NOTE-webbroker-19980511

WebBroker: Distributed Object Communication on the Web

W3C NOTE 11-May-1998

This Version:
http://www.w3.org/TR/1998/NOTE-webbroker-19980511
Latest Version:
http://www.w3.org/TR/1998/NOTE-webbroker
Previous Version:
None
Authors:
John Tigue <jtigue@datachannel.com> DataChannel
Jon Lavinder <lavinder@datachannel.com> DataChannel

Status of This Document

11 May 1998: This document is a submission to the W3C. Please see Acknowledged W3C Submissions regarding its disposition.

This document discusses XML based mechanisms for distributed object communication on the Web. It is not intended to be a final specification. Rather, this document is a report on a proof of concept implementation and is meant to motivate discussion within the W3C. Discovered design principles of a successful architecture are outlined. Many open issues are noted. This is a report on a work in progress.

This document is a NOTE made available by the W3 Consortium for discussion only. This indicates no endorsement of its content, nor that the Consortium has had any editorial control in its preparation, nor that the Consortium has, is, or will be allocating any resources to the issues addressed by the NOTE. The list of current W3C technical reports is available at `http://www.w3.org/TR/`.

Abstract

As of the time of this writing [1998/04] the necessary technological foundation exists to create a unified distributed computing model for the Web encompassing both document publishing and distributed software object communication. For lack of a better term, this model is referred to here as "WebComputing." Applications designed for the WebComputing environment exhibit a mix of features from both the Web publishing and the traditional distributed objects paradigms, blended into a unified model. The goal of this model is to extend the current Web application model such that the benefits of distributed object computing systems such as the OMG's CORBA and Microsoft's COM+ can be realized in a Web native fashion. The objective is to have a system which is less complicated than the

above mentioned distributed computing systems and which is more powerful than HTML forms and CGI.

Contents

---

1. Introduction

The Web is beginning to be used as a platform for a new generation of distributed applications. There is a growing need for the Web's architecture to adopt some of the features of traditional distributed computing systems such as the OMG's CORBA and Microsoft's COM+ while still maintaining the Web's current benefits. This document reports on research into integrating the two paradigms of Web publishing and distributed object computing into a unified model based on Web technologies.

Although the Web is nominally a document transfer system, it has long had mechanisms for distributed application communication. The functionality of HTML forms, HTTP POSTs, and server extension mechanisms such as CGI provide an "upstream" and flexible communication path from the client browser to the Web server. Taken as a whole these mechanisms amount to a simple and successful two-way client/server communication channel which has contributed to the successful adoption of Web technology. Web applications which use this channel for client to server communication are growing more complicated. Unfortunately, the current HTML form POSTing system lacks a foundation for application specific security, scalability, and object interoperability. The current Web application "architecture" is reaching design limitations. This document describes a design which extends the model to include some features of other distributed computing systems in order to address these and other issues.

Distributed object computing systems such as COM+ (and its ancestors OLE, OLE2, COM, DCOM et alia) and CORBA are the results of extensive development and research. The concept of inter-object message "brokering" plays a central role in these types of systems. Message brokering, often referred to as remote procedure calls (RPC), is at the heart of systems whose labels include "client/server", "n-tier applications" and "distributed object computing". The industry knowledge related to message brokering issues has evolved to where the current generation of systems have very similar feature sets. CORBA represents an effort to codify such technology in an open standard. The model presented in this document represents a similar effort.

The WebBroker system differs from CORBA in that the former is based on Web standards. In contrast, CORBA's Internet Inter-ORB Protocol (IIOP) is based on Internet standards. WebBroker is based on HTTP, XML, and URIs and so is termed "Web-native". WebBroker tries to blend and simplify the best features of COM+ and CORBA. The current WebBroker implementation represents the lowest common denominator of interoperability between these systems. Without a commonly agreed upon software component object model, Web applications will suffer from the same architecture incompatibilities which separate CORBA and COM+. Note that there seems to be a desire to unify the models and the politically neutral territory of HTTP/XML may be the best place to realize this goal.

The specific issues addressed in this document are concerned with both the communication between and the description of software application components on the Web. The term "on the Web" is intended to imply that only HTTP is used for message transport and that URIs are used to address the individual software objects, similar to the CGI model. The novel aspect of this proposal is the use of XML for two purposes: as the format of the serialized method messages between software component objects, referred to as marshalling, and as the format of documents which characterize the objects and the messages which can pass between them, the latter corresponding to type information residing in a CORBA interface repository or the Microsoft type libraries.

Whereas URIs are usually used to address (possibly dynamically generated) documents, the same mechanism can be used to address software component objects on a host which has an HTTP server. The argument can be made that this is simply an evolution of CGI and HTML forms. Note that CGI is simply an interface between a Web server and other processes within a host; it does not describe how the software objects within a host appear on the Web. The WebBroker typing documents are concerned only with the objects' network interfaces and do not constrain the internals of the hosts architecture.

