XML Interoperability

Brian Loesgen

Principal Engineer, Stellcom Inc.

Introduction

From business-to-business supply chain management, through to providing web services, interoperability will play a key role in the next generation of Internet-enabled applications. Developers will be increasingly faced with the challenges of having their applications communicate with other systems. Distributed applications will move to the mainstream as developers produce applications that consume web services provided by third parties, and build web services for others to use.

This session will present current and emerging state-of-the-art XML-based solutions to the interoperability puzzle. Standards and initiatives such as SOAP, BizTalk and the BizTalk Server will be presented. Through a pragmatic approach, attendees will see how these XML technologies can be used to solve real-world integration and interchange challenges.

Brian Loesgen is a Principal Engineer at Stellcom Inc., a San Diego-based leader in wireless, e-commerce and Internet solutions. In that capacity he is involved in some of the most advanced web application development projects around. Brian is a co-author of the "Professional XML", "Professional ASP/XML" and "Professional Windows DNA 2000" books from Wrox. He has spoken at numerous technical conferences worldwide. He enjoys playing with bleeding edge software and translating new technology into real world benefits.
Contact Brian at bloesgen@stellcom.com

What's in a name?

When I started preparing this presentation, I started wondering about the *real* meaning of the term interoperability, so off I went and looked it up. Interestingly, all three of the definitions I found are subtly different.

m-w.com (Merriam Webster)	Ability of a system (e.g. a weapons system) to use the parts or equipment of another system
Webopaedia.com	The ability of software and hardware on different machines from different vendors to share data.
Dictionary.com	The ability of software and hardware on multiple machines from multiple vendors to communicate.

None of those really fit, although the last comes closest. My definition would be:

> **The ability of any entity (enterprise, device or application) to speak to, exchange data with, and be understood by any other entity.**

In this paper, we will look at the following three types of interoperability:

- ➢ Supporting multiple devices
- ➢ Enterprise:
    - – Inter-application
    - – Inter-process
    - – Inter-departmental
- ➢ Inter-enterprise (e.g. supply chain)

Developing for Multiple Devices

A Non-XML Based Approach

Conventional application development often means developing for multiple devices, and that form factors of the client devices can be dramatically different. If you base an application on a web browser display size of 800x600, it would never work on a device with a resolution of 4 lines by 20 characters. Conversely, if you took a lowest common denominator approach and sized for the smaller device, the user interface would be lost on an 800x600 device.

Using a non-XML approach, this leaves us writing multiple clients speaking to the server, or writing multiple clients speaking to multiple severs, resulting in something like the following diagram:
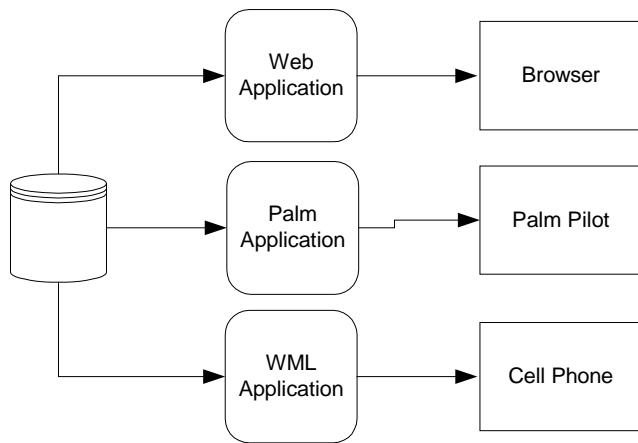
We have a serial path from each device back to the database. Adding a new device means adding a new application that produces the output required for that device.

Limitations of this approach include:

- ➢ Tightly coupled to browser
- ➢ Multiple code-bases
- ➢ Difficult to adapt to new devices
- ➢ Major development efforts for new devices
- ➢ Slow time to market

These are all significant problems, problems that lead directly to increased application complexity, development effort, QA effort and maintenance effort. Each step of the development and deployment process is repeated for each of the targeted devices, resulting in increased dollar and resource costs.

