Mediating Among Diverse Data Formats: Thesis Summary

John Ockerbloom

January 14, 1998

1 Introduction: The Problem of Diverse Data Formats

The tremendous growth of the Internet and its World-Wide Web has allowed millions of people to publish and view vast quantities of data. Much of this data, however, cannot be used effectively by a large audience, because data on the Internet comes in a wide variety of *types* (conceptual structures and interpretations of data) and *formats* (representations of data types as byte sequences). A typical client can only work with a small number of these types and formats. Data will continue to appear in a wide variety of formats, since different formats are suited for different applications, and new formats are being introduced all the time. Existing strategies are inadequate for handling this data diversity, so a new approach is required.

I claim that it is possible to construct a system that allows clients to work with a much broader range of data types and formats than they can today. Because there are so many data formats on the Internet, from such diverse sourcese, I claim that a successful system for this purpose must be *distributed.* A distributed system can take advantage of information and resources throughout the Internet, and work with the Internet's varied data formats in a scalable manner.

This distributed system needs to fulfill four basic requirements. The first is *feasibility*, or the ability to carry out its tasks correctly, robustly, and efficiently. The second is *composability*, or the ability to work with a wide range of past, present, and future data formats, as well as with a wide range of programs that provide and use this data. The third is *expressiveness*, or the ability to describe in detail the function and representation of unfamiliar data types, and the services available to work with them. The fourth is *scalability*, or the ability of the system to handle a growing number of data formats defined by people and groups from all over the Internet.

My thesis represents a new approach that allows users to cope with the ever-growing set of data types and formats used on the Internet. The approach has two parts:

1. An object-oriented model (the Typed Object Model, or TOM) that describes the abstract structure of data formats, their concrete representations, and relations between types.

   The model is expressive enough to encompass formats already in use, as well as new and complex types of data. TOM's data model is simplified in some respects from standard object-oriented models. Most notably, TOM treats objects as immutable values, rather than as mutable variables (though its use of references takes possible changes in the environment into account). This principle allows TOM to work with a wider range of servers, delegate computation more flexibly, and take advantage of more options to relate different types and formats than the standard object-oriented model allows.

2. A distributed network of mediator agents (known as *type brokers*) that collect and disseminate type and format information, and give clients access to servers that convert and operate on data in various formats.

> These agents allow users and applications to look up type information, add new types, use data in unknown formats, and convert data to known formats while controlling information loss. They also allow information providers to define new types and formats, and register new services that operate on these formats. The brokers act as go-betweens connecting client requests with appropriate services.[1]

The central claim of this thesis is as follows:

> It is feasible to design and implement a data model and a type broker network that allow individual data providers to define complex data in a wide variety of formats, and make these data formats widely usable throughout the Internet. A system that uses this model and this network is superior in working with diverse data formats, compared to previous widely-deployed systems, in terms of its expressiveness, its support for composability, and its scalability.

2 Related Work

Many systems have been designed to handle the distribution of data in various forms on the Internet. There are two important design dimensions for such systems. The *component model* of systems explains how the computational parts of the systems are structured, connected, and arranged. The *data model* explains how the data in the system is structured. The data model of an Internet information system can be at least as important as its component model. Many information systems have a special class of data called *metadata* used to manage the "ordinary" data in the system, such as the URL metadata used in the Web to locate documents.

Many information systems can be described in a *layered* architectural style, arranged as in figure 1. The *repository* layer can be thought of as a passive data store. The *server* layer provides data to *clients*, possibly taking the data from one or more repositories. Between data servers and end-user clients, *mediators* provide assistance such as caching, protocol translation, and brokering between client requests and server programs.

The World Wide Web is a well-known example illustrating these concepts. The Web includes HTTP servers (at the server layer), which generally serve files stored in filesystems (the repository layer). The files are viewed through Web browsers like Netscape or Mosaic, at the client layer. Clients can establish a connection with any Web server (except for those behind firewalls or otherwise limiting their access). However, some clients interact with the Web's HTTP servers indirectly, via a *proxy server*. Proxy servers act as mediators between Web browsers and HTTP servers, and are therefore in the mediator layer. The Web serves data in various formats, with a special emphasis on HTML. It uses "URLs" as metadata to indicate how objects on the Web can be retreived.

Most Internet-based information systems, TOM included, are based on client-server connections; examples from the Internet's history include FTP [B$^+$71], Gopher [AML$^+$93] and the World Wide Web [BLCGP92]. Data on these systems resides on a particular server, which can cause bottlenecks for popular data stores. Systems like Usenet news [KL86] and DNS [Pos94] replicate and partition their data store, lessening the load on any one server. Similarly, TOM's type brokers replicate type and format information, easing loads on any given broker.