This document presents XML 1.0 DTDs for the above outlined purposes and a proof of concept implementation. DataChannel is developing a test-bed code base for research and development in this area. Much like Jigsaw, the W3C's HTTP server, the software is written in Java and is freely available as source code. The WebBroker can actually be used with Jigsaw since the latter supports the Java servlets interface and the former is a servlet which embeds within an HTTP server. The code is available at DataChannel's WebBroker site. Although this implementation is written in Java for reasons of portability, the design of the WebBroker architecture makes no assumptions about programming language or platform.

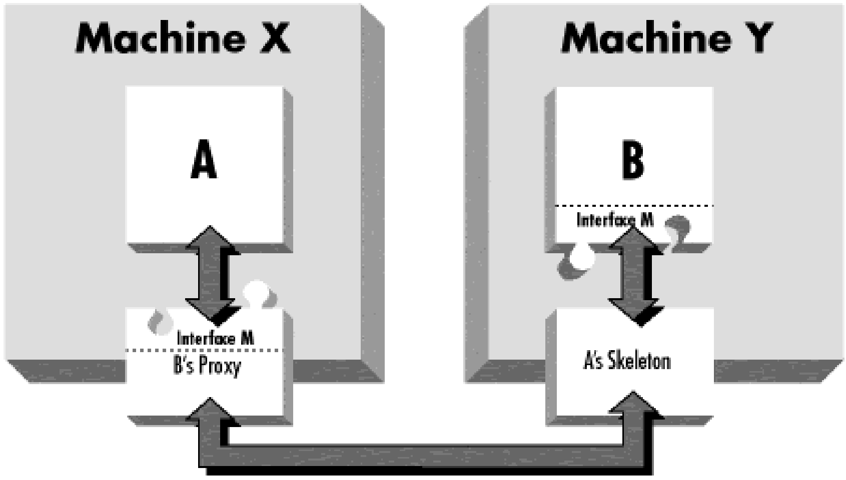
1.1 Definition of Distributed Object Communication in WebBroker

There are many styles of inter-component communication in distributed object systems. The style used by WebBroker is "interface" based. Interfaces are a concept common to COM+, CORBA, and Java. The concept of interfaces has been implemented since the early days of object oriented programming although more recent systems have refined the concept and represented it more explicitly in syntax. Note that although the serialized method calls are referred to as "messages" in this document, this does not relate to "message oriented middleware".

The goal of interface-based distributed object communication is to enable a software object on one machine to make method calls on a software object (potentially) located at another machine without the programmer or the objects having to directly deal with the fact that the communicating objects are possibly on separate machines. The core of this technique is that by typing the target object of the method call to an interface as opposed to a specific class, the method call can actually be made on an intermediary located on the same machine. This intermediary, called a proxy in WebBroker, implements the same interface as the real target of the message. The proxy is simply a helper which relays the message to the intended target. Symmetrically, there is a helper object on the destination host which receives the networked message intended for the target object. This object is termed the skeleton in WebBroker.

In general, the situation is that object A runs on machine X and needs to call a method on object B which runs on machine Y:

- Proxy (of object B) runs on machine X.
- Skeleton (of object A) runs on machine Y.
- The proxy and skeleton are essentially the ends of a virtual connection
- B’s proxy has the same surface (interface) as object B
- Proxy and skeleton do the work of moving the request (method call) and response (method return) over the network as XML documents



Interface-based communication coupled with an axiomatic set of intrinsic (or primitive) data-types and their representation on the wire, allows objects to communicate in a language independent fashion. Hence WebBroker needs a DTD to define the message contents and a DTD for describing the surfaces (or rather interfaces) on the objects. These two DTDs are the main deliverables of this work. Auxiliary DTDs are defined for such things as data encodings within an XML document and primitive data typing.

Performance does not seem to be fundamentally crippled by the nature of this new type of system. Xerox’s ILU effort has already combined an HTTP server with a broker (see [ILU]). The effort reported no performance problems. DataChannel has done a comparative study of various network transports and syntaxes and has found no fundimental problems with the HTTP/XML combination.

1.2 Benefits of the WebBroker System

Simply using XML and HTTP to recast traditional distributed object programming in a Web native fashion has value unto itself, but there are benefits beyond a simple syntactical and transport translation. This section explains some of these benefits.

From the developers perspective, WebBroker unifies the Web browsing and distributed object paradigms. Previous systems required an application to drop out of the Web paradigm if ORB-like functionality is needed (e.g Netscape 4 which shipped with an IIOP ORB or Microsoft IE4 and the DCOM protocol). By formulating a ORB-like mechanism in terms of Web standards the two models are unified and so ease the developer learning curve. This also reduces the amount of code. The transport mechanism is already provided by HTTP; no need for another wire protocol. Likewise, this reduces the number of parsers needed on the client. In general less code means less software errors, lighter clients, and greater interoperability.

