Open Metadata Formats:
Efficient XML-Based Communication for High Performance Computing

Patrick Widener   Greg Eisenhauer   Karsten Schwan

College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280
{pmw,eisen,schwan}@cc.gatech.edu

Abstract

High-performance computing faces considerable change as the Internet and the Grid mature. Applications that once were tightly-coupled and monolithic are now decentralized, with collaborating components spread across diverse computational elements. Such distributed systems most commonly communicate through the exchange of structured data. Definition and translation of metadata is incorporated in all systems that exchange structured data. We observe that the manipulation of this metadata can be decomposed into three separate steps: discovery, binding of program objects to the metadata, and marshaling of data to and from wire formats. We have designed a method of representing message formats in XML, using datatypes available in the XML Schema specification. We have implemented a tool, XMIT, that uses such metadata and exploits this decomposition in order to provide flexible run-time metadata definition facilities for an efficient binary communication mechanism. We also demonstrate that the use of XMIT makes possible such flexibility at little performance cost.

1. Introduction

Wide area applications have been a strong focus of research in high performance computing. This has resulted in the development of software infrastructures like PVM, MPI, and Globus, and it has led to concepts like Grid Computing or the National Machine Room made concrete by ongoing design and implementation efforts at DOE Laboratories, by NCSA/Alliance researchers, and by the broader research community via vehicles like the Grid Forum. Increasingly, research focus in this domain has turned towards component architectures [11] which facilitate the development of complex applications by allowing the creation of generic reusable components and by easing independent component development. Some of the earliest requirements for component architectures in high-performance computing were derived from systems that attach scientific visualizations to running computations, but continuing research has generalized such models to include the ability to also flexibly link remote instruments and even general purpose computational elements[19, 20, 11]. This work has taken advantage of technologies for component-based software development, including systems like Enterprise Java Beans, Microsoft's Component Object Model and its distributed extension (DCOM), and the developing specification of the CORBA Component Model (CCM) in OMG's CORBA version 3.0.

Today's component-based applications range from remote sensors and instruments, to scientific visualization, to multimedia and distributed immersive systems, to collaborative design and manufacturing. Such applications exchange structured data that describes parts or equipment under design, graphical objects being displayed, or scientific data representing atmospheric volumes and chemical concentrations. An important property of mechanisms for exchange of structured data is the degree to which they permit the independent evolution of data definitions, thereby enabling applications to communicate via enhanced data structures. This is difficult when using communication mechanisms like RPC or CORBA's remote object invocations that perform data definition in a programmatic fashion. Embedding metadata into communication or application code may result in good performance, but substantial costs arise when applications evolve, as changes in metadata require consequent modification and recompilation of the codes using such metadata. In addition, this approach limits the usefulness of metadata; while most systems can be usefully abstracted by the structure of the data they exchange, the level of expressiveness of commonly used programming languages such as C is a severe handicap. Finally, embedding metadata also 'hides' it from just the nonprogrammer end users to whom it is typically most useful: the engineers designing parts, the scientists studying atmospheric phenomena, and others, who share data in their distributed collaborative workspaces.

Our work is predicated on the belief that open metadata systems will become increasingly important and useful, especially for non-programmers using distributed computations that share substantial amounts of data. This belief is validated in part by the increasing popularity of metadata standards like XML [4]. However, the success of open metadata systems requires that their use does not unreasonably degrade the performance of applications that use them. Consequently, our work seeks to reconcile openness and performance.

Efficient Binary Transmission of Structured Data. Our work addresses interoperability at levels 'below' those of RPC or CORBA, but with functionality exceeding that of common data exchange formats like XDR. Specifically, we are concerned with the efficient movement of the data structures that are defined at the 'system' level of distributed applications, middleware, and systems, typically using system implementation languages like C. Such structured data usually resides in main memory, and when it is moved across heterogeneous machines, one must deal with issues like byte-order, field alignment, and atomic type representation. Furthermore, at this level, data transmission in binary format is critical, due to the high communication bandwidths or low transmission latencies required, or because of the undue processing loads that would be imposed on systems if they were forced to transform information from end user readable formats, like text, to binary formats, for instance. Sample applications requiring binary data transmission include high performance codes moving scientific or engineering data and wide-area transfers of operational data, where scalability to many information clients and sources implies the need to reduce per-client or per-source processing and transmission requirements. They also include server-based applications in which single servers must provide information to large numbers of clients.

