BINARY XML REQUIREMENTS

As consolidated from the Report From the W3C Workshop on Binary Interchange of XML Information Item Sets

24th, 25th and 26th September, 2003, Santa Clara, California, USA

http://www.w3.org/2003/08/binary-interchange-workshop/Report.html

The Blue comments are the ways in which CoreSystem addresses each requirement. These comments should be studied top down since terms are introduced and expanded in this order.

Maintain universal interoperability - all tools, all documents
CoreSystem converges the four general data models to create a uniform access layer providing an access solution for heterogeneous sources. This new ***context*** data model supersedes **file**, **relational**, **object**, and **XML**, thereby merging the distinction of the domains implied in these legacy data models.

Continue to work with existing parsers and tools?
A context file does not require parsing because it's self-describing aspects is maintained in a separate ***facet*** file maintained in a ***Community Repository***. A facet file is dynamically bound to an instances of a context and can be locally cached on individual ***sites***. A site is an IP node that contains a CoreSystem Virtual Machine (VM) referred to as a ***CoreMachine***.

Do not want a domain-specific solution [e.g. wireless]
The semantic architecture of a context is based upon a rich meta-metadata model that is quite universal and applicable to virtually any domain or discourse.

Efficient storage is important [compactness]
An instance of a context is all binary representations (except for actual UNICODE text). Vast fragmentation may take place during creation, modification, and deletion ***operation*** execution on specific elements and ***attributes***. The CoreMachine re-serializes the instance prior to transmission or storage as a background task. Therefore, the only remaining transmission latency appears as traffic delays. An instance is stored in the file's original format and requires no conversion back to memory representation.

Efficient transmission is important
Instance transmission is optimal since non-text data is represented in binary and all self-describing markup is effectively delegated into a facet. Particular ***composite*** tree branches can easily be excluded from a transmission if not required by the recipient. Remaining transmitted fragments are automatically compacted and serialized to minimal stream size.

Support both storage & messaging representations
Context is defined in three forms: ***module***, ***plexus,*** and ***service***. These forms share the identical composite architecture, but play distinct system roles. A composite represents a branching tree ***structure*** where each branching point is a ***node***. A key advantage to this unified architecture is that it allows sharing of behavior constructs that greatly simplifies human

understanding of a system design and reduces resource logic requirements for machine processing.

A ***module*** is a cluster of objects that represents some specific thing: (e.g. an employee record)

A ***plexus*** is a complex data type or document and is represented as a module attribute: (e.g. a photo of an employee)

A ***service*** maps data between heterogeneous systems: (e.g. financial transaction)

Also, a ***module*** and ***service*** can be declared as a single ***fusion*** context meaning that data instances/records can be copied directly between the storage and messaging realms without transformation.

Want fast decompression, even if it means slower compression
Gzip can be applied in applications where a binary service context instance is still not compact enough. Since there is no parsing demand, any decompression latency will always out perform native XML or binary XML formats.

10x faster than 2003 best practice with textual XML
We address this requirement as a processing demand other than parsing and transmission latencies. The CoreMachine has sophisticated built-in traversal, edit, and transfer operations that perform all in-memory instance transformations on a pointer reference and manipulation basis. These operations take advantage of the tight alignment with the memory manager that performs garbage collection, heap compaction, and storage virtualization three orders of magnitude more efficient than .Net's VM. Exact metrics have not yet been benchmarked, but we expect a minimum of 100x overall performance acceleration.

Support parsing on low power device
No parsing required on native context instances. Parsing is still required when a native XML document is presented to any CoreMachine-enabled device. Since low power devices are typically on proprietary networks, we expect that these early adopters would just prefer to transport context services anyways.

Must reduce processing time (compared with parsing) including data binding time
The CoreMachine can ***convert*** and transfer any practical pair of atomic data types with no loss of value fidelity. A ***coerce*** is a specialized conversion where transfer may result in some fidelity loss. Native big-endian attribute values are automatically converted to little-endian upon request by a foreign caller to the CoreMachine.