Although some proxy and firewall administrators balk at the ideas proposed in this document, this recasting of distributed object communication into HTTP POSTs (or possibly another HTTP verb name, such as INVOKE, as discussed later) of XML documents actually provides a better foundation for secured firewalls than POSTed HTML forms (i.e. mime-type `application/x-www-form-urlencoded`). With POSTed forms, there is no way to constrain the contents of the message entity being POSTed. The design presented in this paper is the foundation for greater security as it enables the firewall to more precisely filter POSTed documents in an application specific manner.

One of the principles of this design is that if a client needs asynchronous notification then this should be accomplished via the HTTP protocol. This implies an HTTP daemon on the client. By unifying the Web client browser with a small (code of less than 2K in size has been realized) HTTP daemon, notification can be realized without undesirable timed polling, a bandwidth wasting technique which is beginning to appear more often. See [DocumationEast] for more detail.

Simply recasting distributed computing binary file formats to XML document types is beneficial. The extensible, hierarchical nature of XML increase interoperability. Consider such constructs as Microsoft’s type libraries (TypeLibs). These are privately formatted binary documents which are accessible through the Win32 TypeLib APIs. They are simply declarative "documents" which describe software components. By recasting these to XML documents (one of XML deliverables in progress) they are open and extensible. This way there is no need for a specialized TypeLib parser on light weight clients; the XML processor is sufficient. TypeLibs can be interlinked. This can be recast to linked XML documents.

Another candidate for syntactical translation is the Interface Definition Language (IDL). IDL documents look like procedural source code but are purely declarative documents void of procedural code. They are made to look like C code (with extra "attributes") because they are meant to represent interfaces to software objects. Interfaces are essentially collections of methods. Declarative documents on the Web are to be expressed in XML syntax.

So IDL files and TypeLibs can both be expressed as XML documents. Indeed they are one and the same. The CORBA Interface Repository and the COM+ type information in the Registry both become a collection of interlinked XML documents available on a Web server. No longer is there an unnecessary distinction between interface description and interface repository.

Recasting interface repositories as collections of documents is one example of where the WebBroker system is actually superior to other available systems on the Web. COM+ and CORBA were designed before the days of the Web. The Web has introduced high latency and cache concerns. The TypeLib APIs and the CORBA Interface Repository APIs provide very granular information about software components through sundry methods in the APIs. The WebBroker system recasts this information as XML documents. The benefit is that only one round trip to the Repository serving host is required to fetch a document compared to multiple calls in the case of the CORBA Interface Repository APIs. In general, the hierarchical nature of XML allows for a redesign of traditionally "chatty" services which do not adapt well to the Web environment.

Predicating the system on HTTP, URIs, and XML tightly constrains the solution set thereby increasing interoperability. Consider the current state of IIOP. IIOP is a specific implementation derived from the General Inter-ORB Protocol (GIOP) which is based on Internet standards (e.g. IP addresses and NDR). GIOP is abstract enough that it could be used to derive, say, "Web-IOP." (The current version of the WebBroker effort is the result of an attempt to find the lowest common denominator between COM+, CORBA, and Java so it does not precisely comply to GIOP. Any attempt to create a Web native inter-ORB protocol should be checked against GIOP.) The authors believe that Web-IOP would increase interoperability over IIOP because given Web-IOP's constraints of HTTP, XML, and URIs there are less permutations which satisfy the constraints than with IIOP.

1.3 Terminology

Two brief notes on the terminology used in this document. Microsoft's systems is referred to as COM+. COM+ is not currently deployed. DCOM and COM are currently deployed in Windows NT and Windows 95. COM+ is an evolution of COM/DCOM which is in flux as of this writing. It has been specified though. It has been designed to more closely model some of the features of CORBA and Java and so is more object oriented than COM/DCOM. WebBroker was designed with COM+ and not COM/DCOM in mind and so that is why the former term is used in this document. Because of the close similarities of COM+ and CORBA and Java, a cleaner abstraction can be derived. For example, COM+ introduces user defined exceptions to Microsoft's model. This is already in CORBA. Exceptions can be reported over the wire so this affects the WebBroker specification.

The other terminology point relates to the terms "proxy" and "skeleton". This represents a mix of terms from DCOM and CORBA. Both systems use the term "stub" but with conflicting definitions so it is not used here because of possible confusion. In WebBroker a proxy is analogous to a HTTP firewall proxy, it acts as a front for the real object. Proxies run on the client machine. Skeletons run on the server. DCOM uses the term stub where WebBroker uses skeleton.

2. Requirements

The following are among the design requirements:

- *Use Web standards*
  Currently deployed distributed object systems can already communicate over the Internet (DCOM and CORBA/IIOP being the two main commercially deployed systems). The goal of Web brokering is to broker messages between software component objects in a Web-native fashion. This implies HTTP as the transport protocol, XML as the syntax, and URIs as addresses.

- *Maintain programming language and platform neutrality*
  Systems such as Java's RMI which impose constraints on the programming language of implementation are unacceptable. Other current distributed object systems are language neutral. This is a requirement for technological adoption. Interoperability across programming languages is achieved by agreeing on a common set of primitive data types.