Efficient and 'Open' Specification of Data Structure. In CORBA, the specification of data structure is achieved with IDL specifications. In RPC, procedure parameters are characterized by their types specified within 'interface module' descriptions. Openness in metadata definition implies that data structure specifications are not linked to certain transmission mechanisms, such as RPC, or linked to specific protocols, such as those used in the transmission of manufacturing, parts, or design information in the automobile industry, or as those used by specific data storage facilities (e.g., database query languages). Instead, openness requires that data structure may be specified independently of data transmission and use, with translations of such structure to the efficient lower-level representations used for data transmission or manipulation 'hidden' from end users.

We have developed a novel approach to high performance, open metadata systems. This approach decomposes the transfer process for structured data, which results in efficient, binary (not text-based) low-level encodings of data, while also maintaining open user-readable and -comprehensible data structure definitions. Our approach is implemented by XMIT (XML Metadata Integration Toolkit), a tool which provides flexible metadata definition using XML, while also supporting high-performance, binary data transmission. We demonstrate in this paper that XMIT provides performance comparable to that of binary data transmission, by using runtime methods of establishing structured data exchange. Small 'startup' overheads are incurred only during 'connection establishment', that is, each time an XMIT-based exchange is initiated and/or the structure of the data exchanged is modified.

To validate our claims, XMIT is applied to one of the earlier 'portal' demonstrations developed by NCSA researchers, a component-based visualization system for hydrology data[18]. Beyond the performance results demonstrated in this paper, our future work expects to derive benefits from XMIT both in the area of runtime type extension (e.g., when less capable visualization engines such as handhelds can customize remote metadata for their own needs) and to create systems that are more amenable to evolution (e.g., to explain new visualization capabilities through the use of remotely-defined metadata).

The remainder of this paper is structured as follows. We discuss how metadata is used and our insights into the process in Section 2. Section 3 describes how the XMIT toolkit works. Our evaluation of XMIT is presented in Section 4. Finally, we discuss related work in Section 5 and conclude in Section 6.

2. Usage of Metadata

The usage of metadata may be decomposed into three components, thereby separating the three actions necessary for the transmission of structured data: (1) discovery, (2) binding, and (3) marshaling.

Discovery. Any binary communications mechanism (BCM) must *discover* the metadata it will use for transmission of a given message. This process can take several forms. Monolithic program construction embeds metadata in communication code. In such cases, metadata is directly available and the discovery process is obviated. Modular programming practice has led to the abstraction of BCMs into library code, and consequently to the decoupling of metadata from BCM processing. This decoupling forces the discovery process to become explicit. Several alternatives have been devised to address metadata discovery. In IIOP, for example, metadata is provided in a structured manner defined by the BCM, and made available to the BCM library as part of program execution. Systems using this approach benefit from being able to make architectural decisions at compile time, which allows direct examination of the behavior of the compiler and examination of the host architecture. However, since the metadata is compiled into the system, it is not readily modifiable (although several systems have taken pains to increase the robustness of such metadata at runtime). Other approaches rely on interface definition languages (IDLs), which enable the expression of architecture-independent message definitions. An IDL compiler for each participating host platform translates these definitions into architecture-dependent definitions, and code generators provide library routines to translate messages at run-time[2].

Binding. Association of a certain metadata format with a particular unit of program data is known as *binding*. In compiled-metadata systems, a data unit is associated with a metadata format via some API call. The programmer is responsible for properly using this call to ensure the compatibility of the program data and the metadata. In contrast, IDL-based systems generate format-specific subroutines which serve the same purpose. Binding usually results in the construction of some type of message format descriptor or token to be used during marshaling.

Marshaling / Transmission / Unmarshaling. The purpose of BCMs is to translate messages from in-memory formats to and from a wire format. All BCMs define 'wire formats' that define how any data will be transmitted over the communications transport layer. A message is provided to a BCM as a region in the address space of a process whose contents must be translated to the wire format. The region is described as a record containing a set of fields. Metadata describes, for each of these fields, its size, type, and layout within the record. This information is used by the BCM to generate wire format records that contain sufficient information for the receiver to properly reconstruct the message. Marshaling is not always a one-pass process; some BCMs are designed to convert metadata into an internal representation and only later use the internal representation to generate wire format transport-layer messages.

Orthogonality. It should be clear that how metadata is provided to a BCM does not in any way influence how that metadata is used for binding or marshaling. Where a BCM uses an internal representation of metadata as described above, for example, it is possible to construct a system that can switch from compiled-in metadata to that provided by a directory server, with each set of metadata being converted to internal representation. In such a system, the method of metadata discovery changes, but the method of metadata binding does not. Conversely, a BCM that relies on IDL descriptions of the messages it sends can make different binding decisions (choosing from among versions of message formats, for example) without affecting the method of metadata discovery.