[1] For simplicity, I will use the term TOM to describe both the data model and the network of software agents described above. Both the model and the actual agents are key contributions of this thesis. It will usually be clear from context whether TOM is being used to refer to a model or to computer programs, but I will clarify meaning when necessary.

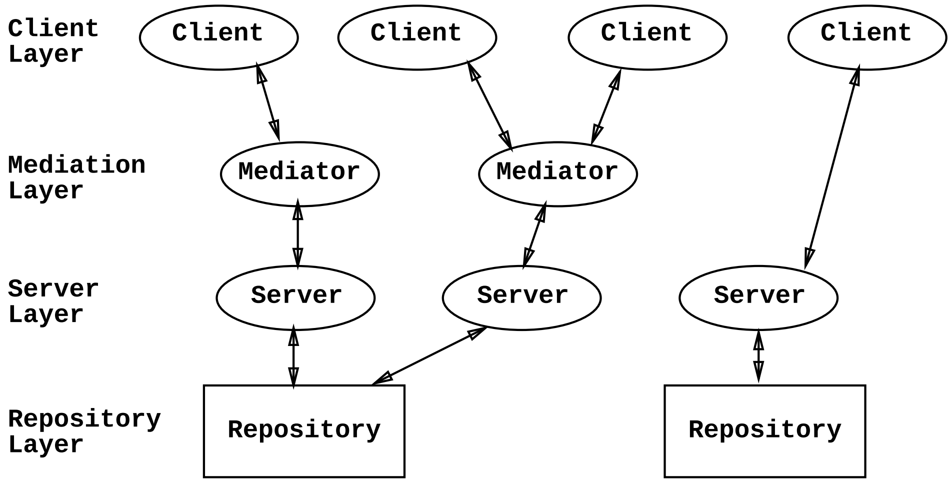

Figure 1: A common layering of components in information systems

Large information systems can be easier to use when mediators or other "experts" are present. In [Wie92], Gio Wiederhold cites numerous uses for mediators, such as finding servers with information a client seeks, coordinating multiple operations to fulfill a user's request, and making systems more fault-tolerant and reliable. Examples of systems that use mediators include Netfind [SP94] MetaCrawler [SE97], Indie [DLO92], and Rufus [SLST93]. The first three examples all help with search. The first two provide front ends to a variety of heterogeneous services, and the third includes a network of brokers that maintain and exhange search information. Rufus is a mediator-based system that gives an object-oriented view of data typically found on a filesystem. TOM's type brokers also act as mediators, and they imitate the data-sharing practices of Indie and the object-oriented adaptation provided by Rufus.

Internet information systems use a variety of ways to describe data, ranging from ad-hoc suffix conventions to sophisticated mechanisms. Three particularly interesting data description mechanisms in use today are MIME [BF92], SGML [Pub86], and OEM [PGMW95]. MIME is used widely in electronic mail and on the Web to name formats, but its descriptive power and scalability are limited. SGML supports self-describing data structures, and makes them easy to parse, but it contains no semantic support, and cannot handle most "legacy" formats. OEM supports the translation of various data formats into a common intermediate data format, and back again. TOM's approach to data translation is more flexible, and does not depend on a single intermediate format.

Recently, a variety of systems have emerged that allow data to be manipulated in an object-oriented fashion over the Internet. Three such systems in widespread use are the Object Management Group's CORBA [Gro92], Microsoft's COM (the basis for OLE and Active-X), and Sun Microsystems' Java [GM95]. These systems, unlike TOM, support writes as well as reads of objects, and are hence well-suited to general-purpose object-oriented applications. However, they are less flexible about allowing objects to be transmitted in arbitrary representations, and are hence less suited for information discovery, and for interpreting legacy data.

3 TOM's data model

TOM's data model allows data to be manipulated either through an abstract, object-oriented interface, or through directly manipulating byte sequences. TOM annotates data with information

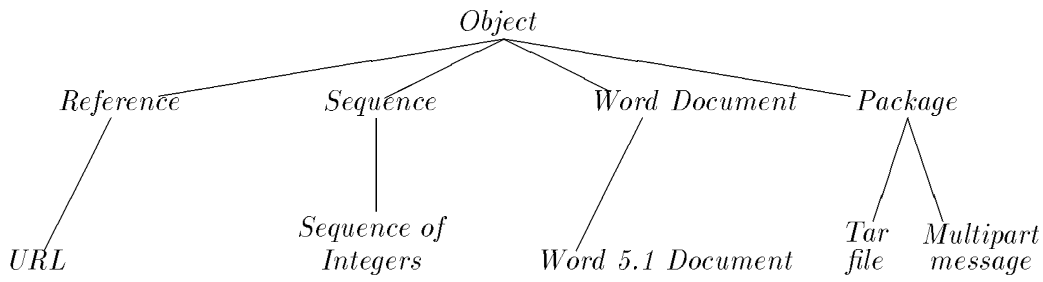

Figure 2: A portion of TOM's type hierarchy.

about its type and format, so that clients can determine which interface they can use to manipulate the data, how the data is represented, and how it can be converted to other formats.

