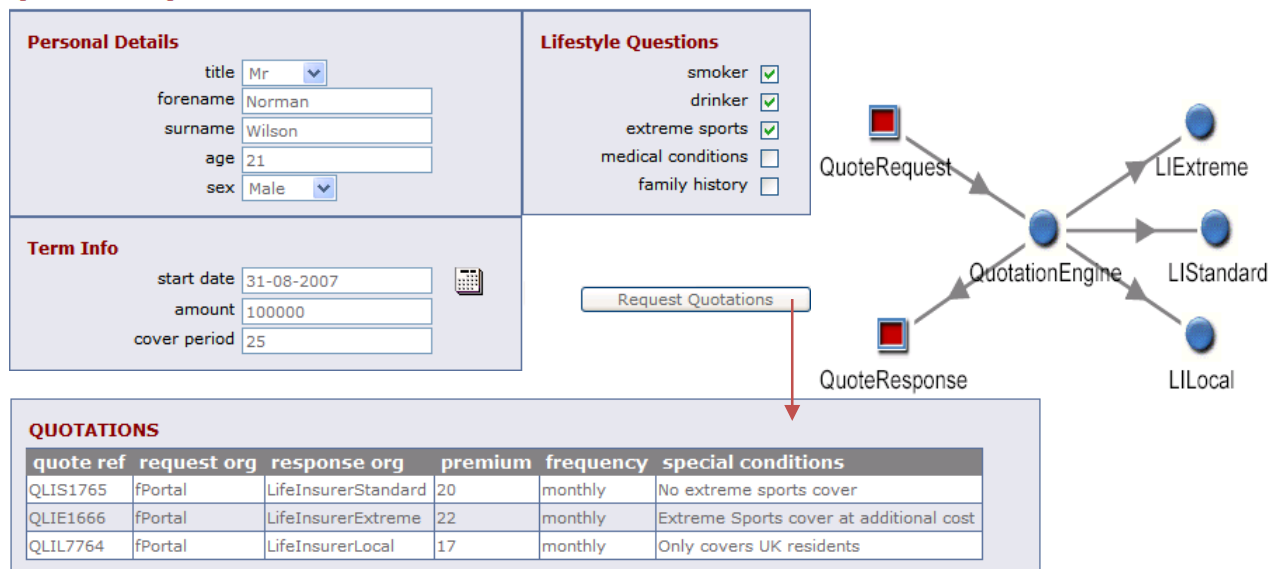


Figure 5. Quote Request Screen, Response Screen and Application Flow

For the rest of this example, we will progress through this application network and look at the role of the SOA Machine and the information that it receives, processes and onward transmits for other SOA Machines and third party services. Again, for simplicity, much of the processing will be undertaken within the **QuotationEngine** SOA Machine. In a real scenario, you would split the functionality into layers as shown in the architecture diagram in Figure 3 and delegate the processing to several SOA Machines.

Invoking the SOA Machine (Inputs)

The SOA Machine can receive requests from XHTML, Java and SOAP clients. The SOA Machine also has the capability to map XHTML information to XML, so there is no need to use non-XML binding frameworks, which means the native XML document flows are preserved.

Service Locator

When an action is received a service needs to be located that will serve the requested action. If the SOA Machine is used in a declarative manner to achieve this requirement then content-based service location makes this process easier. This is the scenario where a request has been received by a SOA Machine and needs to delegate the processing to another machine. The XML listing in Figure 6 has a *Controller* element shown in the *Control* section of the request message. This *Controller* element can be used to determine which SOA Machine is the recipient for this request.

For clarity within this example we will start at the point where the **QuotationEngine** SOA Machine receives the quotation request.