Discussion. First, changes to the discovery process are not likely to result in performance gains. The number of message format changes is small compared to the number of messages sent by typical systems, and discovery of metadata is generally performed either only at program startup or else infrequently during program execution. Second, changes in binding will result in performance gains only with respect to the actual binding decisions being made, that is, it is important to select efficient formats for program data. Third, both data marshaling and transmission must avoid excessive data copying at endpoints. Recent results published by our group[12] demonstrate that the performance of marshaling and transmission is strongly dependent on the 'wire format' used for data.

Changes to the discovery process *can* result in significant usability gains, helping to avoid the necessity of recompilation and/or relinking of components affected by message format changes.

3. The XMIT approach

Discovery using XML. XML is the metadata definition language used by the XMIT tool, which implies that the structures to be transmitted and their formats are described in XML. Usability is improved not only by making data structure explicit through XML, but also by exploiting the resulting independence of the discovery process; as long as the metadata is present when binding occurs, it matters not how the metadata got there. By introducing indirection into the discovery process, XMIT makes it possible for metadata to reside outside the program. One effect of this decision is that changes to the message formats used by distributed programs can be centralized, and XMIT ensures that they are propagated to all program components using these formats. At the same time, however, the fact that structured data is described in XML does not predicate that it is transmitted in this form (i.e., as ASCII text). Instead, we maintain at the level of XML only the association of metadata with data, and use efficient binary encodings of both during actual data transmission.

The XMIT approach provides several advantages:

- It adds flexibility in the definition of message structures. The XML description of a message structure can be retrieved and used to set up binary-format messaging, as opposed to compiling the structure definitions into the program. A change in the structure of data is easily stated as a change in the XML document indicated by a URL, thus insulating the program from changes in message format.

- The rapidly growing set of XML document display and manipulation tools can be used. For instance, since the structure of a message will be represented using XML, schema-checking tools may be applied to live messages received from other parties to determine which of several structure definitions a message best matches.

- The abstraction process inherent in the use of XML for metadata removes the need to consider some platform-dependent features (for example, structure padding).

- XML-based metadata allows application developers to concentrate more on the structure and content of the message and less on the details of the message transport.

Binding and marshaling to efficient, binary wire formats. XMIT is a run-time library; its function is to convert metadata definitions expressed in XML into more efficient metadata suitable for efficient binary transmission. Specifically, XML metadata is converted into an internal representation from which BCM-specific metadata is generated. We next present more detail on the two foundations on which XMIT is built: (1) an open XML-based schema system for describing data types, and (2) an efficient binary heterogeneous communication package. Evaluations of the benefits and costs of using XMIT appear in Section 4.

3.1. Constructing efficient metadata from XML schemas

In this section, we describe how XMIT constructs efficient metadata representations from XML schemas.

Types in metadata. Application programs draw from a well-understood set of primitive types that can be used for composition of message formats: integer, float, character, string (even if their specific definitions vary according to architecture and platform). An equivalent set is provided by the XML Schema specification [10]. XML Schema provides primitive types such as integer, string, and enumeration types that can be used to compose new abstract data types. Metadata definition with XMIT, then, starts with XML documents that contain appropriate type definitions.

The basic XML Schema data types are referenced by using the XML namespace convention[5]. This makes the entire set of data types in a namespace available to the programmer. The basic set of primitive types is fairly intuitive, comprising types such as `string`, `unsignedLong`, `float`, and `byte`[1]. The XML Schema tag `complexType` is used to denote the definition of a new type, named by the `name` attribute.

Inclusion of data elements of a previously-defined type in a new type is accomplished by using the name of an already-defined type in a new XML Schema `element` tag. Each `element` tag includes a `type` attribute, which is a character string. In order to use an element of a previously defined type, the `type` attribute is set to the name of the previously defined `complexType`.

Array types are also specified using attributes of the `element` tag. The `type` attribute is used to specify the base data type for the array. The `maxOccurs` attribute is used to specify the array bounds, and may have either a numeric or string value. If the value is numeric, the value will be used as the absolute size of the array. If the value is the wildcard character `*`, the array is treated as dynamically-allocated. Lastly, if the value is a string, an element of type `integer` with an identical `name` attribute must be present in the structure definition; the value of this variable will be used at run-time to indicate the size of the array. This functionality allows dynamically-allocated arrays to be specified.

Constructing native metadata. XMIT includes an API that allows a programmer to first "load" the toolkit with message definitions (contained in XML documents) from one or more URLs. Once the desired definitions have been obtained, the type of native metadata to be generated is selected. Once this is done, each XML document that contains XML Schema-based message definitions is parsed and the native metadata generation process is carried out.