- *Integrate with current distributed object systems*
  For reasons of technological adoption, currently deployed distributed computing architectures such as Microsoft's COM+ and CORBA should be accommodated in the design. These two systems do very similar things and can already be bridged (see IONA). By designing for the needs of these widely deployed systems, the technology is more likely to be adopted. The ObjectMethodMessages DTD has been designed to be analogous to DCE RPC Protocol Data Units (PDUs). Note that a complete feature isomorphism is not realised in the current version of WebBroker.

- *Be object-oriented*
  Object oriented (or object based) computing has matured greatly in recent years and is now in mainstream commercial use. Any new distributed computing system should be object oriented from the ground up. This includes such concepts as user defined exceptions for reporting errors.

- *Keep the clients and protocol simple*
  Independent of scalability concerns, the protocol and client code should be simple. This encourages technological adoption as it did with early Web technology. This will also allow very low end machines to be able to participate in the system. This is viewed as one of the most important requirements.

- *Put the burden of intelligence on the servers*
  The Web is an erratic environment. For server robustness and scalability the design should impose the burden of intelligence on the server. This allows such "chatty" and complicated techniques as "object reference counting" and "pinging" to be excluded from client code and wire protocol. Pinging is undesirable on the Web. Object reference counting implies the representation of state in the protocol which is not natural to the Web. (Note that specific applications might still implement a pinging mechanism but such techniques must not be part of the core protocols.) A robust and scalable server cannot assume that object references will be correctly accounted on the Web; the server must be able to control resource allocation by itself. This implies a simpler wire protocol.

2.1 Transport

HTTP is the only transport used in the WebBroker system. Note that this work is distinct from efforts which simply put a Web server and an ORB (CORBA or COM+) on the same host. A Web brokering system as defined here uses no transport protocol other than HTTP.

Notification

In general distributed computing situations, the client may need to to "hear" notification of asynchronous events. The client must have a network listening mechanism, a "listener." Although the argument has be made that protocols such as SMTP can be used for purposes of notification, using client side HTTP listeners is a better design choice. Low end machines, such as PalmPilots, have constraints on memory, both persistent and volatile. Most WebComputing clients will have software to function as an HTTP client. This implies that HTTP protocol issues such as parsing and writing HTTP headers will need to be handled by the "talker" part of the client. The same header processing capabilities are required of an HTTP listener. Using a completely separate protocol, such as SMTP, for such things as notification would impose unnecessary stress on the code working set and persistent memory system. Therefore, using the same HTTP protocol to "talk" and "listen" is a better design choice in terms of reducing the number of software flaws, increasing interoperability, and lowering the bar for minimal clients. HTTP is also a protocol which many Web coders are already familiar with.

An HTTP INVOKE Method

Some firewall administrators are uncomfortable with the similarity between the concepts presented in this document and "HTTP tunnelling". The idea has been presented that a new HTTP method would be helpful. Although it is computationally impossible to completely block covert communication, there is some value in having a separate HTTP method (or verb depending on your terminology) for use in for distributed object communication. This is often referred to as the "INVOKE" method. For those who are concerned about covert communication over HTTP, a Web brokering system would actually be an improvement over the current HTTP form POSTing situation. By having applications use the HTTP INVOKE method and not HTTP POST, firewalls and proxies can route HTTP GETs and POSTs quickly while controllably scrutinizing distributed object communications which use the HTTP INVOKE method in a more computationally intense fashion i.e. filter the documents at the firewall. Note that even with an INVOKE method, the HTTP protocol still remains stateless. WebBroker attempts to define possibly stateful services running on top of HTTP. Currently high security sites need products which detect HTTP tunneling on POSTs. The XML DTDs proposed in this submission actually allow more control, not less because the markup tags explicitly delineate data structures in the XML documents. In previous systems, the structure of the bytes in the data packet was externally specified.

2.2 Addressing

The only addressing mechanism is URIs. No COM+ OXIDs, OIDs, IPIDs. The same holds for CORBA. Note that URIs can be "urn:" prefixed UUIDs, which are currently used in DCOM and CORBA. This will ease migration.

2.3 Syntax

Using XML for syntax further reduces the amount of novel code. Lightweight Web clients will likely have an XML processor. Besides not being Web native, adopting DCOM or CORBA syntax would simply increase the amount of code needed in light clients. By assuming XML, issues such as byte ordering and data formatting are already decided therefore increasing interoperability.

XML is also the foundation for data typing. XML-Data, WebSGML notations, and SGML data attributes are various mechanisms which have been worked out to express data types. Indeed one of the deliverables of this effort is a simple mechanism for data typing which is based on XML 1.0. XML 1.0 notation attributes are used for data encoding declaration. The same primitive data types as defined in the XML-Data Note are use by WebBroker. Hooks (namespace declaration and corresponding colonized attribute names) are provided for XML-Data compatibility. Full XML-Data is not actually used since as of this date there is no pubic implementation of an XML-Data processor. The WebBroker system is upward compatible with XML-Data.