Want to be able to create binary thingies directly, not just via pointy brackets
DataString is an atomic data type in CoreSystem and can hold binary "thingies" of any nature. These type containers are effectively partitioned in the CoreMachine and cannot execute arbitrary behavior without explicit permissions. The CoreMachine can examine and manipulate the content of a DataString.

Performance comparable to (or better than) RMI
Remote Method Innovation is transparently performed between network objects on an ***asynchronous*** basis. This ***messaging*** capability is purposely constrained to sites controlled by

the same ***entity*** (a person or enterprise). This lightweight protocol bypasses authentication and permission machinery. Otherwise, all data transportation between entities must be performed through a service ***dialog***. A dialog provides authentication, permission, and long transaction capabilities between entities. Again, since there is no parsing, service instance data can be quickly moved from and to module objects at each ***requester*** and ***responder*** site through ***pointer calculus***.

Must support packaging - bundle multiple files together, maybe with nesting
A context may be declared with multiple ***roots*** that can ***stem*** out to virtually an unlimited number of ***branch*** and ***leaf*** nodes. Each node (element) collection is declared to a particular structure type. Alternatively, a leaf can be declared as ***ClearText***. Whereas a **structure** node represents ***typed data***, a **clearText** node represents ***document text***. This clear separation between data and text representation is in stark contrast to XML that is inherently obtuse and expensive to process. The ClearText technology overcomes the "overlapping hierarchy" problem that limits XML to represent advanced encoding.

Must support random access based on infoset (XPath) or other boundaries, e.g. image, page, indexed; indexes built from stream
CoreMachine operations can ***traverse*** any context instance tree by ***scanning*** or ***searching*** node attribute values. A search predicate is based upon a series of ***logical*** and ***relational*** expressions that walk up a tree to ***target*** a node ***position*** or node ***range***. In addition, a module context can be accessed via a ***key index*** or ***object instance ID*** as well. A ClearText node also supports ***transclusion*** that makes it possible to include or share passages of complete documents over the Internet, without requiring a literal copy of the document: the referenced material is pulled in from the original (which may have been revised in the meantime) at the time of the transaction.

Should be able to update in place based on Xpath
The built-in traverse, ***edit,*** and ***transfer*** operations completely replace the need for an external technology like Xpath. CoreMachine's shared operation patterns process any ***structure collection***, ***string***, or ClearText construct. This architectural singularity greatly reduces user and machine complexity.

Fragments - be able to start reading at fragment boundaries [e.g. repeated sub-elements for repetitive broadcast] interchange fragment context (e.g. location in document tree)
This requirement can be divided into ***design*** and ***execution*** requirements.
Design – There are two forms of composites. A ***local composite*** is exclusive to a particular module, plexus, or service, whereas a ***fragment composite*** is shared between two or more of these contexts. A fragment enables a common schema to become a pattern resource that can be reused. Local and fragment (nested) composites can utilize a fragment. This resource sharing can take place over different composites within a community.
Execution – CoreMachine supports the natural operations of ***bud***, ***prune***, and ***graft*** of composite subtrees. A tree grows by progressively budding additional branch nodes as more detailed data is added. Pruning often occurs when details are to be omitted from transport. Grafting enables subtrees to be moved from one composite stem to another.

Must progressive download [e.g. progressive rendering] to look at a header before the whole stream is seen
A context instance contains a header structure that fully qualifies the remaining instance stream. All attributes are referenced by pointer-offset arithmetic from a base instance address. A

traversal operation may progressively "walk the stream" as additional packets arrive. Edit and transfer operations can execute immediately once memory boundaries are passed. Besides rendering, this capability is crucial when applied to deep packet inspection functions.

Want progressive generation (e.g. no packet-length at start)
A context instance resides in the heap and can progressively generate to an arbitrary size since it is composite of linked memory fragments. The CoreMachine virtualizes low priority fragments to accommodate a growing context instance in a transparent fashion.

