Efficiency Structured XML (esXML / esDOM) A Standard Binary Infoset and API

Stephen D. Williams

Sdw@lig.net swilliams@hpti.com http://sdw.st

Independent Researcher and Senior Technical Director for High Performance Technologies, Inc.

Project site: http://www.esXML.org

Agenda

- Introduction to esXML / esDOM
- Why – Status Quo is Painful:
  - Serialization, IDL, RPC
  - Parsing, Memory Allocation, Small Copies
  - Programming Overhead
- What – Requirements, Paradigms, Solutions
- How – esXML / esDOM
  - esXML – Elastic Memory, VPTRs, XML, COW, Streaming
  - esDOM API
- Other Efforts, Results, Outlook

Context and Primary Goals

- esXML is fully equivalent to XML 1.0 with new, mostly better encoding
- esXML can be fully converted to and from XML 1.0 at any point
- esDOM is the preferred API, although other APIs are easily possible with degraded performance
- esXML capable libraries should also support XML 1.0 for debugging, configurable at runtime; utilities will also easily convert
- esXML is not a compression or serialization

Efficiency Structured XML (esXML)

- esXML is full XML semantics in a binary infoset
  - Directly and efficiently modifiable
  - Wire format and memory format are identical
  - No parsing/serialization except at XML 1.0/esXML boundaries (legacy, debug)
  - Direct support for binary data (images), pointers, and deltas/COW (versioning, session, streaming, undo/exceptions, checkpoint/restart)
  - Network and language portable
  - Self-describing, not IDL or compression based
  - Directly targets N-Tier XML component and application environments, but has wide applicability

Efficiency Structured DOM (esDOM)

- esDOM is a collections/STL/DOM style API
  - Provides minimalist interaction with application data
  - Access similar to 3GL object and library get/set/find
  - esDOM with esXML backend avoids:
    - Data initialization, copy, and linking
    - Memory allocation, deallocation, and garbage collection
    - Language data structures for application/business objects
    - Processing overhead related to data conversion/serialization
  - Supports scoped reference objects to maintain OO
  - Supports advanced intermediation such as tracing, secure access, event processing, rule engines

Why esXML/esDOM?

- Speed and efficiency
- Scalability
- Broadened applicability
- Less preparation
- Less coding
- Less debugging
- New semantics
- Rich intermediation

"Serialization Considered Bad"

- Flipside: "Parsing Considered Bad"
- Serialization/Parsing are not required
- Predates XML: basic to 3GL vs. files, databases, and network communication
- Layering "compression" doesn't help

Interface Definition Languages

- Good as documentation, except:
  - Enforcement by code generation generally
- Schemas/DDL are better
  - Optional validation of data
- Non-self describing encodings are bad:
  - Schema migration, versions, pre-arrangement
  - IDL: CORBA, DCOM, ASN.1/OSI*,WMPEG7
  - Hand designed: MPEG4, ONC-RPC
- Business applications are worst case

Remote Procedure Calls

- Communication model affects efficiency and design
- Half-duplex, synchronous, ties up channel, >O(n)
- Assumes interaction semantics:
  - Client/server roles
  - One call, one response or many responses (ODBC/SQL)
  - Can't easily handle asynchronous, peer, out of order responses.
- Scalability/Efficiency demands real asynchronous message oriented communication and pub/sub for some applications with easy management of many APIs, versions, with standardized security, <O(n)
- SOAP supports both, usually done as RPC

Parsing, Memory, Copies

- Text parsing and object creation costs a lot
- Binary parsing still costs
- Memory allocation / de-allocation / garbage collection and object creation / destruction can greatly outweigh other processing
- Copying large numbers of small items and setting similar references has similar impact and breaks zero-copy goals of efficient design
- These destroy and prevent economies of scale
- Hidden aspects such as poor locality of reference and allocation overhead are important and avoidance of these can provide extra efficiency

Programming Overhead

- Every line of code not implementing business logic and business rules is wasted time, effort, maintenance, and performance
- Accepted wisdom for C++/Java is to write setters/getters to intermediate object data
- Complex data structures and standard algorithms achieved by collection/template libraries
  - Standard use of high level API
  - Representation is private to library
- IDL, stubs, versioning, data conversion take effort and maintenance

Programming Overhead'

- Single definition of business objects is preferred, ex. schema
  - Should not require repetition in each application, language, and database
  - Application code should be application code, with little or no data or meta-data definition, management, and conversion (serialization, allocation)
- Need flexible intermediation, reification, and in-memory transactions/savepoints/COW