An XML Based Approach

The following depicts an alternative, XML-based approach to this problem:
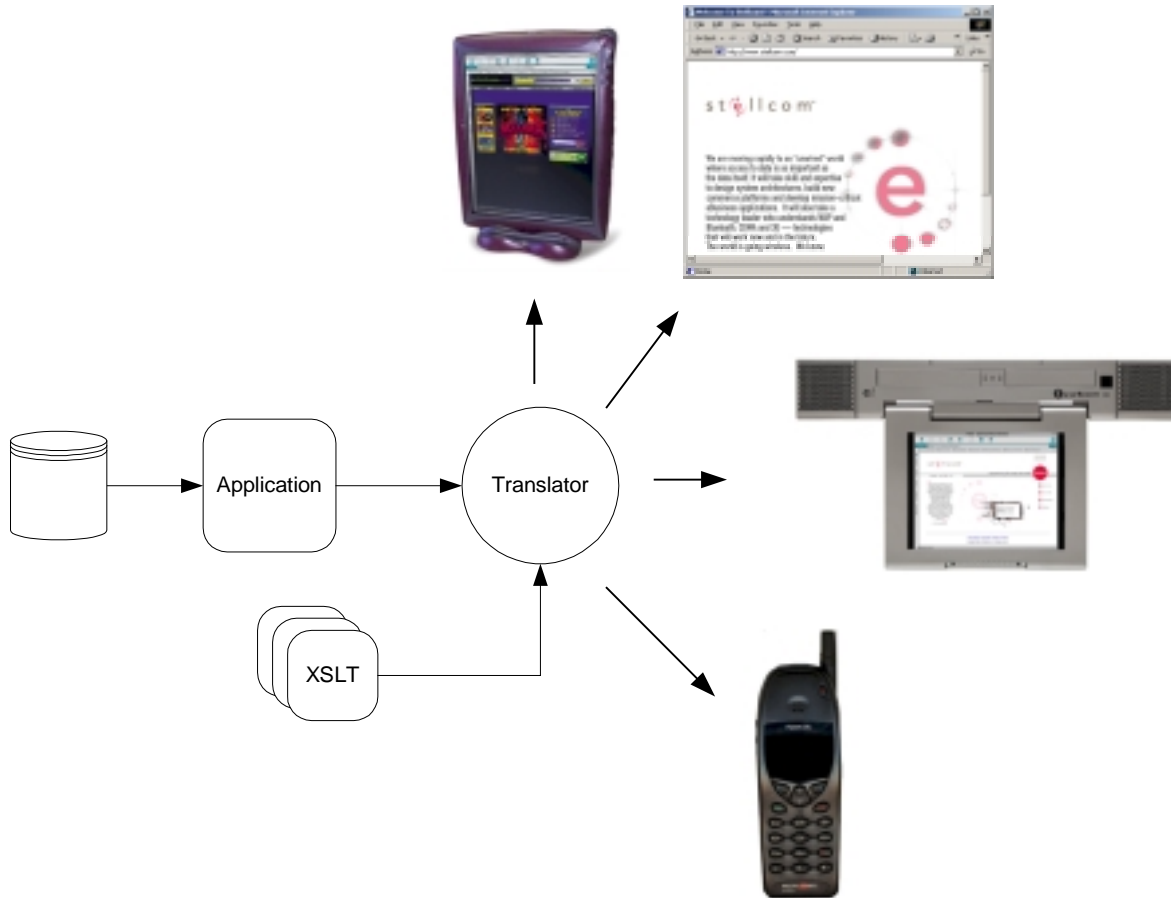
In this case, only one application exists. It runs against the back-end database, and produces an XML stream. A "translator" takes this XML stream, and applies an XSLT transformation to it. Every device could either use a generic XSLT, or have a specialized XSLT that would produce the required device-specific output. The transformation occurs on the server, meaning that no special client capabilities are required.

This "hub and spoke" architecture yields tremendous flexibility. When a new device appears, a new spoke can be added to accommodate it. The application itself does not need to be changed, only the translator needs to be informed about the existence of the new device, and which XSLT to use for it.