TOM models data as immutable objects. A TOM object is a value (not a variable) annotated with the name of a type. Objects can also be encoded as simpler objects (including simple byte sequences), allowing them to be communicated over the Internet or stored in databases or file systems. Encodings are one-to-many partial relations between objects of different types. They allow applications to use information at varying levels of abstraction, according to their needs and capabilities. *Formats* specify a representation of objects as byte sequences, and consist of a composition of encoding relations between a type and the byte sequences that ultimately represent objects of that type. A *shipped object* consists of a byte sequence accompanied by a specification of a format, indicating what object the byte sequence represents.

Type interfaces include *attribute* and *method* specifications. Methods work much like they do in standard object-oriented systems, except that methods cannot mutate TOM objects. Attribute fetches are basically handled like method calls, but take no arguments, and their return value is not dependent on context, like method calls can be.

Types can have *subtypes*, but the subtypes must be *substitutable* for their supertypes. This means that a client program familiar with a type T can use an object whose type is a subtype of T through T's interface, and the object will behave exactly as the client would expect an object of type T to behave. This property allows clients to safely use objects of unfamiliar types through the interfaces of familiar supertypes. TOM's definition of substitutability is similar to that of Liskov and Wing [LW94]. Because TOM objects do not mutate, however, it is not necessary to model history properties, so it is easier to make substitutable subtypes in TOM than it is using Liskov and Wing's general definition. For example, in TOM a sequence of integers can be a subtype of a generic sequence. In Liskov and Wing's model, it cannot be, since strings can be inserted into generic sequences but not into sequences of integers. Since TOM's treats objects as read-only values, inserting elements into sequences is outside of the scope of TOM.

A portion of TOM's type hierarchy can be found in figure 2. Lines are drawn between supertypes and subtypes. For instance, "sequence of integers" is a subtype of "sequence". The complete TOM type hierarchy includes over 100 types at this writing.

TOM also allows *conversions* to be defined on formats, translating objects from one format to another. Information is often lost in a conversion, so TOM supports a notation called *intersub-*

stitutability that can be used to specify what information is preserved in a conversion. A type T is an intersubstitutable type for conversion C if the input and output of conversion C cannot be distinguished when viewed through the interface of type T. (Or, to put it another way, the original object can substitute for the converted object, and vice versa, with respect to T's interface.)

The behavior of methods, attributes, and conversions, and the nature of the encoding and subtype mappings between objects, can be further described with a semantic specification. TOM allows either formal or informal specifications to be associated with a type, method, attribute, encoding, or conversion. The specifications are themselves given in the form of objects. Implementations of the corresponding operations or encodings must conform to these specifications.

As an example of how this data model is applied, consider a generic "sequence" type. The type has an attribute called "length", an integer that indicates the number of objects in the sequence. It has a method called "elem", which, given an integer argument n, returns the n'th object in the sequence. It has a single encoding called "standard", which concatenates the shipped objects representing each object in the sequence.

This type has a number of subtypes, all of which must support the "elem" method and the "length" attribute as defined in the "sequence" type, in order to be substitutable. Two possible subtypes of "sequence" would be "zip file" (which is used for a sequence of files) and "mail spool" (which is used for a sequence of email messages).

Suppose that we define a conversion from a "mail spool" format to a "zip file" format. Let us call an input to this conversion s and the resulting output $C(s)$. Furthermore, let us say for any such s, the value of the "length" attribute for s and $C(s)$ are the same, and the "elem" method, when called on s and $C(s)$ (with otherwise identical input and context) also always returns the same value for either shipped object. If this is the case, then "sequence" is an intersubstitutable type for this conversion, since the conversion does not change the behavior of an object when viewed through the interface of the "sequence" type.

4 TOM's type broker architecture

The basic architectural model of TOM consists of three kinds of components: clients, servers, and mediators called *type brokers*. Servers provide information, and perform operations on this information; clients retrieve information, and request perform operations on them; and type brokers manage information about the servers that operate on the typed objects, and about the types themselves. Type brokers also act as mediators for operation requests TOM components (clients, servers, and brokers) are also known generically as *agents*. Agents can use type brokers to look up type information, invoke other agents that implement parts of an object's interface, convert objects to more familiar formats, and work with objects of unknown types and formats through the interfaces of more familiar supertypes. Developers of new types and servers register information about them with a type broker. The type broker will then pass the information along to other interested type brokers.