Both CORBA and COM+ recognize the interoperability value of a small set of primitive, or intrinsic, data types. WebBroker uses a set which is common to COM+, CORBA, Java, and XML-Data. The only exception is URIs. URIs are the only addressing mechanism used in WebBroker. COM+ and CORBA predate URIs and use other constructs for addressing.

Some Design Choice Rationalizations

This section explains some of the design decisions which were made.

XML's "entity" facility is highly leveraged. Related type description structures such as `InterfaceDefs` and `ExceptionDefs` are referenced as entities. This allows for the instance syntax of the XML documents as therefore the XML application layer code to be abstracted from the document addressing mechanism. Entities can be declared using either a system ID or a public ID. System ID's are URLs. Public ID can be other "ID"s. Different implementations can be based on a simple file system or more complex mechanisms. For example, Java's code space is simply implemented on top of a file systems. This enables the quick adoption of legacy systems such as CORBA Interface Repositories or Microsoft TypeLibs.

The current version of WebBroker represents a very lowest common denominator in terms of features in existing systems and in terms of XML technology. There is no dependency on future or current work such as XLink, XPointer, or XML-Data except data typing.

3. Deliverables

There are two deliverables: the DTDs and a proof of concept implementation.

3.1 The DTDs

A layered approached has been taken in the design of these DTDs. The DTDs have been designed to each handle a separate aspect of the problem. The lowest DTD, `PrimativeDataTypeNotations`, declares primitive data type notations (in a fashion compatible with XML-Data but not dependent on an XML-Data processor). Another DTD, `AnonymousData`, defines how to data-type XML elements. A third, `ObjectMethodMessages`, defines document types which are the serialized method calls and returns between objects. Other DTDs are defined for describing software component interfaces. The latter part is an adaptation of work by McCool and Prescod (see [McCoolAndPrescod]).

The issue of XML verbosity is proving to be a non-issue. Nonetheless, for maximum terseness each element type name in each DTD can be reduced to a single Unicode character. This has proven to be of little performance benefit. Any performance benefit is arguable overshadowed by the corresponding lack of comprehension on the part of humans reading such documents. DataChannel has produced these terse variations for completeness, but does not suggest their use at this time.

ObjectMethodMessages

The goal of these declarations is to express in XML documents the serialized method calls and returns between software component objects. The higher level issue of interface definitions is outside the scope of these declarations. See the InterfaceDef DTD for such issues.

This version is not final; it will need to be modified in order to reflect details of COM+ as it is further refined. See the WebBroker site ([WebBrokerSite]) for the latest information.

These XML definitions are designed to define two types of XML documents: one for serialized object method calls and the other for serialized object method returns which are collectively termed "object method messages." Documents complying to this DTD are expected to declare the document element type as either `objectMethodRequest` or `objectMethodResponse`.

For example:

```
<!DOCTYPE objectMethodRequest
          PUBLIC
          "-//DataChannel//DTD ObjectMethodMessages V1.0//EN"
          "http://xml.datachannel.com/ObjectMethodMessages.dtd"
          >
```

Another example:

```
<!DOCTYPE objectMethodResponse
          SYSTEM
          "http://xml.datachannel.com/ObjectMethodMessages.dtd"
          >
```

Note that for network and parse efficiency, all the following element and attribute names can be mapped to single character tokens. This is not done here for the sake of human readability. The terse analog of this DTD is available at [TerseAnonymousData]

There is no facility for "structs" or "complex" or "composite" structures. Complex data types (corresponding to, for example, C struct) are not addressed in this level of the "stack". These declarations address only how to serialized data into an XML document. The construction of complex data types from a sequence of primitives is the responsibility of the marshalling code. Such issues are addressed in the InterfaceDef DTD. This DTD deals only with serialization of data. It is assumed that the proxies and skeletons on both ends of the connection know how to marshal and unmarshal the primitive data types into (possibly) higher level constructs. This is analogous to DCE RPC where Request and Response PDUs do not have named data structures.

Note that in the ObjectRef element there is no explicit support for "reference counting." Reference counting does not scale well on the Web and complicates the client and protocol. The Microsoft Transaction Server (MTS) employs a more scalable model where "state" is separated from implementation, allowing an object to have control over its own "liveness" without being controlled by reference counts in other processes. Not only does the lack of reference counting make the server more scalable, it also keeps the clients and protocol simple.

A later version of this DTD may have the ability to name the data being marshaled. This would be useful for situations where parameters are optional and so need to be named to be correctly identified. For this version, correctly sequencing data into a document is the responsibility of the proxy and skeleton.

Correctly behaving proxies and skeletons can be algorithmically generated from the InterfaceDef documents.