The XML parser creates a Document Object Model [3] tree-based representation of all of the syntax elements in the document. Constructing native metadata from this tree is done by a selective traversal. First, subtrees of the document tree corresponding to the set of all `complexType` element tags are extracted. Each one of these subtrees defines a separate message format. Each subtree is then traversed to pick up its `element` nodes, each of which corresponds to a field in the message format.

For each `element` node, the XML Schema data type is retrieved. The selection of a native metadata system implicitly selects a mapping from the supported set of XML Schema data types to those supported by the native system. The mapping also includes information such as structure offsets and data type sizes for BCMs requiring them. Lastly, the XMIT produces an appropriate binding token representing the collection of message formats. This token can be

[1] The complete set of XML Schema data types, as well as their definitions, is available at `http://www.w3.org/TR/xmlschema-0/`

used directly with the chosen BCM to perform marshaling and unmarshaling.

3.2. Supported BCM metadata representations

XMIT generates “native” metadata in several different forms. It is designed in a modular fashion so that support for additional BCMs is easily added.

PBIO. PBIO [14] provides facilities for encoding application data structures, so that they may be transmitted in binary form over computer networks or written to data files in a heterogeneous computing environment. PBIO has the ability to select from different metadata formats for a specific encoded record, and it offers efficient encodings of such formats. The use of PBIO within C or C++ programs, however, typically relies on compiled knowledge about such formats, thus not realizing the gains in usability we seek to attain. XMIT removes the need for compiled-in structure formats when using PBIO.

Java. We support two different methods of integration with Java-based systems. XMIT can generate Java source code from a set of XML Schema descriptions, with the individual elements of each `complexType` represented as fields of a class. Composition of message formats is expressed as Java object composition. The Java source code is generated with references to the Remote Method Invocation (RMI) mechanism already in place, so that it can more quickly be used. More interestingly from our point of view, XMIT can generate Java bytecode corresponding to these classes through the use of a third-party bytecode generator. These bytecodes are automatically loaded into the Java VM, so that the classes are immediately available to the running system.

Others. We plan to implement SOAP/XML-RPC style interfaces and also IIOP.

4. Evaluation of the XML-based XMIT metadata tool

We have constructed a set of experiments and used them to evaluate the performance of XMIT’s integration with the PBIO package. We believe it fairly obvious that separating the metadata discovery and binding processes provides benefits in usability for metadata definition. Our goal in building XMIT was to investigate whether these benefits can be attained while still realizing the performance advantages of binary data transport.

Our results indicate that using XML as a metadata definition language can yield a large gain in usability with almost no performance penalty:

- Binary transmission of structured messages is superior in almost every case to using XML as a wire format. This holds *except* when senders and receivers use XML directly, when they transmit messages as unstructured ASCII, and when these messages are small. In this case, the additional overheads implied by the use of XMIT outweigh the benefits attained from the improved efficiency of binary transport.

- For the high performance applications and usage scenarios for which XMIT was developed (e.g., for the Hydrology application discussed below), we have experienced superior performance for binary transport using XMIT even when binary transport performs in its worst case (small messages, encoding/decoding required at both ends) and XML transport is at its best (small messages, no encoding/decoding required). In one application-based experiment, XML messages are 3 times larger than the corresponding binary messages (see Figure 1 for the structure definition and XML encoding), resulting in the XML-based solutions experiencing twice the latency than the solutions using XMIT.

4.1. XML is inappropriate as a wire format

Our argument for the contributions of XMIT rests on the claim that, while describing message formats in XML provides usability advantages over using native BCM metadata, binary transmission is necessary for high performance applications. Usability advantages are seen in the recent emergence and rapid adoption of XML-based communication protocols and systems[9, 6, 1]. However, our previous work[12] shows that XML suffers from the necessity of performing string conversions on both sending and receiving ends of a connection. Systems using XML as a wire format can both avoid concerns over heterogeneity of machine architectures and remain robust in the face of dynamically changing message formats. However, the price of this flexibility is prohibitive; encoding/decoding times are between 2 and 4 orders of magnitude greater than binary mechanisms.

4.2. Characterizing the performance of XMIT

Since the introduction of XML metadata into a system whose communication is performed using PBIO doesn’t add any additional overhead to data transport, it isn’t meaningful to compare communications times of systems with and without such metadata. Any metadata-related overhead occurs only at program startup; since the approach we describe uses PBIO format registrations derived from the provided XML format descriptions, PBIO-based communications can continue as if normal PBIO metadata were being

```
typedef struct
{
  int timestep;
  int size;
  float *data;
} SimpleData

<SimpleData>
   <Timestep>9999</Timestep>
   <Size>3355</Size>
   <Data>12.345</Data>
   <Data>12.345</Data>
   <Data>12.345</Data>
    ....
    ....
    ....
   <Data>12.345</Data>
</SimpleData>
```

Figure 1. A C structure defining a sample message and a sample XML encoding of the structure. The XML expansion results in a considerably larger representation of the data, significant when exchanging many messages.

used. This allows any increased cost of discovery and registration to be amortized across the entire set of messages sent using a particular metadata format. As the number of messages sent in a particular format can reasonably be expected to dominate the number of format discoveries and changes, the overall effect on performance should be tolerable. Note also that in both compiled-metadata and IDL-metadata systems, format changes require manual intervention at every source and sink point (in the form of recompilation of systems that cannot cope with the format changes).