TOM-aware agents communicate using a special protocol (TOP, the Typed Object Protocol). TOP includes commands to operate on objects, register new type and agent information, and retrieve information about types. TOP's complexity and efficiency is comparable to that of HTTP.

The computation in TOM can be viewed as a three-level architecture, as shown in figure 3. A client can perform an operation on an object by sending a request to a type broker, along with the object and any other required data. The type broker will then pass along the request to a server that implements the operation. (Some type brokers also implement certain operations themselves.) Alternatively, a client can bypass the broker and directly contact a server that implements the

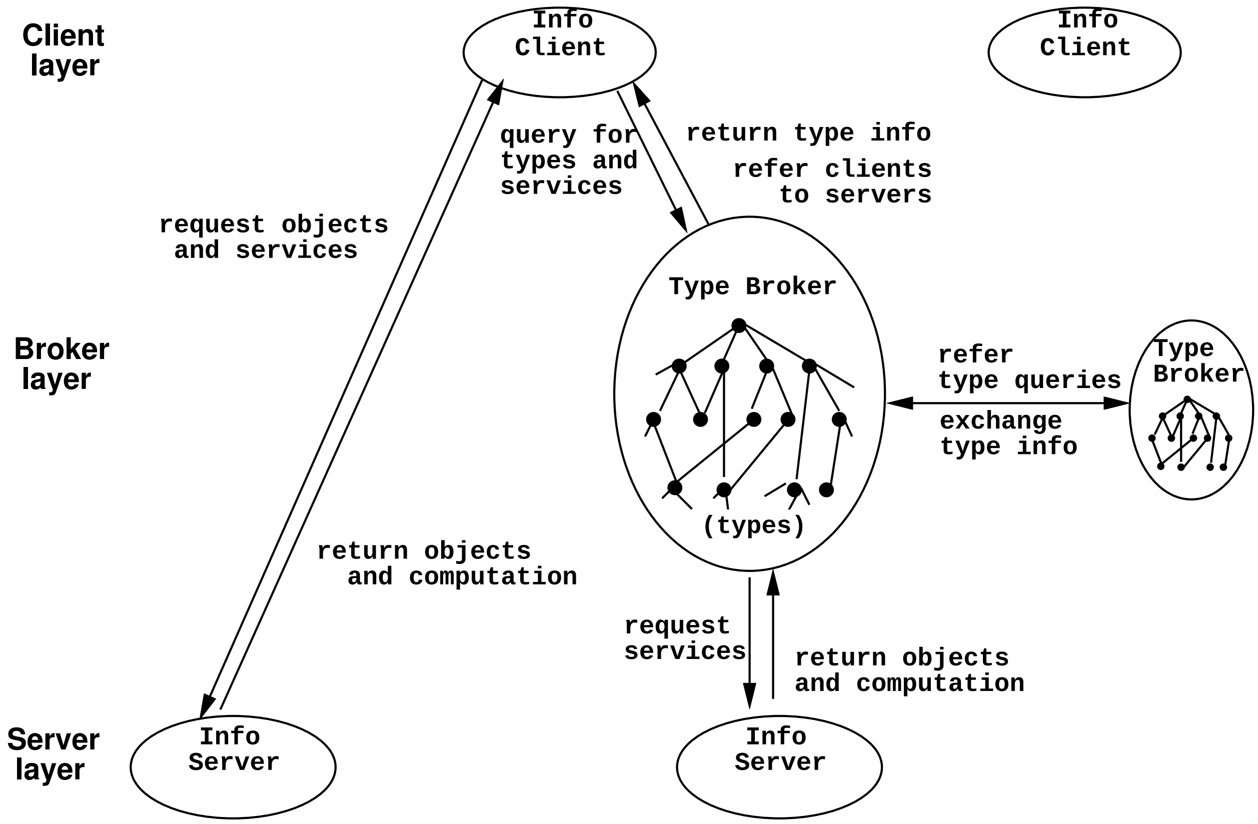

Figure 3: The basic components of TOM

operation. Another alternative is for the client to send a request to a type broker to convert the object to a form that the client can operate on directly. To conserve bandwidth, TOM allows objects being sent to or from an operation to be communicated by reference, rather than by value.

TOM interoperates with clients and servers outside of TOM through the use of *gateways*. Gateways are mediators between TOM agents and non-TOM agents. Server-side gateways receive TOM requests and translate them into requests to non-TOM programs, such as Web servers. Client-side gateways receive non-TOM requests through some API (for example, the Web's CGI interface) and translate them into non-TOM requests. Gateways can also communicate data between TOM agents and non-TOM agents.