You're also ready for the web-enabled belt buckle (The *what*? I hear you cry!). In a few years time there is no doubt that we will be using devices dramatically different than those that have been conceived of to date. Not allowing for that probability means increased costs will be incurred later to accommodate them.

The conclusion to all this is that developing an architecture that ensures your e-business applications will be able to work well across a broad range of devices poses significant technical challenges. A properly architected system, should allow for extensibility to devices that have not been designed yet and may have radically different display characteristics.

In a properly designed system, XML technologies can be applied to address these challenges and protect your infrastructure investment and application.

SOAP

Every once in a while an idea comes along that on the surface appears very simple, yet has the power to affect the way applications are designed. SOAP (Simple Object Access Protocol) is such an idea. XML-based SOAP messages have the potential to transform the way we write distributed applications, and how we exchange data. At the time of this writing it was not a standard, but had been submitted to the W3C as a note. In all likelihood the W3C will establish a working group for XML protocols, with SOAP being the first one they examine.

SOAP is yet another mechanism that permits remote procedure calls, or remote method invocation - the same functionality as DCOM (Distributed COM), IIOP (Internet Inter-ORB Protocol) and others. In fact, SOAP isn't as full featured as some of those. As stated in the SOAP protocol, it does not attempt to support:

> ➢ Distributed garbage collection
> ➢ Bi-directional HTTP communications
> ➢ Boxcarring or pipelining of messages (batching multiple method calls into a single message as an optimization and to reduce network traffic)
> ➢ Objects by reference
> ➢ Activation (creating components and establishing connections to components)

SOAP is platform neutral, language neutral, and not dependent on any particular object model. Therefore, a SOAP-enabled distributed application could span multiple operating systems, consisting of objects from different vendors, written in different languages, and based on different object models. This is perhaps the long sought-after panacea of true component re-use.

Although earlier versions of SOAP were bound to HTTP, under the current protocol SOAP is not bound to any particular transport. You can send a SOAP message by SMTP or FTP. Although different transports can and will be used, it is likely that most SOAP message will indeed travel over HTTP.

Conceptually, you can think of SOAP as being the XML version of DCOM. The SOAP specification defines what a remote object method call should look like. The fact that when traveling over HTTP the method calls and data travel as plain text on the widely accepted and deployed port 80 means that SOAP-enabled distributed applications will be easier to deploy than DCOM-based distributed applications, can flow through firewalls and over SSL.

Conversation and Message Types

SOAP over HTTP enables distributed applications through two types of web communication scenarios: request/response and fire-and-forget:

> ➢ In the fire-and-forget one-way communication scenario, the originator invokes a method call in a remote object, but doesn't require a return value.
> ➢ In the request/response scenario, objects can have a bi-directional communication, with the sender invoking a method call and receiving a return value.

However, there is no real-time bi-directional communication; method calls and return values are passed back and forth through HTTP. A SOAP message will always fall into one, and only one, of the following categories:

> ➢ A method invocation, a Request
> ➢ The result of a method invocation, a Response
> ➢ A Fault

The method invocation originates on the SOAP client, and the SOAP server returns the result and fault. If a SOAP message is a fault, then, by definition, it cannot also contain a return value.

The Envelope Please

The SOAP protocol defines an envelope for a message, and a format for the XML payload container. The SOAP protocol includes a custom HTTP header, in addition to the XML of the message envelope and payload.

If a SOAP message travels over HTTP, then the protocol dictates that it include a SOAPAction HTTP header:

```
POST /myserver HTTP/1.1

Host: www.mydomain.com
Content-Type: text/xml
Content-length:nnnn

SOAPAction: my-name-space#myMethod
```

The reason for this is that it makes SOAP a routable protocol. A non-XML capable device, such as a firewall or router, could route the SOAP message without needing to understand the XML content.

The XML payload is a standard well-formed XML file. You could send binary data as part of the invocation or response, using Base64 encoding – the XML standard for encoded binaries. This would allow you to do things such as sending images with your SOAP request. Alternatively, rather than making binaries or resources part of the SOAP message, you can include a URI that points back at them.