The Factbase

When the Quotation request document is received and the Factbase is loaded with the request, the Factbase may look like the listing shown in Figure 6. In addition to the information received for the quotation request we may need to preserve some header information for our middle-tier to enable service location and screen orchestration. This is the information contained in the header *Control* section of the message. *Please note* that the wrapper used for the message here is purely for illustration purposes, you would use your own wrapper to suit your particular application context. The boxes containing comments in Figure 6 will be explained in more detail in the next few sections.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <mvc:eForm xmlns="http://www.hyfinity.com/xplatform" xmlns:mvc="http://www.hyfinity.com/mvc" xmlns:xg
   = "http://www.hyfinity.com/xgate">
3    <mvc:Control>
4      <is_script_enabled xmlns="http://www.hyfinity.com/mvc">true</is_script_enabled>
5      <Page xmlns="http://www.hyfinity.com/mvc">QuoteRequest.xsl</Page>
6      <Controller xmlns="http://www.hyfinity.com/mvc">mvc-Quotation-QuotationEngine-Controller</Controller>
7      <action xmlns="http://www.hyfinity.com/mvc">getQuotes</action>
8    </mvc:Control>
9    <mvc>Data>
10     <quote_request xmlns="">
11       <quote_ref/>
12       <personal>
13         <title>Mr</title>
14         <forename>Norman</forename>
15         <surname>Wilson</surname>
16         <age>21</age>
17         <sex>Male</sex>
18       </personal>
19       <lifestyle>
20         <smoking>true</smoking>
21         <drinking>true</drinking>
22         <extreme_sports>true</extreme_sports>
23         <medical_conditions>>false</medical_conditions>
24         <family_history>>false</family_history>
25       </lifestyle>
26       <term>
27         <start_date>2007-08-31</start_date>
28         <amount>100000</amount>
29         <cover_period>25</cover_period>
30       </term>
31     </quote_request>
32   </mvc>Data>
33 </mvc:eForm>

```

Make a copy of the **quote_request** fragment to create the request **workspace (LIExtremeQuoteRequest)** for the **LIExtreme** Quote Service

Make a copy of the **quote_request** fragment to create the request **workspace (LIStandardQuoteRequest)** for the **LIStandard** Quote Service

Insert composite container element **quote_responses**, which will be used to contain the responses from different service calls to remote quotation services

Figure 6. Initial Factbase Status

Declarative Logic (Native XML Rules)

Once the Factbase is loaded with the request, the rules within the Rulebase are activated and executed against the Factbase. What types of rules are used against the incoming document will be dependent on the application scenario, but example rules may include the need to validate the incoming request, we may use a rule to cache some information for later use, and so on.

It is important to note that all this processing does not require lots of 3GL plumbing code to enable receipt, manipulation and onward transmission of documents. The actions within the SOA machine remain focused on the business problem using a document-centric approach. Figure 7 shows a Rulebase containing five **Rules**. The first rule inserts a document container (as shown in the third comment box in Figure 6) that will hold the quotation responses from the three separate service calls. The next three rules each call a remote service to obtain a quotation for the request. There is a **LIExtreme** service that covers extreme sports and there is a **LIStandard** service that excludes extreme sports. Both of these are SOAP services and use SOAP Wrappers for their request and response message structures. The third service is a local quotation service called **LILocal**, which is called as a Web Service, but does not use a SOAP Wrapper.

The final rule sets the *Page* element in the *Control* section to indicate which screen to display next. This will be conditional on the outcomes of the other rules in order to assist the **ScreenOrchestrator** to determine which web page to display next. For example, if we have a successful response then we may set the next screen to be the **QuotationResponse** screen, but if all calls failed or did not return any suitable products then we may wish to display an error screen.

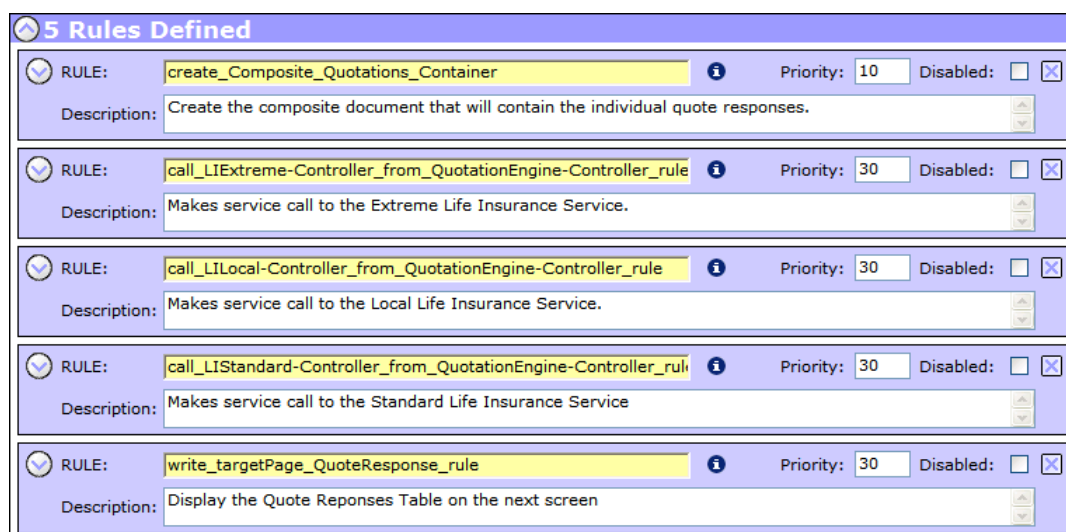