Type brokers represent information about types as objects, which can be examined, operated on, and converted just like any other TOM object. A *type description object* contains all the information used to define a type (as described in the previous section) along with certain other information associated with the type. Specifically, it contains:

- Names used to refer to the type.
- The names of supertypes.
- Encoding specifications for the type.
- A specification of the type's semantics.
- The definition of methods and attributes of the type.
- Conversions from the type's formats.
- Administrative information about the type, such as its creator and its publication status.
- For each method, attribute, or conversion, a list of agents that implement the operation.

Type brokers store information about many related types, allowing them to reason about the overall graph of types in useful ways. For example, they can plan a complex conversion by composing a series of simpler conversion steps.

TOM's type brokers are designed to handle failures gracefully. TOP allows servers to return different codes depending on the reasons for failure. This allows brokers to adjust their behavior appropriately for different failure modes. For example, if a "server error" code is returned, a broker will contact another server to try the same operation, if one is available, but if a "bad input" code is returned, the broker will not attempt this.

Type brokers are arranged in a peer network. A type broker queries its peers about newly registered or changed types, and update its own database of type description objects accordingly. In this way, new type information propagates throughout the Internet.

5 Scaling Up: The Growth of the Type Graph

In order to reap the full advantages of a distributed typed information system, we need more than just an expressive data model and reliable brokers. We also need to allow the universe of known types to grow in a decentralized fashion, and we need to ensure that when new type information is added, it does not invalidate existing type information. This requires us to carefully consider what constraints TOM should place on registering new type information.

For instance, in order to avoid name conflicts between new types and old ones, TOM divides its namespace into a hierarchy of subspaces, controlled so that type name conflicts do not occur. For example, the subspace of type names starting with "net:" is further subdivided according to the domain name of the broker on which the type is first registered. That type broker ensures that it does not assign the same type name more than once. Other type namespaces can be more centrally controlled, and allow shorter, and more mnemonic names. A type can have multiple names, assigned from different namespaces, so that a popular type originally defined with a long, hard-to-remember unique name can also get a shorter, more easily remembered name from a more tightly controlled namespace.

Besides assuring name uniqueness, TOM also needs to ensure that new type information does not change the semantics of existing types in unexpected ways. TOM therefore prescribes these rules for new type information:

- It should be possible to extend type descriptions with new information, such as new methods, encodings, and agents. This enables an existing type to be used in new ways, without requiring a new type definition every time a new use for a type is found.
- Extensions to published types must be *conservative*; that is, they must not require that the type carry more information than it originally did (lest existing implementations and representations be invalidated), and they must not invalidate any of the existing constraints on the type (lest existing client expectations be invalidated).
- Certain additional information about a type must be approved by a type's maintainer, if incorrect information might make the type unusable.

These principles lead to the following rules:

- After a type is published, its name, its declared semantics, and the signature and semantics of its initial operations cannot change.
- New type names, encodings, and conversions can be added at any time, as long as they do not introduce naming conflicts, and are consistent with the type's semantics.
- Agents for particular operations may be added or deleted at any time, to reflect available services on the network. The agents may implement the operations in any way consistent with the operation's declared semantics.
- New attributes and methods can only be added if their return values can be derived from existing attributes and methods. (In other words, these new attributes and methods are convenience functions, and do not represent any new functionality not already supported by the type. For example, a "degrees-celsius" attribute could be added to a type that already has a "degrees-fahrenheit" attribute, since the Celsius value can be inferred from the Fahrenheit value.) New class methods, which do not take an input object and therefore do not depend on the functionality of the type, can be also added at any time.
- New subtype relations can only be asserted by the owner of a subtype, since the assertion of a subtype relation may affect the semantics of the subtype, and may cause the subtype to behave in unexpected ways if the subtype claim is false (for example, if the subtype is *not* substitutable for the supertype, as TOM requires).

Because it sometimes takes a while for the definition of new type to stabilize, TOM allows new a type to be flagged as "experimental" when registered with a type broker. The definition and semantics of experimental types are subject to change, until the author of the type upgrades it to "published" status. Clients are warned not to depend on the type's semantics, and type brokers might decide not to propagate experimental types to other brokers. This "experimental" stage allows for flexibility and experimentation while designing a type, without forcing a large number of obsolete types to litter the type graph.

Because new methods and subtype relations can be declared at any time, it is possible for naming conflicts to be inadvertently introduced between two methods coincidentally sharing the same name, defined on different types. To avoid these conflicts, and to clarify which methods in subtypes specialize supertype methods, TOM requires that a method in a subtype that specializes a method in a supertype explicitly declare the method it is specializing. This declaration also allows renaming of subtype method specializations when this is necessary, such as when two methods in unrelated supertypes share the same name.