The impact of using XMIT lies in the additional time required to retrieve XML Schema-based metadata and the time required to parse and construct BCM (in this case PBIO) metadata. We define the **Remote Discovery Multiplier** (RDM) as the ratio of the time needed by XMIT to register a message format with respect to to the time needed by PBIO to register the same format using compiled-in metadata. RDM provides an indication of the *cost of remote metadata*; that is, the quantifiable performance penalty associated with the harder-to-quantify usability benefits derived from the use of XMIT.

4.3. Experimental environment

Our experiments were conducted on a Sun Ultra 1/170 with 128 Mb RAM running Solaris 7. The XML parser package used by XMIT is the Xerces-C parser[8] available from the Apache Project. The XML documents containing the message formats were hosted on an Apache HTTP server in uncompressed form.

```
typedef struct asdOff_s {
  char* centerId;
  char* airline;
  int flightNum;
  unsigned long off;
} asdOff;

IOField asdOffFields[] = {
  { "centerID", "string", sizeof (char*),
     IOOffset (asdOffptr, centerId) },
  { "airline", "string", sizeof (char*),
     IOOffset (asdOffptr, airline) },
  { "flight", "integer", sizeof (int),
     IOOffset (asdOffptr, flightNum) },
  { "off", "integer", sizeof (unsigned long),
     IOOffset (asdOffptr, off) },
};

<xsd:complexType name="ASDOffEvent">
  <xsd:element name="centerID"
        type="xsd:string" />
  <xsd:element name="airline"
        type="xsd:string" />
  <xsd:element name="flightNum"
        type="xsd:integer" />
  <xsd:element name="off"
        type="xsd:unsignedLong" />
</xsd:complexType>
```

Figure 2. The top item is a C structure definition representing a hypothetical message. The middle and bottom items are metadata representations for the structure, expressed in PBIO and XMIT metadata, respectively. Note that these metadata formats are not themselves exchanged as messages are sent; rather, format identifiers are generated which allow component programs to retrieve the metadata on demand.

4.4. Proof-of-concept implementation

The gains in usability attained from the use of XML for data definition imply only small additional costs compared to using lower-level metadata representations like PBIO. In Figure 3 we characterize the time required to parse and register metadata for different structure sizes (see Figure 2 for an example structure). Format registration time for XMIT includes the time necessary to parse the XML description of the format and register the format with PBIO. **Structure Size** is the size of the language-level structure in bytes. **Encoded Size** represents the size of a buffer resulting from a marshal operation in PBIO for each metadata approach.

Note that the Remote Discovery Multiplier remains relatively constant even as the structure size increases. This indicates that XMIT imposes a constant factor of overhead

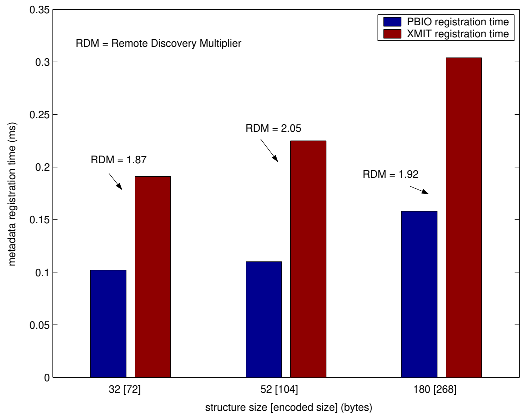

Figure 3. Format registration costs using PBIO and XMIT.

on the metadata discovery process. Also, it is important to note that registration time does not necessarily increase in strict proportion to message size, but instead corresponds more closely to the complexity of the message (in terms of size, number of fields, and nested definitions). The measurements in this figure show that the additional flexibility attained by use of XML comes at a small price. The performance penalties implied by using XMIT depend on the complexity of the data structure being used.

4.5. An application example

One of the early high-performance demonstration applications developed by NCSA researchers was a component-based visualization system for hydrology data (we will refer to this application as *Hydrology*). The architecture of this application (shown in Figure 5) comprises several distributed processing and visualization components that communicate using a shared set of message formats. *Hydrology* is representative of the use of PBIO in production applications, and so presents a useful opportunity to evaluate XMIT. We had previously modified this application to use PBIO as a wire format.