Goals

- Increase efficiency & remove overhead
- Improve development process / libraries
- Introduce certain semantics to 3GL
  - Delta/COW, relative virtual pointers
- Portable, flexible, (proto) standard
- Support, but don't require token compression, indexing, etc.
- Support in-place modification efficiently

Requirements

- Full implementation of all XML 1.X semantics
- Support C++, Java, Perl, Python, etc.
- Reduce or remove parsing/serialization
- Fixed buffer memory management (pools, reuse)
- Support arbitrary modification, I.e. frequently changing application variables
- Compact representation, but allow for amortization of frequent change overhead
- Support copy on write: sessions, versions, savepoints
- Provide everything needed for business object programming
- Support arbitrary logical data structures - pointers
- Support updates and streaming
- Directly support binary inclusion without encoding

New Paradigms

- Wire format and memory format are the same
  - Avoid data conversion / serialization at network, file, component, or database boundary
- Directly modifiable compact data structure
  - Modification overhead is offset by better locality of reference and avoiding other overhead
- Business objects always maintained in esXML
  - Avoid creating data structures and code
- Advanced semantics for applications: delta, comprehensive intermediation
- Global Efficiency: minimize effort to process, store, communicate, program

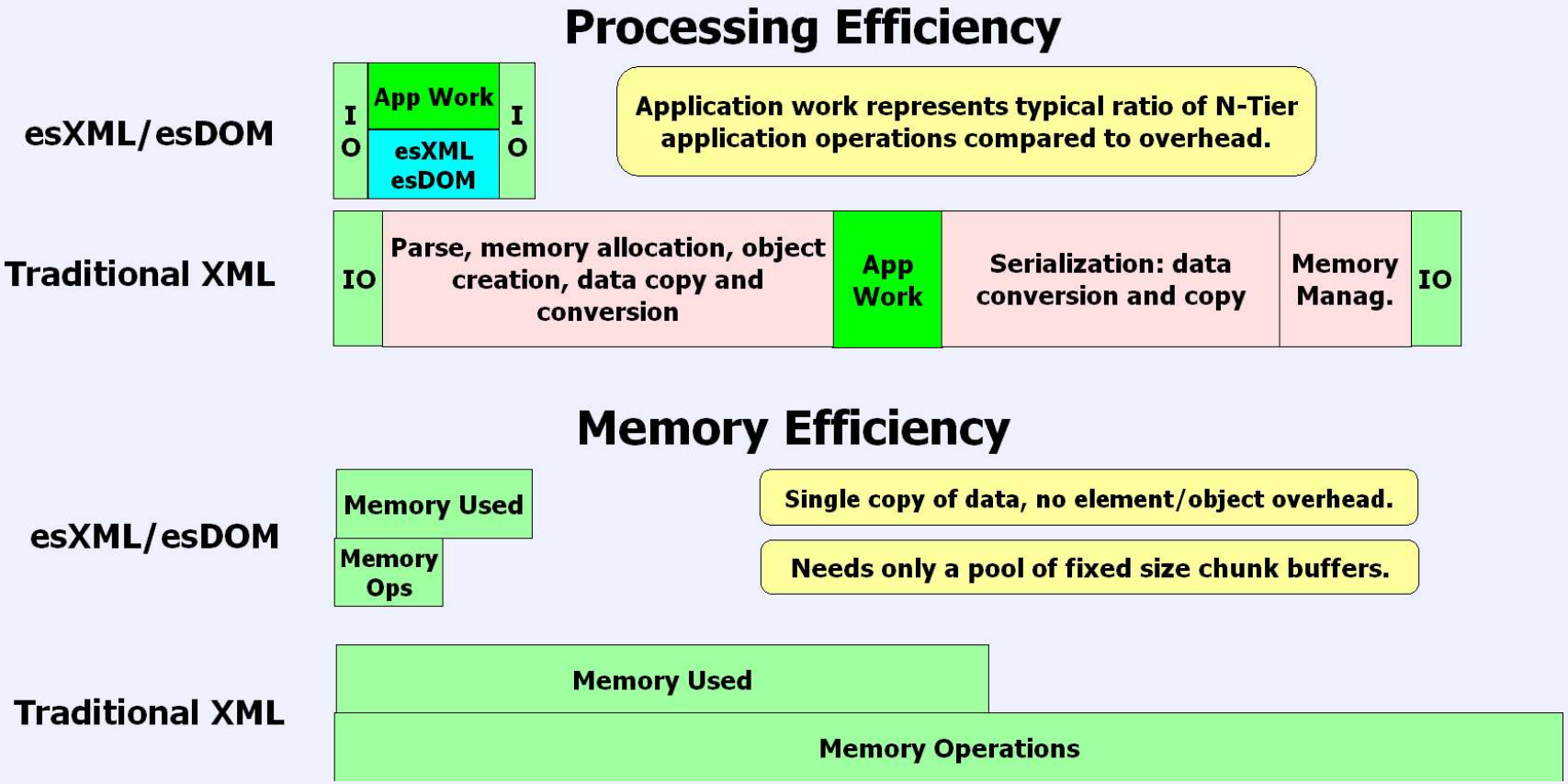
Solutions

- Portable, variably sized, binary structure
  - Token/length equivalent to XML 1.0
  - Supports fast depth or breadth first traversal
- Tunable Chunked buffers
- Elastic Memory
  - Support holes and chunk block management
  - Fast, low level delta/copy on write
- Virtual Pointers
  - Sticky, cheap, tracking references
  - Both ID/IDRef and precise offset

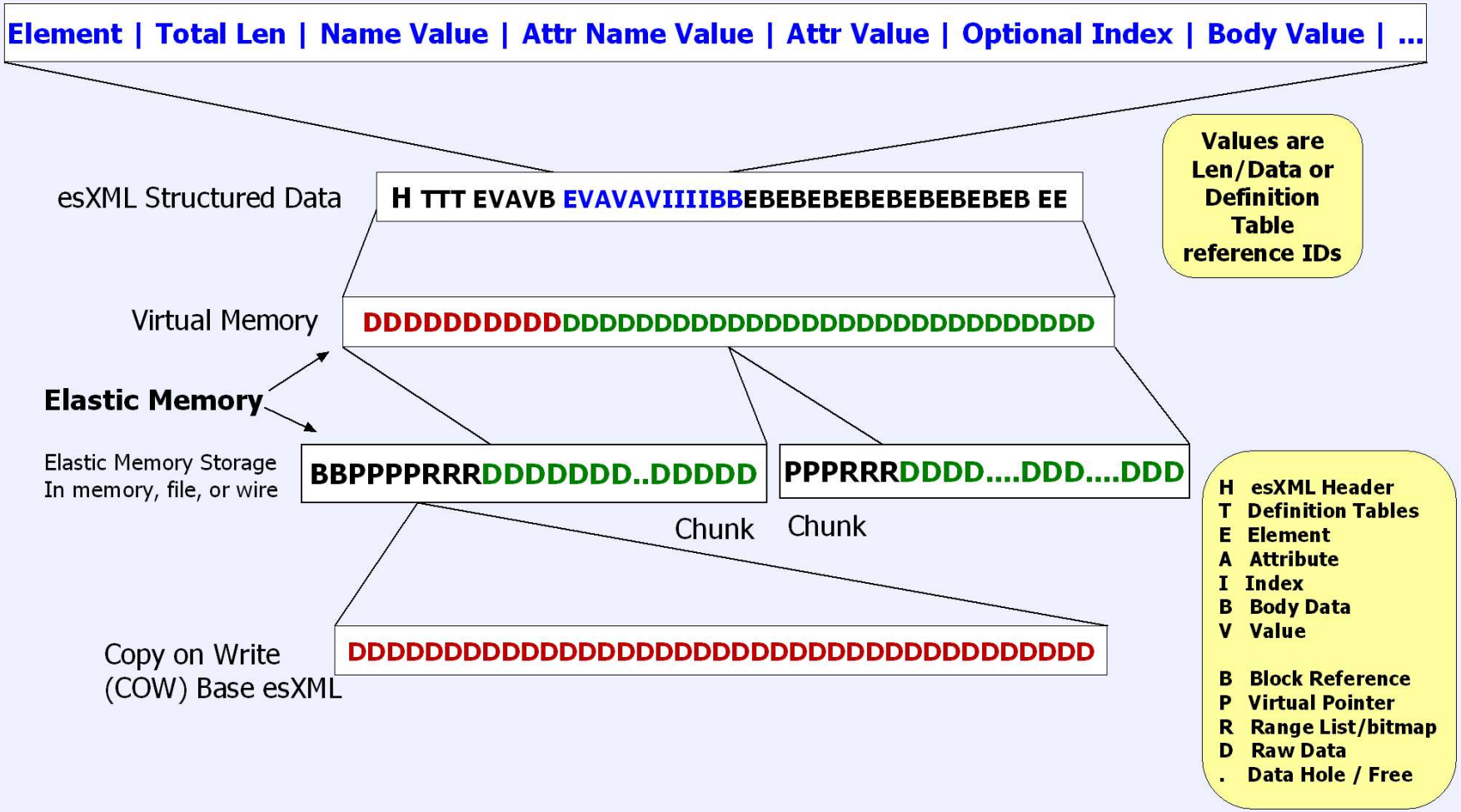
Solutions'