Want XML Fragment Interchange, e.g. for query results & document subsets
Again, CoreSystem separates the concern between typed data and document text.
Typed data – A ***query*** operation can either traverse a linked set of networked objects or walk up a tree composed of objects. The result can either be a list of instance IDs or a set of ***table*** rows containing attribute values culled from traversed objects. Second order predicate expressions can be applied to a data table to further qualify results.
Document text – A ClearText string passage can be ***copied*** or ***moved*** from one document to another with no loss of generic markup that is maintained as ***associate element collections***. Specialized markup elements are automatically recognized if the ***source*** and ***destination*** documents are of the same ClearText node type.

Transmitted format shall be easy to convert to/from native XML Schema data types
CoreTalk has been working with Kantay Systems <www.kantay.com> in the development of generic tools that can parse virtually any XML (or other well-formed) schema into a context type. Their technology can also dynamically marshal documents of a type into context instances as well. The CoreMachine maintains integrity with the XML document infrastructure by directly serializing a binary context instance into text prior to transmitting to a non-CoreSystem node. Parameterization enables customization to meet particular XML format requirements.

Must support XML Security, e.g. via canonical XML & reconstruction thereof
A context instance always remains in a canonical form and natively processed by CoreMachine operations. These "core" operations are a finite set and represent the only method of processing due to the sophisticated pointer calculus machinery. It is virtually impossible to inject foreign instructions and is the basis for a secure ***net-centric operating environment***.

A context instance contains only data, and is free of metadata or operating behavior. The nested tree structure of the binary itself is even obscured from pattern recognition sniffing techniques. While this binary is inherently secure for most applications (thus eliminating the need for additional process-intensive steps of encoding and decoding) selected elements or only the semi-structured strings within a context instance may be encrypted for additional security. Alternatively, selected elements and even particular attributes may be excluded from a transmitted instance based upon a user privacy mask. A non-reputable digital signature can also encapsulate an instance.

Want arbitrary precision numerical data formats
A ***number*** attribute can be declared as a 8, 16, 32, 64-bit integer or a 32, 64, or 80-bit float type. Alternatively, a number attribute can be declared as a 8, 16-bit or BCD ***DigitString*** of arbitrary digit item ***capacity*** with a declared decimal place. A symbolic enumeration is represented as a 16-bit integer.

Want directory at the start, e.g. for allocation
Not sure what is meant by this requirement, but assume it deals with generation of new context instances. An instance is always allocated from a context scheme that is sourced from a particular Community Repository that maintains a directory of published schema types. This unique architecture assures representation integrity across the Net.

Negotiation: fall back to text format if receiver can't understand binary
A context instance always identifies its source Community Repository and a CoreMachine can fetch particular ***operation instructions*** and facets on demand if these files are not already locally cached in a site.

Must be able to distinguish text XML from binary format on inspection
CoreTalk takes issue with the fundamental premise of XML being a human usable information format in any but the most simplistic applications. Complex distributed computing requires extensive automation and by default requires sophisticated tool environments to enable "knowledge workers" to work effectively. Therefore, CoreTalk makes no contention that a binary context can be human inspected in any practical manner. The CoreDesigner environment moves the Web browsing experience to a universal iconic Net navigation experience. This technology greatly expands the world population that can effectively create, manipulate, and share contexts.

Must be clear about MIME media type to be used
Multipurpose Internet Mail Extensions refers to an official Internet standard that specifies how messages must be formatted so that they can be exchanged between different email systems. MIME permits one to include virtually any binary type of file or document in an email message to be base64 encoded as text. A ***DigitString*** attribute is foreign typed and maintains the original binary format of the source file when represented within a context instance. Therefore, the common encoding/decoding problems associated with MIME messages are not encountered. A CoreMachine simply innocuously transfers these embedded messages back and forth between foreign callers.

Must not rely on HTTP [e.g. file support]
The CoreSystem platform specification is self-contained and designed to work directly on top of TCP-IP protocol. The platform uses a "statefull" ***CoreProtocol*** stack that is independent from the Web's HTTP protocol and supports "intelligent" Net exchanges.