As a consequence of our modifications, the metadata corresponding to the message formats was compiled into the application. We removed the compiled-in metadata definitions from the application, and used XMIT to retrieve the message formats from an HTTP server. These message formats (see Figure 4) were translated by XMIT at run-time into PBIO-style metadata. The PBIO metadata was then used to marshal and unmarshal data sent between the components of the application.

Cost of remote metadata in *Hydrology*. In order to evaluate the impact of XMIT, we performed tests similar to the

```
typedef struct {
     char *name;
     unsigned server;
     unsigned long ip_addr;
     pid_t pid;
     unsigned long ds_addr;
} JoinRequest;

<xsd:complexType name="JoinRequest">
  <xsd:element name="name"
        type="xsd:string" />
  <xsd:element name="server"
        type="xsd:unsignedLong" />
  <xsd:element name="ip_addr"
        type="xsd:unsignedLong" />
  <xsd:element name="pid"
        type="xsd:unsignedLong" />
  <xsd:element name="ds_addr"
        type="xsd:unsignedLong" />
</xsd:complexType>

typedef struct {
  int timestep;
  int size;
  float *data;
} SimpleData;

<xsd:complexType name="SimpleData">
  <xsd:element name="timestep"
        type="xsd:integer" />
  <xsd:element name="data" type="xsd:float"
        minOccurs="0" maxOccurs="*"
        dimensionPlacement="before"
        dimensionName="size" />
</xsd:complexType>
```

Figure 4. Sample C structures and XMIT metadata representations from the *Hydrology* application.

ones described above in order to determine a Remote Discovery Multiplier. Figure 6 displays the results of this experiment; as in the previous experiment, format registration time includes the time necessary to parse the XML description of the format and register the format with PBIO, and **Structure Size** is the size of the language-level structure in bytes.

The results of this experiment reinforce two conclusions about XMIT. First, for structure sizes similar to those in the proof-of-concept implementation, the RDM remains within the same order of magnitude. This indicates that the overhead of using XMIT is still tolerable. Second, it is important to consider the structure of the data type for types where RDM varies widely. For example, in the first experiment, a 180 byte structure produces an RDM of 1.92, while a 152 byte structure from the *Hydrology* application produces an RDM of 4. The reason for this is that the first structure is constructed primarily of composing other structures, while

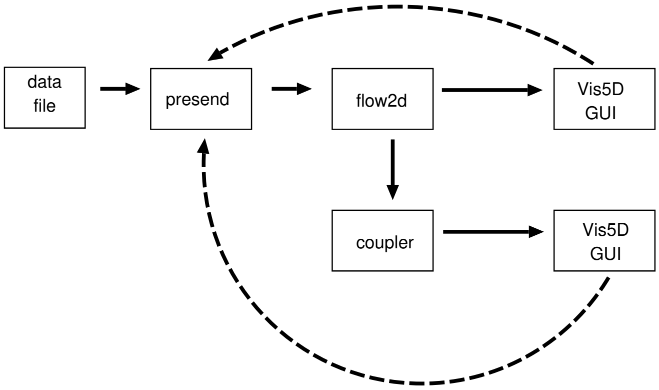

Figure 5. Architecture of the Hydrology application. Solid arrows represent data flows, and dashed arrows represent control/feedback channels. The message formats used in the data flow and control channels are shared among all application components. Data is read from a file and visualized after processing by different components.

the *Hydrology* structure is constructed of a large number of primitive data types. This large number of data types increases the amount of work the XMIT parser and metadata generator must perform, compared to the natively-defined metadata. Our future work on XMIT will explore this problem further in search of ways to keep the RDM at a reasonable level with respect to the native structure size.

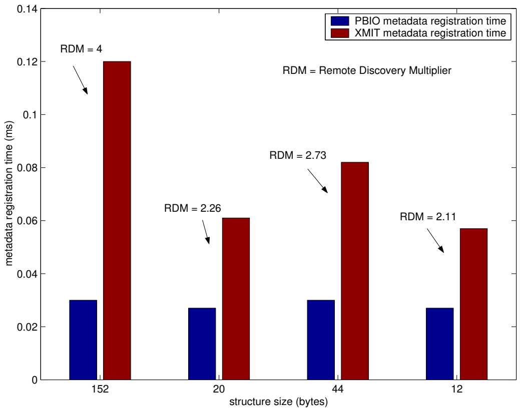

Figure 6. Format registration costs using PBIO and XMIT for the *Hydrology* application.