6 TOM in action

The thesis describes two case studies of TOM in action. The TOM Conversion Service is a scalable, distributed application for converting documents that can be run from any Web browser. I show it is usable by novices and experts worldwide, and that it gives access to a wide range of conversions, in a more scalable manner than existing conversion applications. The TOM Frame Service allows users without a frame-enabled Web browser to view and navigate through framed web sites. The TOM Frame Service application, and its definition of new TOM types for framed HTML, show how TOM enables new data formats to be introduced and adapted to existing applications more smoothly than they have been introduced on the Web.

6.1 The TOM Conversion Service

TOM's network of brokers maintain extendable databases on types and conversions. TOM is therefore particularly well-suited for handling conversion problems. TOM's brokers can handle an unlimited number of conversions, and take advantage of conversion services spread out around the Internet. Clients for these conversion services can be quite small. Our smallest conversion application program was only a few hundred lines of C, yet could take advantage of the full range of conversions defined in TOM's broker network.

We found that a practical conversion service had to handle three challenges. The first is that many users need to convert documents like mail messages, zip files, and other documents where the data to be converted is *part* of a document, rather than the document itself. The second is that documents often come without any meta-information that identifies their format. The third is that novices and experts both need to be able to use the tool effectively. (We designed the conversion service not only for computing professionals, but also for schoolteachers new to the Web.)

The infrastructure of TOM made it possible to address all three of these challenges in a scalable, straightforward manner.

To solve the first problem, for instance, we defined a generic "package" type, which included attributes and methods for describing items in the package, and extracting them from their packaging. We included derived operations so that common forms of description and extraction could be handled with a single method call. We then made mail messages, MIME multipart messages, and the other packages *subtypes* of the generic package type. This allowed the TOM Conversion service to handle all kinds of packages in a uniform manner, without having to know details about any of them.

To solve the format identification problem, we designed routines that would identify formats from patterns within the documents themselves, and could be augmented to handle new formats simply with a change in a configuration file. This format identification routine could then be invoked like any other TOM operation, allowing clients to take advantage of TOM's broker network to handle format identification if necessary.

To solve the user interface problem, we took advantage of the well-defined, modular interface that TOM provides through its type brokers. This interface can be composed with a variety of user interfaces. We implemented three distinct conversion clients, each with its own interface. These ranged from Web-based interfaces that novices can easily use, to Unix command-line interfaces that can convert an entire directory to the user's specifications. The Web-based interfaces represented client-side gateways between TOM and the Web. We were also able to easily implement server-side gateways, so that TOM could retrieve documents from the Web.

The ability to incrementally bring more information and services into TOM makes it clearly superior in functionality and ease of use to more monolithic conversion techniques. Users no longer have to find and install standalone conversions to convert files. Rather, they can just request what kind of conversion they want, and if the appropriate conversions have been installed anywhere that has been registered with a TOM type broker, the appropriate conversion programs are automatically invoked. Likewise, users do not have to settle for a fixed set of conversions supported by a particular standalone conversion program. Rather, TOM clients automatically take advantage of new types, operations, and conversions that are added to TOM's global type graph.

6.2 The TOM Frame Service

The TOM Frame Service case study allowed us to compare how easily new data formats are adopted on the Web with how easily they are adopted with TOM. Two slightly different definitions of "frame sets" were introduced to the Web in 1996 by rival browser companies. Neither definition has yet been adopted as an official Web standard, as of late 1997. Frames posed major backward-compatibility problems with older browsers, and even browsers specifically designed to handle frames have taken months or longer to correctly manage backtracking and bookmarking.

With TOM, I was able to define a generic "frameset" type that covered the common features of both companies' frame definitions. (The company-specific frameset definitions are subtypes of the generic "frameset" type). I also put together a service that would convert framed HTML pages into non-framed HTML pages, and allow frameless web browsers to navigate through the framesets, in a few days. The operations on frame types and the CGI/Web interface take up only about 1000 lines of code combined, not including the code already in the generic TOM server kernel.

The TOM Frame Service shows how TOM can help make the Web fulfill one of its original design goals: the wide dissemination of of data in expressive, widely-usable forms. Originally, the Web managed this through common standard for formats like HTML and URLs, but vendor extensions soon resulted in sites that could only be viewed through specific browsers. TOM allows the creation of more expressive data types, such as those created by vendor expressions, while still allowing them to be viewed and used by a wide variety of clients.

7 Conclusions

With TOM and its applications, I have shown the feasibility of a data model and a type broker network that allow individual data providers to define complex data in a wide variety of formats, and make these data formats widely usable throughout the Internet. Furthermore, I have shown TOM to be superior to previous widely-deployed systems, in expressiveness, composability, and scalability.