AnonymousData

The AnonymousData DTD defines structure for assigning data tying attributes to character data in element content, a simple way to represent data types in XML 1.0 documents. AnonymousData is used as a helper DTD for the ObjectMethodMesssages DTD and for the interface typing DTDs (InterfaceDef et alia).

The data-types defined in XML-Data (see [XMLData]) are used. XML-Data's schema facilities are not used. Instead, the XML 1.0 DTD facilities are used as much as possible. Data typing is not achievable in straight XML 1.0.

For maximum flexibility, AnonymousData allows any sequence of data-types to be expressed. It is the responsibility of higher level DTDs to constrain these definitions to a particular sequence of data typed elements. This DTD has only primitive data-types and arrays, no complex data-types.

This DTD is named AnonymousData because the element type names are the same as the data type i.e. the data is anonymous, it is not "named." CORBA/IIOP and COM+ do not name data as it goes across the wire. Rather their data marshallers simply "know" where data element boundaries are supposed to occur in the byte stream. Or rather the knowledge of data type sequencing is in the method definitions. Denoting the method ID in the serialized method message is sufficient to allow a lookup on the interface definition for the proper method which defines the proper data typed information sequencing. The parallel with CORBA and COM+ is maintained in order to maximize possible interaction.

XML is character based and structurally self-describing. The boundaries of the data elements can be represented as XML open and close tags. This is useful for variable sized data. This design also allows for quick adoption of existing systems (e.g. CORBA and DCOM). A simple XML 1.0 processor can be used with a small amount of code in the application layer to perform data typing. Of course, integrating the processor and data typer is more efficient, but at least this way the system is easier to reproduce with current standard technologies.

The AnonymousData DTD has been designed to handle the most common data-types which are expressed in the most widely deployed distributed object systems. The following table maps data-type schemas between these various systems. The first two columns are taken from [ComToJavaMap]. The CORBA mapping is taken from [ComToCorbaMap]. The other columns are added for clarity.

COM+ Type (IDL)	CORBA Type	Java Type	WebBroker Type	XML-Data Type	Interpretation
boolean	boolean	boolean	boolean	boolean	true or false
short	short	short	short	i2	Signed 16-bit integer
int (long)	long	int	int	int (or i4)	Signed 32-bit integer
?	?	long	long	i8 (alias = long)	Signed 64-bit integer
char	char	char	char	char	LCD is ISO Latin-1 (not acceptable)
float	float	float	float	float.IEEE.754.32	IEEE single precision floating point number
double	double	double	double	float.IEEE.754.64	IEEE double precision floating point number
byte	octet	byte	byte	bin.hex	8-bits, opaque
BSTR	string	java.lang.String	string	string	sequence of ASCII chars (unacceptable)
N/A	N/A	java.net.URL	URI	URI	an address or name

The following Ole Automation types have not yet been implemented: CY, DATE, VARIANT, SAFEARRAY. They might eventually, but were not necessary during the proof of concept development.

The following Ole Automation types are not be expressed: IDispatch, IUnknown. Rather they are expressed as COM+ object references, not DCOM structures. DCOM structures can be expressed in a DTD on a higher level than this low level primitive data-typing DTD.

Some types may be null. Null is indicated by an empty element of the appropriate type e.g.:

```
<string />
```

Arrays appear as follows:

```
<intArray length="2"><int>432908</int><int>0</int></intArray>
```

OR

```
<intArray />
```

OR

```
<intArray></intArray>
```

The first is a normal `int` array. The second is null occurring in the place of an `intArray`. The third is an `intArray` of length zero. As per normal XML 1.0 markup minimazation, the dt:dt attribute is not explicitly included because it has a default value declared in the DTD. It was decide to not have an

ENTREF attribute which could be used to signify &null;.

There are some data types which cannot evaluate to null:

- A boolean is always an empty element which always has a non-defaulted value attribute. booleans are never null after interpretation. They may appear in the form:
```
<boolean value="true" />
```
or
```
<boolean value="true"></boolean>
```
- A char can never be null, so an empty element means the it should be interpreted as the character whose code is zero.
- Numeric whole numbers or floating point numbers can never be null. An empty element indicates the it should be interpreted as 0 or 0.0 (as appropriate).

The length attribute is currently required on all array element types. As an aside, this boils down to the same issue as chunked streams and the HTTP Content-Length header. Declaring the length is nice (easier parser memory allocation on read side because length is known at start of array read), but mandatorily having to calculate it can be expensive in terms of memory for small machines; the entire array needs to be held in memory to determine the value which needs to be assigned to the length attribute in the open tag. Perhaps this could be made optional but strongly recommended.

Having an explicit length attribute for strings may seem unnatural to anyone experienced with SGML but it helps (dumb) data marshallers, and reduces the amount of code which needs to be written in order to Web enable legacy code.

For network and parse efficiency, all the data typing element and attribute names can be mapped to single character name tokens. For the sake of readability this is not done in AnonymousData but the technique is explicitly mapped out in TerseAnonymousData.