Impact on marshaling time in *Hydrology*. In addition to not imposing unduly on the metadata registration process, XMIT also does not add to the time necessary to marshal application data buffers. Figure 7 displays the results of marshaling different application data structures with native PBIO metadata and metadata generated by XMIT. These results demonstrate that the XMIT translation process results in native metadata that is just as efficient as compiled-in metadata. **Encoded Size** is the size of an application data structure after the marshaling process has been performed.

For purposes of further comparison, previous research[12] has established that MPI[15] takes on the order of 10 times as long as PBIO to encode a structure of comparable size ($\sim$100 bytes) to those in this experiment (Figure 8 provides an illustration of encoding costs using PBIO, XML, CORBA, and MPI). For such a structure, Figure 7 shows that PBIO using XMIT-generated metadata performs essentially identically as it does when using native metadata. These results show that XMIT can generate entirely comparable metadata to PBIO, and also allow encoding times that greatly outperform systems like MPI.

4.6. Summary of results

Our experiments support our contention that the usability benefits of XML-based metadata can still be obtained at negligible performance cost with respect to native binary-format metadata. Furthermore, we have shown that XML-only data transmission, while providing the same usability benefits, does not allow the high-performance communication that a distributed application using XMIT can achieve. XMIT is still relatively immature, but we have demonstrated that remotely-defined metadata can be exploited in this manner with beneficial results for applications, their developers, and their users.

5. Related work

In a sense, most research on high-performance computing that involves the exchange of structured data is to some degree related to our work. All such packages have some facility for describing the structure of messages that will be

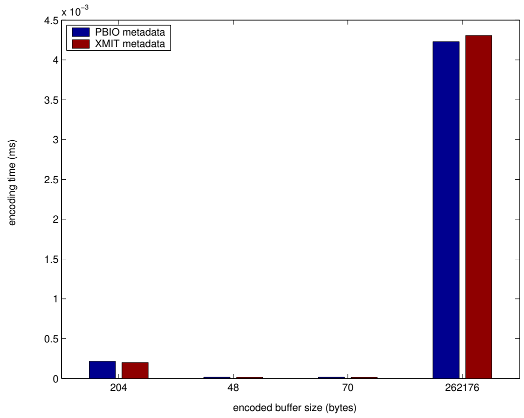

Figure 7. Structure encoding times using PBIO and XMIT for the *Hydrology* application.

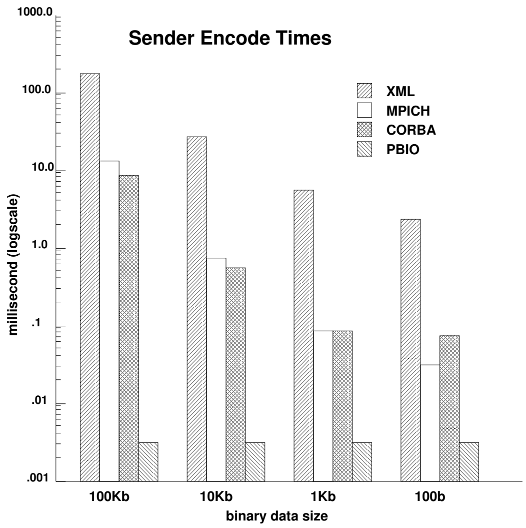

Figure 8. Send-side encoding times for various message sizes and binary communications mechanisms.

exchanged and governing the translation of messages into a specific wire format. Their differences with respect to this research lie in how they define and manage this metadata, and to what degree they can make use of efficient communication mechanisms.

Packages like PBIO, PVM[17], and Nexus[16] support language-level features for defining messages in which the communicating applications "pack" and "unpack" messages, building and decoding them field by field. Their operational norm is for all parties to a communication to agree *a priori* on the format of messages. The support for remote metadata in XMIT relaxes the coding impacts of this agreement. PBIO supports a form of restricted evolution in message formats in which elements may be added to message formats without causing receivers of previous versions of the message to fail. Other packages, such as MPI[15], allow the creation of user-defined data types for messages and fields and provide some marshaling and unmarshaling support for them internally.

Another general class of communication systems uses IDLs to define the structure of messages. Systems such as Sun RPC[7] and CORBA[2] provide an implementation-language-independent set of data types that can be arbitrarily composed. The resulting structure definitions are processed by a code generator that produces implementation-language code that is included in the application program. Our use of XML Schema happens at run-time, which results in added benefits in terms of interoperability and the potential use of analysis and verification tools now being developed. We know of no commonly-used specification for automated exchange of IDL definitions, even though CORBA IDL is relatively mature. In contrast, exchanging metadata defined in XML leverages (nearly) ubiquitous HTTP transport services.