Figure 7. Declarative native XML Rules

Data Validation

In a model-driven development approach you would expect much of the data validation to be undertaken based on the rules within the data model, which means we can conduct the data validation within the client, especially if the logic on the web client can be generated from the underlying schema models. However, additional validation can be undertaken on the server, including cross-field validation.

We could simply apply a schema against the request to validate the information, which is one of the basic standard **Actions** provided by the SOA Machine. Alternatively we may write individual rules to validate different aspects of the information within the Factbase.

Inserting Documents and Fragments

We will focus on the five rules that are defined in Figure 7, the first of which is used to create a container element called *quote_responses* to hold responses from the remote service calls. This first rule is expanded in Figure 8.

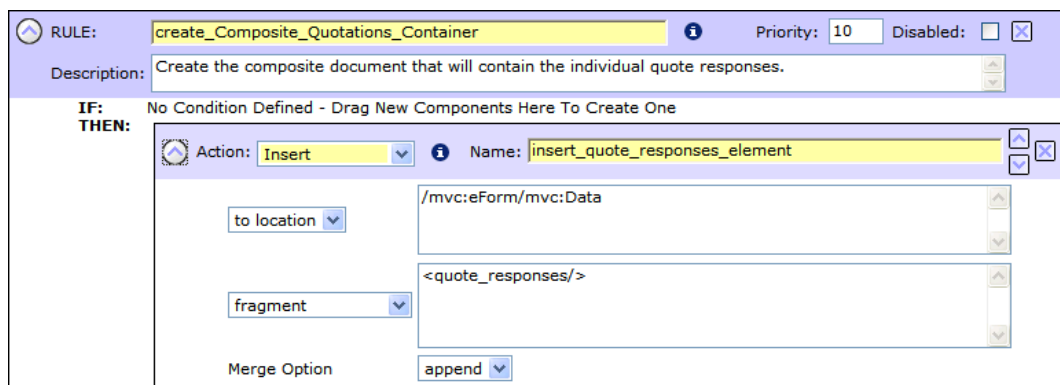


Figure 8. Composition Container Rule

The rules are organized in an 'If Condition Then Actions' format. The condition can be any valid *XPath*, typically based on data in the Factbase. We will see examples of conditions in later rules, but this particular rule is unconditional. The available Actions include **Validate**, **Transform**, **Copy**, **Insert**, **Delete**, Etc. We are using a simple **Insert** action here to insert an XML fragment, in this case the *quote_response* element, but we could have inserted any well formed XML fragment or loaded an XML document from a file if required. The 'to location' item identifies the location within the Factbase or Workspace where the fragment will be inserted, indicated by an XPath. In this case the fragment will be appended after the *Data* element as shown in Figure 6.

Using other XML Assets

Using the information in the Factbase we will construct a request structure for the remote services. Let's have a look at one of the expanded rules that calls a remote web service. Please see Figure 9. This rule has three actions. The first action uses a standard XSLT transformation to add a SOAP Wrapper for the remote service that requires a SOAP Wrapper (the LIExtreme and LIStandard web services will need this action, but Local service will use the unwrapped request structure). With the **Transform** action we can specify the fragment that requires transformation and also the location within the Factbase where the response will be inserted.

Workspaces

As we construct various requests, responses and intermediate documents, the Factbase can become large. The use of Workspaces enables us to structure our XML information better.

As shown in the listing in Figure 6, to avoid 'bloating' the Factbase, we will use a separate Workspace called **\$LIExtremeQuoteRequest** within this rule to hold the transformed request structure. This structure is shown in Figure 10. The structures of the three rules that call the remote services are very similar. The request to the LIStandard service after the transformation will be held in a workspace called **\$LIStandardQuoteRequest**. The request to the LILocal service is unwrapped and the request information can be sent straight from the Factbase, i.e. XPath */mvc:eForm/mvc:Data/quote_request* within the document in Figure 6.

The second action uses the **Invoke Service** operation to call another service. This operation uses a 'from location' to indicate the source of the request structure. This can be a location within the Factbase as well as a location within a Workspace. In our scenario we will use the **\$LIExtremeQuoteRequest** workspace created earlier. The second parameter indicates where the

response will be inserted. This can be inserted in a separate workspace or within the Factbase directly. In this scenario we will insert the response within the composite element *quote_response* that we inserted using the first rule. An example response from this service call is shown in Figure 11.

The screenshot displays the configuration for a rule named "call_LIExtreme-Controller_from_QuotationEngine-Controller_rule". The rule's description is "Makes service call to the Extreme Life Insurance Service." The rule is currently disabled and has a priority of 30.

IF: The condition is "check_service_call". The check expression is:


```
/mvc:eForm/mvc:Data/quote_request/personal/age[.>25] and /mvc:eForm/mvc:Data/quote_request/term/amount[.>500000]
```

THEN: The rule contains three actions:

- Transform:** Action name "add_SOAP_Wrapper". It uses the "Soap_Wrapper.xsl" stylesheet. The source location is "/mvc:eForm/mvc:Data/quote_request" and the target location is "\$LIExtremeQuoteRequest". The merge option is "append".
- Invoke Service:** Action name "call_service". The source location is "\$LIExtremeQuoteRequest" and the target location is "/mvc:eForm/mvc:Data/quote_responses". The merge option is "append". The service is "mvc-Quotation-LIExtreme-Controller" and the action is "Call".
- Copy:** Action name "remove_SOAP_Wrapper". The source location is "/mvc:eForm/mvc:Data/quote_responses/soap:Envelope/soap:Body/quote_response" and the target location is "/mvc:eForm/mvc:Data/quote_responses/soap:Envelope". The merge option is "replace".

Figure 9. Remote Service Orchestration and Document Repurposing Rules

Producing (Outputs) from the SOA Machine

We have used quite a few Actions so far, but the SOA Machine will become quite complex if we were to compose all the requests for all the services. In reality we may use different SOA Machine instances to handle the quotation requests for each of the remote services and simply delegate the standard quotation fragment to each of these dedicated Machines. This ability for SOA Machines to call other SOA Machines very easily enables the construction of larger composite applications.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <soap:Envelope xmlns="http://www.hyfinity.com/xplatform" xmlns:fo="http://www.w3.org/1999/XSL/Format"
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
3    <soap:Body>
4      <quote_request xmlns="">
5        <quote_ref/>
6        <personal>
7          <title>Mr</title>
8          <forename>Norman</forename>
9          <surname>Wilson</surname>
10         <age>21</age>
11         <sex>Male</sex>
12       </personal>
13       <lifestyle>
14         <smoking>true</smoking>
15         <drinking>true</drinking>
16         <extreme_sports>true</extreme_sports>
17         <medical_conditions>false</medical_conditions>
18         <family_history>false</family_history>
19       </lifestyle>
20       <term>
21         <start_date>2007-08-31</start_date>
22         <amount>100000</amount>
23         <cover_period>25</cover_period>
24       </term>
25     </quote_request>
26   </soap:Body>
27 </soap:Envelope>