Support one-way communication
Not sure what is meant by this requirement, but assume it refers to a multicasting process where ***complex*** and ***document*** instances are distributed to multiple requester sites on a periodic basis. Server responder sites located at the edges of the Net perform this content maintenance and delivery task. An entity subscriber provides registration to a syndicate delivery list.

Must work on asymmetry in bandwidth
CoreSystem assumes that TCP-IP delivers clean assembled packet streams through CoreMachine's ***channel*** interface. Channel CoreProtocol logic takes full responsibility for maintaining asynchronous message and service dialog traffic integrity.

Can use schemas to help encoding

Any context instance is based upon a schema type sourced from a particular Community Repository that assures correct encoding.

Must support schema version detection, multiple schemas at once
A community declares a series of ***releases*** as a context undergoes transition to maintain synchronization within an evolving domain. Each release undergoes five life cycle stages: ***new***, ***draft***, ***published***, ***expiring***, and ***obsolete***. An instance based upon an older context version is automatically converted to the latest version when loaded into the memory heap if the particular site specifies this option. Foreign connector users are automatically informed of impending changes to simplify cut-in of revised connector interfaces to reduce community partner resistance to change and lower operating costs.

Robustness in face of mismatched schemas [e.g. reader makes right]
Mismatched schemas are not possible under the CoreSystem architecture.

May support download of new schemas, or some other way of schema evolution
A Community Repository automatically distributes the latest context release versions to member sites.

Not all devices will accept new schemas: support read-only schemas for recipient
A community member may limit context evolution, but will not be able to process a later version that it receives from another entity site.

Must support open content, e.g. elements, values, subtrees not in the schema also want to be able to modify understood items & pass on (forward) subtrees, including elements you didn't understand
All data under CoreSystem is strongly typed and tied to a structural ontology. Common understanding is achieved within and between communities of interest in an automated fashion.

Self-describing format
Again, the context model separates but maintains the association of an instance (content) from its meta-facets (format) information.

Solution must be mass market, commodity price points
CoreMachine is a software module analogous to the Java Virtual Machine. It will be ported to different platforms from router / server, PC, mobile and embedded devices. While the PC version will most likely be free, the other versions will command a royalty on a per device basis.

CoreDesigner is a comprehensive development environment, roughly in the class of Microsoft .Net Studio, Borland JBuilder, or Sun ONE Studio. Tools such as this are generally licensed to the user on a “seat” basis with an expectation that the client will pay an annual maintenance and/or upgrade fee.

But, CoreDesigner is also like a Web browser. Instead of interacting with hypertext pages, the user is navigating and interacting with shared context resources sourced from the distributed ***CoreMemory***. From this perspective, CoreDesigner is more like a “Net navigator” providing a window into a semantic–based knowledge infrastructure.

It is therefore important to remove all user access barriers so that anyone can navigate this knowledge infrastructure (with community membership) in order to assure ubiquitous market adoption across all domains.

Adapting the Adobe Arobat® business model, CoreDesigner can be freely downloaded to enable anyone to examine context resources or conduct service transactions. Use fees come into play only when new context is created and commercially distributed. Context is composed of ***CoreObjects*** that can be metered. This capability provides a linear means to charge for the use of CoreDesigner. This model accommodates the needs for both causal and professional users alike. Of course, will also offer the traditional one-time license model with version upgrade fee as well.

Must support fast access to individual parts of the infoset, [e.g. for headers] mixed mode encoding
All context instance access uses pointer calculus for very fast access to any node (element) or attribute. Encoding is uniformly controlled via community ownership and sharing.

Must support custom & multiple compression schemes for data types
Binary representation pretty much eliminates need for part compression, but can be performed on entire instance if necessary using standard encoding technologies.