TOM's expressiveness is both broad and deep. Its breadth can be seen in the range of types and formats TOM supports: at this writing, TOM has definitions for over 100 types and thousands of formats. TOM formats cover a wider range of types than MIME, and a wider set of transmittable representations than systems like CORBA, COM, or Java. TOM's depth of expressiveness is also superior to both MIME and to common Internet-based object-oriented systems. Unlike MIME, TOM supports the invocation of operations and conversions on objects. TOM also supports varying representations and substitutable subtyping to a degree not supported by other networked object-oriented systems.

TOM is also highly composable, in three imporant respects:: first, the ability to use a wide variety of data structures, including legacy data; second, the ability to interoperate with a wide variety of systems and application programs; and third, the ability to be adopted and integrated incrementally, with little investment required to reap initial benefits, and larger benefits reaped with further investment in the system. (Or, to express all this in another way, TOM tolerates a wide variety of practices in managing and using data.) TOM's ability to handle legacy data, through conversion and encapsulation, is a particularly important strength as the volume of legacy data worldwide increases. TOM's broker architecture also allows TOM to act as a mediator to bring diverse components together, supporting most of the mismatch bridging strategies suggested by Shaw [Sha95]. TOM can be adopted at several levels of investment, each offering greater rewards. One can simply use existing TOM services, like the TOM Conversion Service, through ordinary Web browsers, or one can install one's own type brokers, and define new types, services, and applications, to take advantage of the full range of TOM's capabilities. This is described in more detail in the thesis.

The thesis case studies also show that TOM carries out its tasks reliably and with acceptable performance. (Moreover, judging from the number of European users of the TOM Conversion Service, TOM's response time is acceptable even over transatlantic links.)

I designed the architecture of TOM to scale gracefully as the number of types, users, and services increased. Experience with the types and services we have added to the two type brokers at Carnegie Mellon indicates that TOM scales up well in practice. TOM is designed ultimately to be deployed widely, with many independently-managed brokers on different sites on the Internet. It is difficult to predict with certainty whether new problems will arise at that scale, but early indications are promising. Analysis in the thesis shows that TOM should scale better in many ways than the Web does. For example, the Web's form of reference, the URL, requires that a particular server be contacted for resolution. This can cause bottlenecks for popular services delivered via the Web. In contrast, TOM provides server-independent ways to refer to type services, and gives type brokers the ability to transparently contact any of multiple servers capable of performing a service.

8 Contributions

The ultimate goal of my thesis work is to make it easier clients across the Internet to take full advantage of a wide range of complex data types and formats. I have focused primarily on "read-only" applications such as data discovery, display, and analysis, rather than on "read/write" applications such as those involving data mutations. In this area, I have made substantial contributions.

My contributions towards the goal stated above are of two kinds:

Useful programs, type information, and services: The artifacts and type and format information developed for this thesis are themselves important contributions towards more flexible use of diverse data formats. The TOM Conversion Service is used worldwide to convert and unpack objects that clients cannot interpret themselves, and the TOM Frame Service provides "backward compatibility" of framed Web services with World Wide Web browsers that do not adquately handle frames. The type brokers implemented for this thesis mediate for dozens of services, accommodate several kinds of clients, and handle registrations for new types, formats, and services in a clean, scalable manner. The graph of type information that these brokers maintain covers hundreds of commonly-used (and less commonly-used) formats, and includes useful hierarchies like the "package" hierarchy. TOM makes it much easier for people and programs to handle the ever-growing range of data types and formats on the Net, without requiring them to keep up with all the new data types or maintain their own routines to handle all the old data types.

Lessons on how to make complex, heterogeneous data widely usable. Aside from providing the artifacts above, I have analyzed the design of TOM and discussed the strengths and weaknesses of the system. My design and analysis of type brokers gives an example of how mediators can be used in a loosely-managed distributed system to take advantage of knowledge and services from across the Internet. I have also defined and demonstrated the utility of a data model that combines many of the advantages of object-oriented "black-box" data types and low-level concrete representations. I have shown how that data model can be applied to a distributed, heterogeneous environment. I have shown how a "read-only" model of objects, though limited in its application, can take advantage of data from a wide variety of servers, including legacy data; and how a system using that model can raise the common denominator of data types that are widely communicated and understood. I have introduced a measure of information loss in conversions, known as "intersubstitutability", and shown how it can be used to safely and automatically convert between formats while preserving vital information, without requiring the use of a "standard" intermediate data representation.