```

Figure 10. Example Request Workspace

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <soap:Envelope xmlns="http://www.hyfinity.com/xplatform" xmlns:soap="
   http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3    <soap:Body>
4      <quote_response xmlns="">
5        <quote_ref>QLIE1666</quote_ref>
6        <request_org>fPortal</request_org>
7        <response_org>LifeInsurerExtreme</response_org>
8        <premium>22</premium>
9        <frequency>monthly</frequency>
10       <special_conditions>Extreme Sports cover at additional cost</special_conditions>
11       <status>
12         <code>01</code>
13         <description>successful</description>
14       </status>
15     </quote_response>
16   </soap:Body>
17 </soap:Envelope>

```

Figure 11. Remote Web Service Response

XML Document Manipulation

We used the *Insert* Action in the first rule to insert a container element to hold the responses from the remote services. There is a wide range of Actions that can be used to manipulate the structures of documents contained in the Factbase and Workspaces. The third action used is *Copy* to 'strip' the SOAP Wrapper from the response of the two services that communicate via SOAP. This is because the SOAP Wrappers may not be required within the local application context and also the resulting

composite structure under *quote_responses* is more aligned to the final screen data that is required for the *Quotations Response* screen.

The copy action uses *from* and *to* locations. The *from* location indicates the fragment to copy and the *to* location indicates the target where the copied fragment is to be inserted. In this example we copy the response fragment *quote_response* from within the SOAP Body and replace the SOAP Envelope completely. This has the effect of replacing the SOAP Envelope with the *quote_response* contents within the Body of the same Envelope.

Business Logic Development

Using these declarative rules we can also construct business logic based on the information contained in the Factbase and the Workspaces. Within Figure 9 for example we can insert a condition that will ensure the rule will only fire if the applicant is older than 25 and the requested cover amount is greater than £500,000. These rules are declarative, based on the native document structure, and do not require additional data structures. This ensures greater visibility and intuitive handling of XML content, whilst preserving the document structure and flow.

In a rigorous data driven design approach a separate SOA machine would be implemented to decide the data that should be checked for policy related information. This would then define which services are required and ultimately described by business users. A separate SOA Machine would then perform the various Service calls based on data defined by the policy SOA Machine data results.

Web Service Orchestration and Composition

After the three separate services have been invoked and appropriate responses have been received, we should end up with the document structure shown in Figure 13.

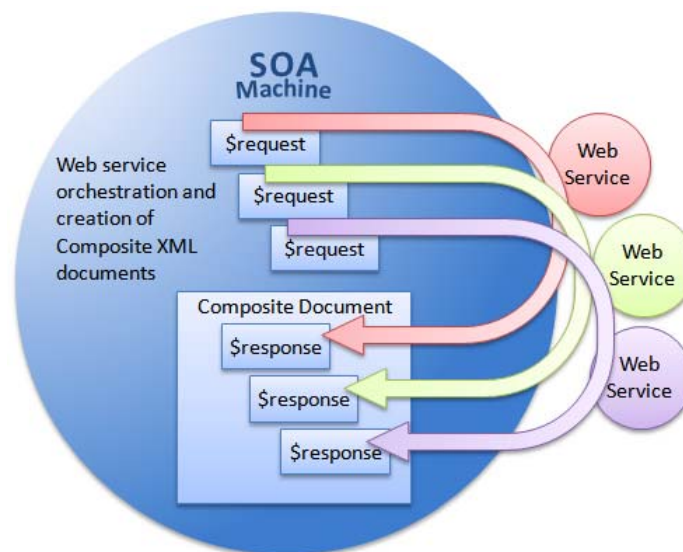


Figure 12. Workspaces, Orchestration and Composite Documents

This shows the three separate *quote_response* elements contained within the *quote_responses* element. Using this approach it is possible to construct composite documents as well as higher-level machines that use services of lower-level machines.

User Interface Orchestration

We will need to decide which screen to display after a SOA machine has finished executing and is ready to return to the customer. In this simple scenario, we may decide to return to the QuoteRequest screen if some of the supplied information is invalid, or return to the QuoteResponse screen if valid quotations were received from the remote services. Or you may wish to display a

Fault (Service Unavailable) message if responses are not received from the remote service invocations.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <mvc:eForm xmlns="http://www.hyfinity.com/xplatform" xmlns:mvc="http://www.hyfinity.com/mvc" xmlns:xg
   ="http://www.hyfinity.com/xgate">
3    <mvc:Control>
4      <is_script_enabled xmlns="http://www.hyfinity.com/mvc">true</is_script_enabled>
5      <Page xmlns="http://www.hyfinity.com/mvc">QuoteResponse.xsl</Page>
6      <Controller xmlns="http://www.hyfinity.com/mvc">mvc-Quotation-QuotationEngine-Controller</Controller>
7      <action xmlns="http://www.hyfinity.com/mvc">getQuotes</action>
8    </mvc:Control>
9    <mvc:Data>
10     <quote_request xmlns="">
32     <quote_responses xmlns="">
33       <quote_response>
34         <quote_ref>QLIE1666</quote_ref>
35         <request_org>fPortal</request_org>
36         <response_org>LifeInsurerExtreme</response_org>
37         <premium>22</premium>
38         <frequency>monthly</frequency>
39         <special_conditions>Extreme Sports cover at additional cost</special_conditions>
40         <status>
41           <code>01</code>
42           <description>successful</description>
43         </status>
44       </quote_response>
45       <quote_response>
46       <quote_response>
47     </quote_responses>
48   </mvc:Data>
49 </mvc:eForm>

```

Figure 13. Composite Documents

Figure 14 shows one example rule that will display the default QuoteResponse page if at least a single *quote_response* has been received. The SOA Machine that called the QuotationEngine will use this *Page* element in the header *Control* section to locate the indicated page and apply a transformation to the returned Factbase to produce the next screen.

RULE:	write_targetPage_QuoteResponse_rule	Priority: 30	Disabled: <input type="checkbox"/>
Description:	Display the Quote Responses Table on the next screen		
IF:	Condition: check_for_successful_response Check: /mvc:eForm/mvc:Data/quote_responses/quote_response/status/code [.= '01']		
THEN:	Action: Assign Name: assign_next_page_name to location: /mvc:eForm/mvc:Control/mvc:Page value: QuoteResponse.xsl		

Figure 14. User Interface Orchestration Rules

Multiple SOA Machines working together

So far the SOA Machine has largely been discussed in the singular context, but multiple SOA Machine instances can be created to act together in myriad combinations.

Processing Sequences

In this scenario, a 'processing sequence' is used to construct a typical SOA application architecture. This is very similar to the example discussed so far.

The **Service Locator** machine receives web requests and delegates them to the appropriate **Data Validator**, which then calls the **Business Logic** machine after data validation. The business logic machine uses the **Service Orchestration** machine to call remote services. Once a typical processing iteration is complete the **Screen Orchestrator** determines which screen to display next.