InterfaceDef et alia

The InterfaceDef DTD (and its related DTDs ModuleDef, ExceptionDef, and TypeDef) is used to define software component interfaces and the messages which can pass between them. This functionality is less well defined than ObjectMethodMessages. InterfaceDef documents correspond to both IDL files and interface repository information in other systems. InterfaceDef documents can be used to generate proxy and skeleton implementation code.

A design goal of these related XML document types is to correspond to existing interface definitions and the APIs for accessing those definitions (e.g. COM TypeLibs accessed through APIs such as ITypeLib, ITypeInfo, and ITypeComp and also the CORBA Interface Repositories and its access APIs). The novelty and value in the WebBroker system is the use of XML for syntax and the Web conscious design. Because the Web can have long, slow connections between hosts, it is desirable to minimize the number of client to server round trips. Traditionally in LAN-based distributed computing it is acceptable (even encouraged) to have very high granularity APIs which fetch small pieces of information from the server. On the Web it is desirable to pack as much related information as possible in a single round trip. This must be balanced against a modular design. The balance point is also constrained by the nature of XML mechanics.

InterfaceDef documents can be compared to Microsoft TypeLibs. A TypeLib is a binary document which corresponds to an XLink group link (and embedding of) sundry related type definition documents, similar to how an HTML page may have links to its graphics and other related sub parts. In the WebBroker system, a client machine will download a desired "TypeLib" root document, keep the connection open, read the "TypeLib" to discover what other documents need to be downloaded, and after doing so it will then create the in-memory representation of the "TypeLib". (Technically that amounts to multiple round trips but an HTTP persistent connection makes that inexpensive). Perhaps Microsoft could even expose these documents through the existing Win32 TypeLib APIs.

Another benefit of this design is that by packing up the type information into a collection of XML documents, only the relevant parts need be downloaded, resulting in efficient federating as opposed to bringing down large type repositories. In this way the interface repository (or NT Registry type informations) is (controllably) exposed to the Web. This Web conscious design demonstrates the value of XML as the syntax of the Web's APIs.

The concept of `Exception`s has been assumed for reporting faults even for the COM environment. Microsoft is heading in that direction so there is no need to propagate non-object oriented HRESULTs. The Microsoft Java Virtual Machine already expresses HRESULT error codes as instances of the class com.ms.com.COMException. The COMException is what is transmitted over the Web by WebBroker.

One of the design goals of this effort is to enable a simple client and a simple protocol. For client-side simplicity, one design option is to allow method parameters to only be marshaled `in`. The same information to be fetched via `out` marshalled paramaters can be achieved by defining a struct which contains all the desired information and using that as the return type. The only information marshaled out would be the return value (or an exception). This way the client-side proxy does not have to map data structures which are marshaled both `in` and `out`.

The mapping between CORBA's meta object facility, WebBroker, and COM+:

CORBA	WebBroker	COM+ (TypeLibs)
ModuleDef interface	ModuleDef document	ITypeLib
InterfaceDef interface	LCDInterfaceDef document	ITypeInfo
ExceptionDef interface	ExceptionDef document	TBD
OperationDef interface	methodDef element	FUNCDESC
ParameterDef interface	parameterDef element	ELEMENTDESC
AttributeDef interface	not implemented	VARDESC
ConstantDef interface	constantDef element	VARDESC
TypeDef interface	ComplexDef document	ITypeInfo
Repository	Root URI of Type information documents	NT Registry Key HKEY_CLASSES?

`ParameterDef`s do not occur as independent document roots. Rather they occur as elements which are contained in an `interfaceDef`. Applying the "lowest common denominator" design goal means that there are no such things as `AttributeDef`s because Java does not support them well.

In general, many element names defined in these documents are of the form "XxxxDef" and "XxxxRef" where XxxxDef is the definition of some structure and XxxxRef is a reference to a XxxxDef element with the same name prefix: e.g. ExceptionDef and ExceptionRef.

In a situation such as DCOM where the included defs are actually in a legacy format such as typeLibs, you might want to use a notation declaration for transitional purposes, say:
<!NOTATION MSTYPELIB PUBLIC "Microsoft TypeLib v2" >
This has not been done here. TypeLibs, or some subset of their information, should be algorithmically migratable to the WebBroker system. See WG4_N1958 for a precedent of this style of identification.

Note that the XML namespace facility is not relevant to this issue. XML namespaces and software code module namespaces are completely unrelated and one can not be used to express the other.

3.2 Proof of Concept Code

A proof of concept was implemented as a Java servlet. Servlets are currently deployable in many Web servers. HTTP POST was used since there is no INVOKE method in HTTP 1.1. Note that many servlet engines are flaky. JRun is the best implementation that DataChannel has experimented with. It is freely downloadable from Live Software Inc.