Want to continue to work with SAX and DOM, Pull parsing, e.g. existing APIs
SAX and DOM play no part in the CoreSystem ecosystem. Instead, a ***connector port*** provides a foreign interface to a service instance. Each service context publishes a ***manifest*** that provides custom interface instructions. A manifest is auto-generated by a Community Repository and downloaded into an entity site. The motivation for this invention is based upon the observation that the primary purpose of a service is to transport complex data structures between heterogeneous systems. A manifest performs this function effectively by presenting a flattened interface to a service instance providing direct access to all nested nodes and their attribute values through foreign method calls. The application connector developer need not be concerned with any data access details and simply makes slot calls to write to and read from the instance. Write calls are constrained by both collection cardinality and attribute value constraints. An attribute value constraint is based upon either a number range or a string regular expression.

Must require minimal changes to application layer
The connector port method ***signature*** interfaces to Java and the C family of languages to support virtually any form of exchange with host application and database technologies.

Should support validation on receiving
A context instance is always in a canonical form and does not require validation.

Must approach efficiency of hand-coded (binary) formats [as per information theory]
The CoreSystem binary format is very close to “optimal” information theory and open to independent verification of this claim.

Must work for both data and documents
Again. whereas a **structure** node represents ***typed data***, a **clearText** node represents ***document text***. Each is processed in an optimal fashion.

May be willing to have lossy compression, infoset subset e.g. might be able to lose comments, processing instructions, whitespace (some applications may be able to lose some kinds of content, e.g. precision on SVG coordinates)
Will not support lossy compression. Design intent is extensively created and maintained in Community Repositories and ***Entity Repositories***. This intent is directly converted from icons-to-instruction bits for optimal processing. White space only exists in ***TextStrings*** and ClearText and remains innocuous. Strongly typed structured nodes support any level of precision for any domain.

Want round-tripability wrt. canonical XML
A canonical context instance can be processed between any set of sites that has been granted permission to a source Community Repository.

Must be able to represent complete infoset
All CoreSystem components are represented in ***coreObjects***. A coreObject is a ***unit of measure***, ***term***, ***template***, attribute, ***gateway***, ***reference***, structure ***template***, operation, ***method***, composite, or context. These levels of abstraction enable comprehensive capture of composite information in a uniform manner as shared ***meta-resources*** without presupposing a particular domain.

Should support arbitrary extensions to infoset
A Community Repository supports delegation semantics that enable the arbitrary extension to any data type or template declaration.

Consider efficient support for other data structures, e.g. linked list, directed graph, e.g. id/idref optimization to pointers, point to any character, xPointer or something
Composite node items are internally maintained as bi-directional linked lists and traversed through logical memory pointers. A module context maintains object instances as both node composite trees as well as networked instance graphs. String items (bits, digits, and characters) are automatically converted to bi-directional linked item lists when edited and re-serialized into a common memory fragment at session end.

Consider option to propagate inherited data (e.g. xml:lang, xmlns:) down
All CoreSystem coreObjects are uniquely identified on a global basis via a concatenated set of ID integers independent of their symbolic names. Each coreObject has one source held in a particular Community Repository. With permission, a particular coreObject can be a ***link copied*** by another Community Repository as an extended meta-resource and shared by these member sites as well.

Must work equally well on high-frequency stream of small messages that may grow bigger if you use gzip on them; also large files with many small objects, e.g. a million floats
CoreSystem provides the option to transport asynchronous messages between an entity's sites as one optimization technique to streamline high- frequency traffic. Between a pair of entities, a service dialog will grow on a linear basis since all self-describing information is delegated to a context facet file.

Must minimize bandwidth for both small and large messages
The context architecture is a optimal format for both small and large messages (dialogs).

Must be easy to implement
Internally, the CoreSystem architecture is extremely sophisticated to provide unprecedented automation to enable a much broader sector of the population to participate in the emerging knowledge economy.

Want to specify the order of serialization... e.g. may want to send header up front, or may want leaf nodes first and may need to inform receiving end
Not supported, we don't understand the motivation behind this ordering capability!