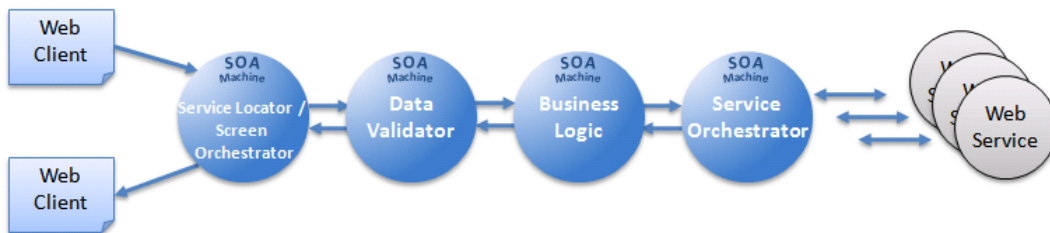


Figure 15. SOA Machines forming processing sequence

Distributed peer-to-peer Networks

SOA Machines can be arranged to act as *autonomous service agents* that can collaborate with other services to create peer-to-peer networks. In this scenario the Finance Portal uses the Life Insurer, the General Insurer and the Mortgage Provider to sell packaged mortgage products - a combined mortgage, life insurance and home insurance product for example.

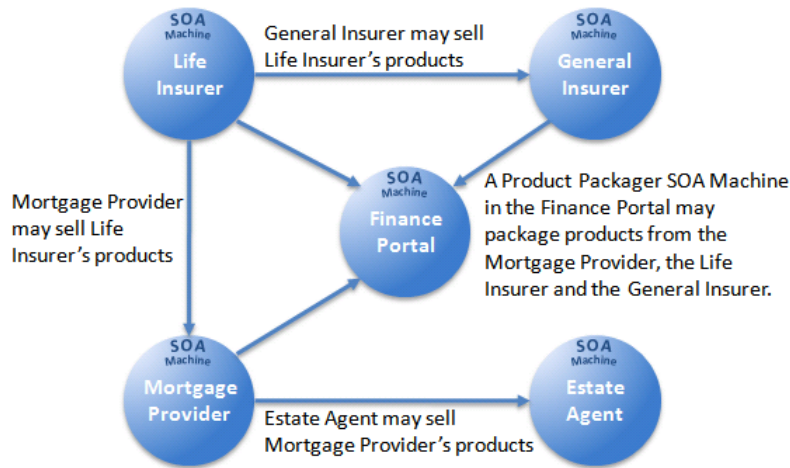


Figure 16. SOA Machines forming peer-to-peer networks

Networks with centralised control

SOA machines can be arranged so that certain machines act as controllers for others. In this scenario, we may have a machine that calls three separate machines, together forming a single transaction. The controlling machine may employ compensating tasks to 'undo' certain operations based on the outcomes of different parts of the transactions.

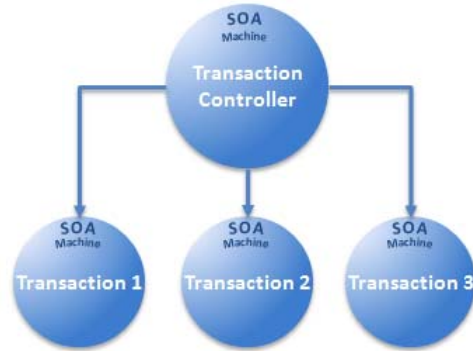


Figure 17. SOA Machines acting as Controllers

The possibilities are endless

The SOA machines can be used in many different configurations to suit myriad application scenarios. Also, remember that each SOA Machine can be distributed over HTTP, providing very flexible deployment scenarios.

Figure 18 shows a combination of the above three example scenarios, which may form the Quotation transaction. For example, the transaction controller may represent the composite quotation service, controlling separate parts of the long transaction. One part of the long transaction calls the quotation service processing sequence, which in turn makes remote service calls to the peer-to-peer network to obtain a quotation for the product.