A third class of systems uses object systems technology, providing for some amount of plug-and-play interoperability through subclassing and reflection. This is a significant advantage in building loosely coupled systems, but there are performance penalties for communications that have strong impacts in the high performance domain. For example, CORBA-based object systems use IIOP as a wire format. IIOP attempts to reduce marshaling overhead by adopting a "reader-makes-right" approach with respect to byte order (the actual byte order used in a message is specified by a header field). This additional flexibility in the wire format allows CORBA to avoid unnecessary byte-swapping in message exchanges between homogeneous systems but is not sufficient to allow such message exchanges without copying of data at both sender and receiver. XMIT compares favorably to this class of systems; many XML-parsing packages [13, 8] allow run-time investigation of message formats. Also, our integration with an efficient wire format provides high-performance communication.

Systems using XML as a wire format or as a metadata description language[9, 6, 1] take a different approach to message format flexibility. Data is transmitted in ASCII form rather than in binary, with each record represented in text form with header and trailer information identifying each field. This allows applications to communicate with no *a priori* knowledge of each other. However, XML encoding and decoding costs are substantially higher that those of other formats due to ASCII/binary conversion. In addition, XML transferred as ASCII text has substantially higher network transmission costs because the ASCII-encoded record is larger than the binary original (an expansion factor of 6-8 is not unusual [12]).

6. Conclusion

We have presented the motivation for and details of the implementation of XMIT, a tool for run-time discovery of metadata for high-performance communications. In addition, we have detailed a method of constructing metadata based on data types taken from the XML Schema specification. Our results indicate that XMIT enables applications to obtain the benefits of using XML for metadata description while still allowing them to communicate using high-performance binary data communications mechanisms, at low overhead cost.

In the future, we intend to explore dynamic incorporation of new message formats into applications at run-time, as well as generation of language-level message object representations in both C++ and Java.

Acknowledgments

We are grateful to Fabiàn Bustamante, Van Oleson, and Dong Zhou for their assistance and comments. This work was supported by the DARPA ITO Directorate's *Information Expedition* program, the State of Georgia's Yamacraw Research Initiative, and by equipment donations from Sun Microsystems and Intel Corporation.

References

[1] Blocks Extensible Exchange Protocol. `http://www.bxxp.org`.

[2] Common Object Request Broker Architecture. `http://www.omg.org/corba2`.

[3] Document Object Model. `http://www.w3.org/DOM`.

[4] The extensible markup language (XML). `http://www.w3.org/TR/1998/REC-xml-19980210`.

[5] Namespaces in XML. `http://www.w3.org/TR/1999/REC-xml-names`.

[6] Simple Object Access Protocol. `http://www.w3.org/TR/SOAP`.

[7] XDR: External data representation standard. IETF RFC 1014.

[8] Xerces XML Parser. `http://xml.apache.org/xerces-c`.

[9] XML-RPC specification. `http://www.xmlrpc.com/spec`.

[10] XML Schema. `http://www.w3.org/XML/Schema`.

[11] R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker, and B. Smolinski. Toward a common component architecture for high performance scientific computing. In *Proceedings of the 8th High Performance Distributed Computing (HPDC'99)*, 1999. `http://www.acl.lanl.gov/cca`.

[12] F. Bustamante, G. Eisenhauer, K. Schwan, and P. Widener. Efficient wire formats for high performance computing. In *Proceedings of Supercomputing 2000*, November 2000.

[13] J. Clark. expat - XML parser toolkit. `http://www.jclark.com/xml/expat.html`.

[14] G. Eisenhauer and L. K. Daley. Fast heterogenous binary data interchange. In *Proceedings of the Heterogeneous Computing Workshop (HCW2000)*, May 3-5 2000.

[15] M. Forum. MPI: A message passing interface standard. Technical report, University of Tennessee, 1995.

[16] I. Foster, C. Kesselman, and S. Tuecke. The nexus approach to integrating multithreading and communication. *Journal of Parallel and Distributed Computing*, pages 70–82, 1996.

[17] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM 3 Users Guide and Reference manual*. Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, May 94.

[18] NCSA. Environmental hydrology demo. `http://scrap.ssec.wisc.edu/˜rob/sc98`.

[19] C. M. Pancerella, L. A. Rahn, and C. L. Yang. The diesel combustion collaboratory: Combustion researchers collaborating over the internet. In *Proceedings of SC 99*, November 13-19 1999. `http://www.sc99.org/proceedings/papers/pancerel.pdf`.

[20] B. Parvin, J. Taylor, G. Cong, M. O'Keefe, and M.-H. Barcellos-Hoff. Deepview: A channel for distributed microscopy and informatics. In *Proceedings of SC 99*, November 13-19 1999. `http://www.sc99.org/proceedings/papers/parvin.pdf`.