4. Open Issues

The following have been identified as open issues:

- WebBroker needs to be contexted into the currently expressed W3C goals. Perhaps such an effort does not belong in the W3C
- Need to synchronize the DTDs and proof of concept code
- Better tools such as proxy and skeleton generators are sorely needed
- More research needs to go into interoperability between DCOM, CORBA, Java
- Scalability needs to be tested as thoroughly as Xerox ILU's

Standards Effort and Deliverables

This section outlines identified potential deliverables of any standards effort.

Recent efforts have separated specification of architecture from specification of implementation details (binary API interfaces). For example, in the W3C there is the XML Working Group (architecture) and the DOM Working Group (API). In CORBA, there is the IIOP spec (architecture) for how the protocol looks on the wire and there is the binary interface (API) specification which defines how an ORB and an application relate to each other within a machine. The same patterns could be used for a WebBroker effort. An effort could be made to specify the network protocol and another effort to specify the binary interface between the broker and an application.

For example, consider the issue of client side callbacks. The protocol architecture spec should say something like "each object on a host is required to have a unique ID scoped to the host address." The implementation spec should say something like "For simple clients, the interface IUuidGenerator is specified (with the following IDL interface ...and the following semantics ...and the following name scope for locating the service on the system...) to be used by client objects to generate unique IDs. For more sophisticated clients the following interface ISubURITreeNameNegotiator is specified (...various details...) for use in negotiating sub URL trees which the interface implementor shall guarantee will prevent host-wide URL namespace collisions. This interface enables the client application to negotiate the URL subtree it is to be notified on.". In general this specification would be concerned with defining the software environment in Web browsers and Web servers.

The first deliverable of a Working Group is a process document defining the context of the effort. Goals and non-goals need to be specified. For example, full interoperability of existing COM+ and CORBA components would probably not be a goal. Yet a Web broker system should allow a disciplined programmer to use a Web brokering system to communicate between a COM+ server and a CORBA server using a lowest common denominator of functionality.

Issues such as implementability and dependencies would need to be defined. For example, research has shown that the only hard requirement not yet satisfied by a W3C standard or standard endorsed by the W3C is primitive data typing. This is the only as yet identifies unsatisfied dependency. Desired and unacceptable features would need to be surveyed.

The first non-process oriented deliverable should be a level 0 specification analogous to the work in the DOM Working Group. Indeed one requirement of the effort should probably be that client side software should be implementable on top of the DOM (even if that would not be the most efficient possible implementation). Such a document would define the lowest common denominator architecture and services necessary for implementation. Level 0 should have no optional features. This should define the minimal foundation for a Web native distributed object communication architecture. A target environment would be defined along with the interfaces to the required services.

5. References

DocumationEast
: (DataChannel) *XML Enabled Mechanisms for Distributed Computing on the Web*, au. John Tigue. See http://www.datachannel.com/developers/webbroker/DocumationEast.html.

WebBroker
: (DataChannel) *XML for Distributed Computing: WebBroker*, au. John Tigue. See http://www.datachannel.com/developers/webbroker/XMLSeattle/index.html.

WebBrokerSite
: (DataChannel) *The WebBroker site*, au. John Tigue. See http://www.datachannel.com/developers/webbroker/.

Shibui
: (DataChannel) *Shibui: links to pages related to Web native distributed computing*, au. John Tigue. See http://www.datachannel.com/developers/webbroker/shibui.cdf.

ExLangs
: (W3C) *Web Architecture: Extensible Languages*, au. Tim Berners-Lee and Dan Connolly. See http://www.w3.org/TR/NOTE-webarch-extlang.

ComToCorbaMap
: (OMG) *Mapping: COM and CORBA*, au. OMG. See ftp://www.omg.org/pub/docs/formal/98-02-21.pdf.

GIOP
: (OMG) *General Inter-ORB Protocol*, au. OMG. See ftp://www.omg.org/pub/docs/formal/98-02-218.pdf.

McCoolAndPrescod
(University of Waterloo) *Software Component Interface Description in SGML*, au. Michael McCool and Paul Prescod. See http://www.cgl.uwaterloo.ca/Projects/meta/sgml97/mmccool/index.html.

ComToJavaMap
(Microsoft) *Type Mappings Between Java and COM*, au. Microsoft. See http://www.microsoft.com/java/sdk/20/jnative/type_mappings_between_java_and_com.htm.

XMLData
(Microsoft et alia) *XML-Data*, au. Andrew Layman et alia. See http://www.w3.org/TR/1998/NOTE-XML-data-0105.

WG4_N1958
(ISO JTC1/WG4) *Information Processing -- Text and office systems -- Using SGML Public Identifiers for Specifying Data Notations*, au. M. Bryan. See http://www.ornl.gov/sgml/wg8/document/1958.htm.

TerseAnonymousData
(DataChannel) *TerseAnonymousData*, au. John Tigue. See http://www.w3.org/TR/1998/NOTE-webbroker-19980511/TerseAnonymousData.dtd.

ILU
(Digital Creations) *The ILU Requester: Object Services in HTTP Servers* , au. Paul Everitt. See http://www.w3.org/TR/WD-ilu-requestor-960307.