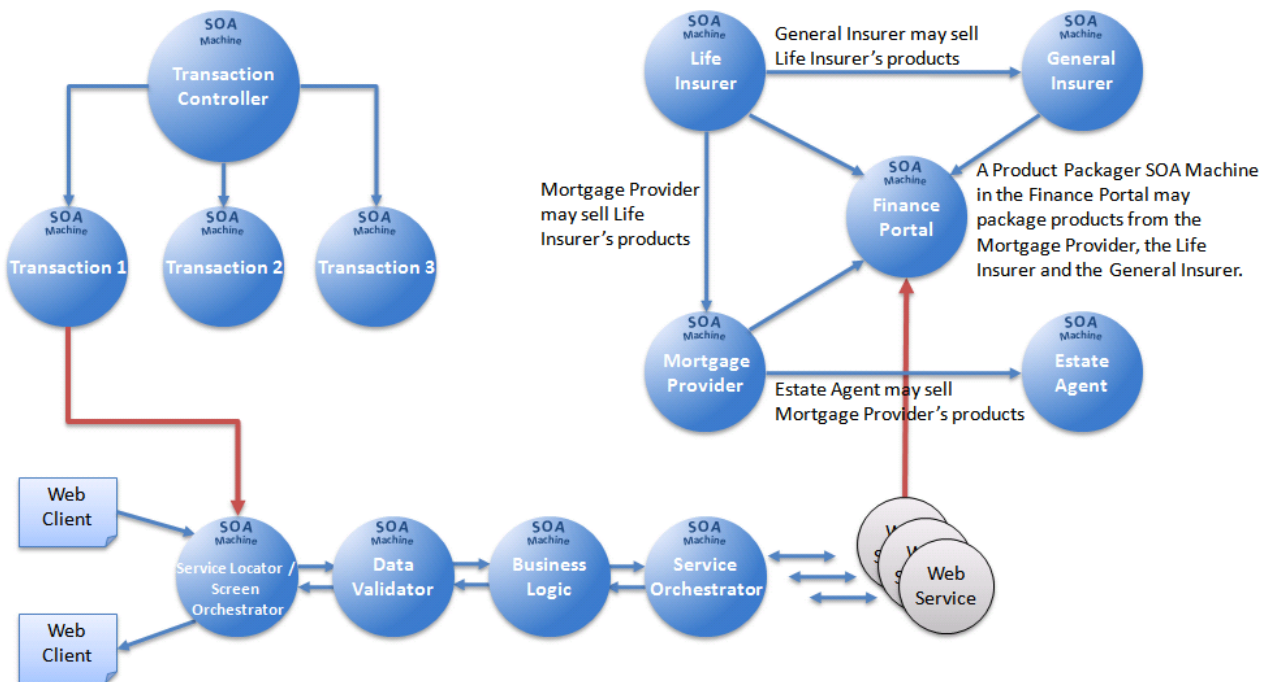
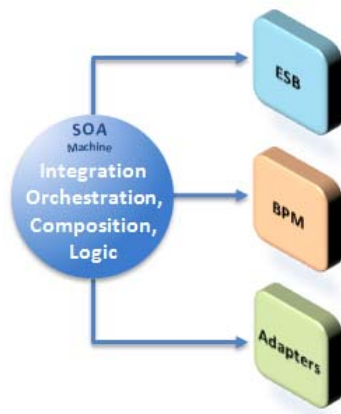


Figure 18. SOA Machines in network of networks

Integration and the SOA Machine



The SOA Machine can be invoked via HTTP URL, XML SOAP and Java applications. The SOA Machine can invoke other services, including web services interfaces of other infrastructure products such as ESBs, BPMs, Adapters, etc. This provides a means to use the SOA Machine with existing SOA components, enabling greater reach for applications constructed using SOA Machines.

Figure 19. SOA Machines in integration scenarios

Scalability and the SOA Machine

The SOA Machine is re-entrant; therefore the logic within *rulebases* is completely transferable. This means that SOA Machines can be pooled, providing scalability that is only limited by physical machine resources. The SOA machine pools have *min* and *max* settings to enable certain machines to be preloaded for busy services and also constrain the upper limits to preserve machine resources where resources are scarce. This provides a highly scalable architecture that can be adapted to suit different applications and usage scenarios.

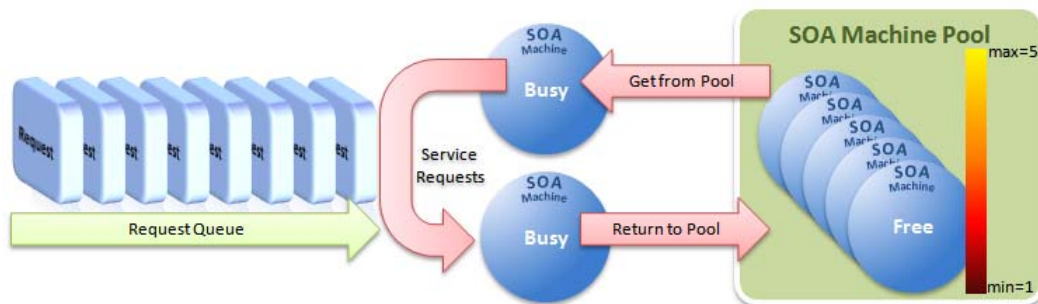


Figure 20. SOA Machines and Scalability

The design and runtime environments of the SOA Machine

The SOA Machines themselves are defined using XML Metadata. A metadata view of the SOA Machine is shown in Figure 21. Metadata (document) view of the SOA Machine and the SOA Machine Runtime Platform. This is the metadata version of the SOA Machine introduced in Figure 4. The SOA Machine specification is therefore platform-neutral and, based on its XML metadata specification, SOA Machines can be easily described by a graphical design studio.

The deployment process for SOA Machines is quite straight-forward, simply because the specification is a collection of XML files. This 'XML' Specification' can be deployed onto the runtime platform for execution. At runtime there is only one physical SOA Machine software component. But, each logical SOA Machine definition results in a separate instance, which itself can be replicated many times and pooled for scalability.