The SOAP Header

The `<SOAP-ENV:Header>` element is optional. The intention behind the header is to send extended information along with the message. `<SOAP-ENV:Header>` elements are used to pass implicit information. For example, a purchase transaction may consist of several individual messages that are part of a transaction. A `<SOAP-ENV:Header>` could be used to tie the related the messages together. To enforce compatibility, elements contained inside a header can have a `mustUnderstand` attribute. If this attribute has a value of "1", it indicates to the receiving application that it must be able to understand and correctly process it. If the attribute has a value of "0", then that is functionally equivalent to it not being present.

Proper usage of this attribute will help in the creation of more robust applications that are extensible, but also able to raise errors if they receive messages containing data that is not fully understood.

If a SOAP server receives a message that includes a header element with a `mustUnderstand` value of "1", and it is not expecting it, then in order to comply with the SOAP protocol it should return a SOAP Fault (error message, see next section) with a SOAP faultcode of " MustUnderstand".

Who's Fault?

The SOAP protocol includes a way for the invoked method to return an error message to the requestor. This is achieved by having a `<SOAP:Fault>` tag as a child of the `<SOAP:Body>`. It is important to remember that there are only three types of SOAP messages: a request, a response, and an error condition. Any given SOAP message will fall into one and only one of those categories. Because of this, the only items that will ever appear as children of the `<SOAP:Body>` tag are details of the error.

The SOAP protocol dictates that a standard SOAP Fault message shall contain the following child elements:

Element	Contents
`<faultcode>`	One of the four values presented below.
`<faultstring>`	Human readable indication of error cause.
`<faultactor>`	Indicates who, or what process, was attempting to run when the fault occurred.
`<detail>`	If present, contains application-specific error information.

The Version 1.1 SOAP protocol defines fault codes as being in a "generic.refiner" model, the text to the left of the period is a generic fault message, the text to the right is more detailed (e.g. "Client.authentication"). The protocol defines the following generic fault codes: "VersionMismatch"," MustUnderstand", "Client" and "Server".

You may add additional sub-elements, provided they are namespace qualified. Any additional information you may wish to convey would, more than likely, be application-specific, and as such should be placed inside the `<detail>` element.

Serving It Up

The SOAP protocol defines what the message envelopes should consist of, and how error conditions should be communicated back to the requestor. Implementation details are completely up to the developer.

Whichever form the implementation takes, there will need to be some form of SOAP server that either acts upon, or acts as a routing agent for the SOAP message. This is the URI end point of the `POST`ed HTTP request.

Examples of this could include:

> ➢ An ASP that examines the message, instantiates a COM object and makes a method call
> ➢ A routing agent that looks at the HTTP header and routes the message on to another location
> ➢ An ASP that invokes a function within itself, using the XML payload as parameters

The above are only a few samples of SOAP end points. It is the fact that SOAP only defines a protocol, and that the entire implementation is left to the developer, that makes it so open and flexible.

Security

The good news is that you no longer have to pay homage to the Keepers of the Network, pleading with them to open a port. SOAP messages can be sent using the firewall-friendly and

generally open port 80. The bad news is that any methods you expose will now be accessible to anyone that knows how to call them, so security is a concern.

However, this is not as serious a problem as many have envisioned. SOAP messages can be sent through normal secure mechanisms such as SSL or HTTPS, and use normal access control and authentication processes. Firewalls and other HTTP filters can be configured to block messages based on the content of the HTTP header, thereby determining which methods can be called, and even by whom. The fact that the method name can be in the SOAPAction HTTP header (as well as in the body) means that messages can be blocked without needing to be able to understand the structure and content of the message itself. In order for someone outside your firewall to wreak havoc, you would first need to give them the capability by exposing the objects and methods to do so when invoked, give them access privileges, and even then they would need to know what to call. In other words, if careful attention is paid to security during the architecture phase, then problems shouldn't arise.