9 Future work

Much more can be done based on my research to date with TOM. Possible future work includes:

Extensions of TOM's design and implementation include:

- **Adding facilities for client-run code** such as Java applets, so that clients can learn how to execute operations on objects themselves.
- **Using TOM in conjunction with other systems** such as CORBA, Java, and COM, as well as in appropriate domain-specific systems such as digital libraries.
- **Extensions for security and electronic commerce.**
- **Adding extensions to the data model** such as generalized exceptions, first-class parameterized types, and parameterized conversions.

Further research involving TOM includes:

Empirically studying multi-site scalability once TOM type brokers are deployed and used outside Carnegie Mellon.

Studying the utility of new specification mechanisms beyond the signatures, intersubstitutable types, and informal semantic specifications that TOM currently supports.

Studying how TOM can be applied to read/write applications either through using it in conjunction with other systems, or through extending TOM's own data model.

Further research beyond TOM includes:

Advanced mediator research in areas such as searching, cataloging, and cooperation between different types of mediators

Solving heterogeneity problems in other domains such as those involving similar but incompatible protocols and APIs.

New applications for widely distributed data systems and studies of their social impact.

The population of the Internet – both the number of people that use it and the volume of data it contains – continues to boom. I expect the growth of the Internet, computing power, data storage capacity, and network bandwith to continue at a rapid pace for at least 10 more years, and possibly substantially longer. New and significant distributed data systems and applications should continue to be introduced for at least that long. I hope that while this expansion period continues, my thesis work will serve as a basis on which people can build to facilitate data interoperability, and also serve as a case study for designing widely distributed information systems to accommodate diverse needs.

References

[AML+93] F. Anklesaria, M. McCahill, P. Lindner, D. Johnson, D. Torrey, and B. Alberti. The internet Gopher protocol (a distributed document search and retrieval protocol). Internet Request for Comments (RFC) 1436, March 1993.

[B+71] Abhay Bhushan et al. The file transfer protocol. Internet Request for Comments (RFC) 172, June 1971.

[BF92] N. Borenstein and N. Freed. MIME (multipurpose internet mail extensions): Mechanisms for specifying and describing the format of internet message bodies for the format of ARPA internet text messages. Internet Request for Comments (RFC) 1341, June 1992.

[BLCGP92] T. J. Berners-Lee, R. Calliau, J-F Groff, and B. Pollerman. World wide web: The information universe. *Electronic Networking: Research, Applications, and Policy*, 2(1):52–58, 1992.

[DLO92] Peter B. Danzig, Shih-Hao Li, and Katia Obrazacka. Distributed indexing of autonomous internet services. *Computing Systems*, 5(4):433–459, 1992.

[GM95] James Gosling and Henry McGilton. The Java (tm) language environment: A white paper. Available from Sun's Web server at <URL:http://java.sun.com/whitePaper/java-whitepaper-1.html>, 1995.

[Gro92] Object Management Group. *The Common Object Request Broker: Architecture and Specification*. QED Publishing Group, Wellesley, MA, omg document number 91.12.1, revision 1.1 edition, 1992.

[KL86] Brian Kantor and Phil Lapsley. Network news transfer protocol. Internet Request for Comments (RFC) 977, February 1986.

[LW94] Barbara Liskov and Jeannette Wing. A behavioral notion of subtyping. *ACM Transactions on Programming Languages and Systems*, 16(6):52–58, November 1994.

[Ock98] John Ockerbloom. Accommodation: The key to making widely adopted composable systems. In *Proceedings of the Workshop on Compositional Software Architectures*, Monterey, CA, January 1998. to appear.

[PGMW95] Yannis Papakonstantinou, Hector Garcia-Molina, and Jennifer Widom. Object exchange across heterogeneous information sources. In *Proceedings of ICDE*, Taipei, Taiwan, March 1995.

[Pos94] J. Postel. Domain name system structure and delegation. Internet Request for Comments (RFC) 1591, March 1994.

[Pub86] Association Of American Publishers. *Standard for Electronic Manuscript Preparation and Markup*. Association of American Publishers, Washington, DC, 1986.

[SE97] Erik Selberg and Oren Etzioni. The MetaCrawler architecture for resource aggregation on the web. *IEEE Expert*, 12(1):8–14, 1997.

[Sha95] Mary Shaw. Architecture issues in software reuse: It's not just the functionality, it's the packaging. In *Proceedings of the Symposium on Software Reusability*, April 1995.

[SLST93] K. Shoens, A. Luniewski, P. Schwarz, and J. Thomas. The Rufus system: Information organization for semi-structured data. In *Proceedings of the 19th VLDB Conference*, Dublin, Ireland, 1993.

[SP94] Michael F. Schwartz and Calton Pu. Applying an information gathering architecture to Netfind: A white pages tool for a changing and growing internet. Technical Report CU-CS-656-93, University of Colorado at Boulder, Boulder, CO, 1994. revised July 1994.

[Wie92] Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, March 1992.