The SOA Machine uses a small Java footprint in its default deployment, requiring a JVM and Web Server only to execute.

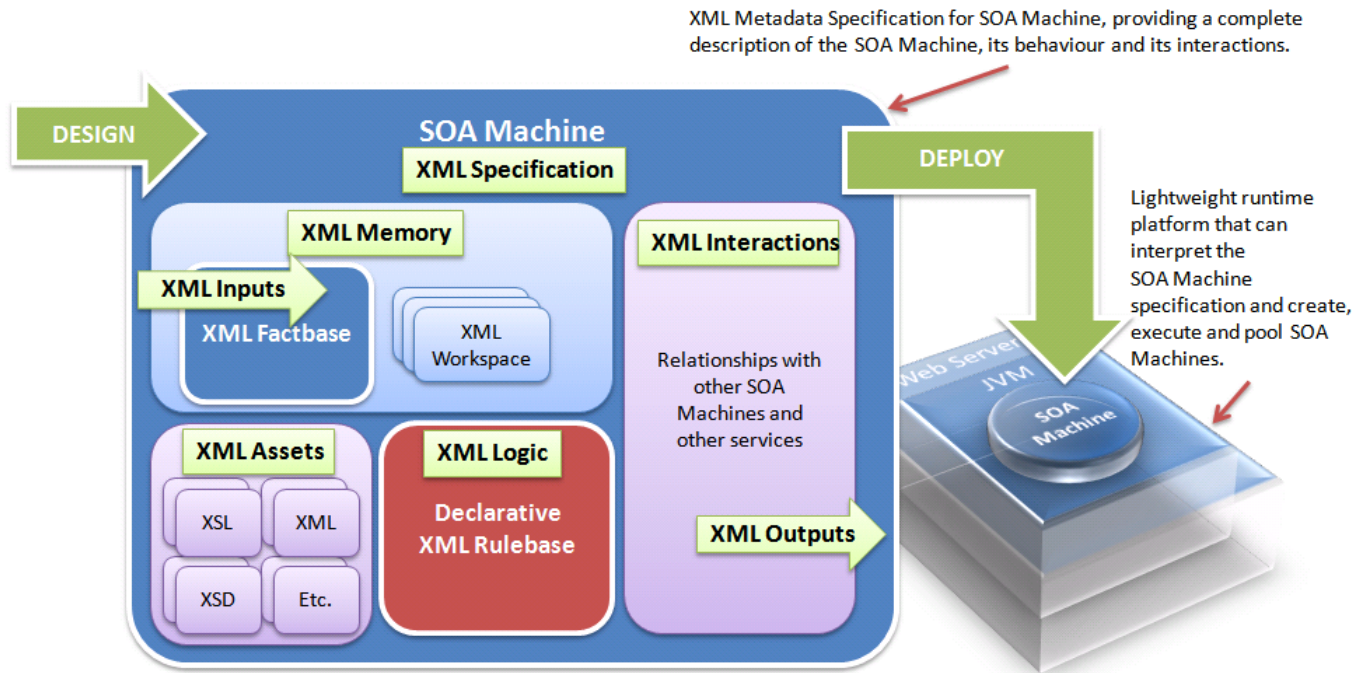


Figure 21. Metadata (document) view of the SOA Machine and the SOA Machine Runtime Platform

Conclusion

It is possible to define complete applications as a series of cooperating Service Components powered by the SOA Machine. Each component performing a different type of task, but processing XML inputs, assets and business rules in a consistent manner. The approach is simple but extremely powerful and allows business application builders to harness XML in a highly productive manner.

With the maturity of RIA for SOA applications and the increasing numbers of SOA services that are becoming available, a fresh look at the way we compose and develop SOA applications can lead to new and innovative methods of application creation and deployment. Do we really need some of the complex tooling and the heavy-duty, complex and costly architecture components associated with traditional enterprise applications? Are design tools and infrastructure components designed for integration and application hosting before XML and SOA were coined as terms, the best route to the design and development of modern SOA applications?

Using a new approach and technology that is designed-for-purpose and 'understands' and 'thinks' in XML can be used to produce applications that are:

- Easy and cost-effective to Design, Build and Deploy
- Easy and Cost-effective to Maintain in the future
- Multi-Purpose, Mobile, Portable, Flexible and Agile – just as SOA intended.

If you are interested in this topic you may also find the following papers useful:

"Web Application Development for the SOA Age – 'Thinking in XML'"

"Automating Rich Internet Application Development for SOA and Enterprise Web2.0"

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 and SOA applications by automating mundane developments tasks and reusing readily available information models. This is achieved through its '4GL RAD Style' declarative service composition technology that can be used to build Enterprise Web 2.0 applications without the need for 3GL programming skills.

Hyfinity has customers worldwide and is known for its innovation in rapid development tools and platform 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