There is no inherent security breach created when you write a distributed application that uses SOAP messages. Someone with bad intentions would have to know where to go (URI to the endpoint), plus they would have to get through whatever security and authentication mechanisms are in place. Further, they would need to know what message format was expected. If you have a SOAP server that accepts orders, then worst case is someone (who knows the endpoint AND has passed through your security) would flood your system with false orders. Lastly, the developer of the SOAP server would have had to expose the pieces required to do damage. If the developer exposed a method that did a "format c:" then perhaps the design should be re-evaluated…

Good design practices coupled with an eye to security considerations will result in applications that can safely be exposed.

Encoding Data

The SOAP protocol defines a formal method for encoding data, and supports all simple and compound data types required by a modern application. It supports the passing or strings, integers, arrays (including multi-dimensional and partially transmitted arrays) and more. This formalization of the encoding of data is the key feature of SOAP that will enable cross-platform interoperability of objects constructed using different object models.

BizTalk

BizTalk is a multi-faceted initiative by Microsoft to facilitate XML-based data interchange. Microsoft's BizTalk initiative consists of the following parts:

> ➢ Schema Repository (BizTalk.org)
> ➢ Framework (the BizTalk specification)
> ➢ Enabling Tools (including the BizTalk server, schema mapper)

The BizTalk.org schema repository is intended to promote schema discovery, re-use and standardization. BizTalk.org is an independent body supported by numerous corporations including Microsoft, SAP, CommerceOne, Boeing, BP/Amoco and more.

The BizTalk framework is a specification that defines what the envelope of an XML message should look like. It provides a standard layout for information such as sender/recipient addresses and attachments, which can in turn be read and acted upon by BizTalk-aware tools and applications.

Lastly, there will be a suite of tools from Microsoft and other tool vendors that will be focused on facilitating the exchange of BizTalk messages.

The BizTalk Framework

At the time this was written, version 2.0 of the BizTalk framework had just become available in draft form. Prior to version 2.0, BizTalk and SOAP were two distinct definitions of an XML message. With version 2.0, this has changed, as BizTalk messages are now SOAP messages.

You can think of BizTalk as adding value to SOAP messages. BizTalk includes routing and delivery information beyond what is provided by SOAP. For example, with BizTalk, there is a mechanism defined for idempotent delivery (deliver once and only once, that is "if you get this order four times, only enter it once").

BizTalk messages can indicate they require a receipt, and the BizTalk framework defines what a receipt shall look like. A receipt is a SOAP-compliant message; it is not a BizTalk-compliant message.

BizTalk Products

At the time this was written, the following products from Microsoft had just been made public as part of the BizTalk Server 2000 beta:

BizTalk Editor	Schema editor.
BizTalk Mapper	Allows mapping of one type of document to another. Used to set forth translation rules.
BizTalk Application Designer	Workflow modeler. Allows a business analyst to graphically design a workflow, and a developer to define implementation, all within the same tool. The output of the tool is an XLANG (XML grammar describing a business process) document.
BizTalk Management Desk	This is where you define entities that BizTalk needs to know about, i.e. ports, channels, document definitions, etc.

Summary

Interoperability is a critical concern for developers. Applications as isolated island will soon become a thing of the past, as we move towards a world of loosely coupled applications capable of exchanging data freely and communicating with a broad range of devices.

In this session, we have:

- ➢ Defined Interoperability
- ➢ Seen XML Interoperability to talk to multiple devices
- ➢ Seen XML Interoperability using SOAP to implement a pub/sub message monitor
- ➢ Seen the BizTalk initiative and how they combine to facilitate XML interchange

Resources

There is a wealth of resources on the Internet on these topics, including:

SOAP:

- ➢ `MSDN.Microsoft.com`
- ➢ `Develop.com/soap`
- ➢ `www.soap-wrc.com`
- ➢ Your favorite XML site!

BizTalk:

- ➢ Framework: `www.microsoft.com/biztalk`
- ➢ Repository: `www.biztalk.org`
- ➢ Server: `www.microsoft.com/biztalkServer`
- ➢ Partners: `www.microsoft.com/biztalk/partners.htm`