- Tags/values/bodies still presumed to be text
- Optional tokenization tables
  - Tags, attributes, values, body text
- Optional element indexes
- Object/doc GUID/UUID (possibly Tumblers)
- Possible standard compression in the future
  - For bandwidth pinch – IO/storage bound
  - Not for normal processing – processing bound

esXML vs. Traditional XML



esXML Structure



esXML Elastic Memory

- An intra-object virtual memory space
- Balances several requirements
  - Compactness, efficient in-place modification
  - Fast traversal, language/architecture portable
  - Supports extended semantics
- Chunked data – tracked storage blocks, splits when needed
- Free space, gap management – optional 'condensation'
- Each block may have any possible number of 'holes' / gaps
- Fast support of delta layering
- Manages Virtual Pointers (vptrs)
  - Indirect reference to elements/attributes/CDATA or offsets
  - Vptrs remain valid regardless of inserts/deletes
  - Does not require stable memory locations
  - Referred to externally by a handle

esXML Deltas

- Unifies implementation of many similar semantics through straightforward mechanism
  - Delta (change) layers, Copy on Write (COW)
  - Transactions, undo, savepoint/restart, session
  - Similar to Ted Nelson Xanadu derivative document versions
- A Delta is an esXML object that refers to GUID (or Tumbler) of base, read-only esXML object, recursively
- Handled by Elastic Memory layer, transparent to XML
- Many parallel deltas possible on the same base
- Can provide basis for shared context, message reuse
  - IDL-like common schema and header-compression-like message optimization

esXML Tables

- Optional optimization by tokenization
- Tag and attribute names
- Tag, attribute, CDATA values
- Ideal range of techniques in flux
  - Whole name/value occurrence
  - Index key-style compression
  - Phrase identification
  - Text pooling
  - Hash/index techniques
- Default to occurance at 'root' of esXML object
- Can occur unabridged at any element level
  - Needed for efficient insert/extract

esXML Elements and Data

- Type tokens
- Variable size length indicators
  - Efficient for small objects, capable of large
- Length based nesting, traversal
- Names / values, or table reference
- Binary data contained unencoded
- References, but not expression of vptrs
- Each element can have index
  - Level or subtree
- Each element can have token tables
  - Normally only at root and any indicated subroots

esDOM API

- Primary desired semantics are 3GL element and collection-style operations
- Fast Xpath reference, direct paths primary
- insert, append, set, get, presence, count, iterate
- create/factory, loadXML, saveXML
- loadesXML, saveesXML – very fast, just I/O
- getesXML – return a 'scoped' esXML reference
  - 'chroot' like confinement to subtree
  - Supports objects hierarchies by multiple scoped references to the same storage
- Intermediation – security, debugging, meta-processing
- Correct efficiency-errors in DOM

esDOM API Sample

```
/* Factory Methods */
    public esDOMInterface getesDOM(String path) throws esDOMException;
    public esDOMInterface clone(String path) throws esDOMException;

/* Append, insert, and set esDOM */
    public Node append(String path, esDOMInterface esdom) throws esDOMException;
    public Node insert(String path, esDOMInterface esdom) throws esDOMException;
    public Node set(String path, esDOMInterface esdom) throws esDOMException;

/* Append, Insert, Set data elements, converting non-string data types */
    public Node append(String path, Object data) throws esDOMException;
    public Node insert(String path, Object data) throws esDOMException;
    public Node set(String path, Object data) throws esDOMException;
```

esDOM API Sample'

```
/* Getter for String return value. */
public String get(String path) throws esDOMException;
public String get(String path, Object default) throws esDOMException;
/* Getters for data types other than String. */
public Integer getInteger(String path) throws esDOMException,
 NumberFormatException;
public int getint(String path) throws esDOMException, NumberFormatException;
public Integer getInteger(String path, Object default) throws esDOMException,
 NumberFormatException;
public int getint(String path, Object default) throws esDOMException,
 NumberFormatException;
/* Utility, collection methods */
public int elementCount(String path);
public boolean elementExists(String path);
public void remove(String path) throws esDOMException;
```

EsDOM Sample Usage

```
esDOM es = new esDOM(); /* new, from XML 1.0, from esXML */
es.append("/a", "test");
es.append("/b", 1);
es.append("/c", 1.1);
es.append("/d", false);
es.set("/b", 2);
es.insert("/aa", "test2");
es.insert("/a[2]/a", "test3");
es.append("/a", "test4");
String test = es.get("/a[3]"); /* "test4" */
test = es.get("/null", "default"); /* "default", could be value, NULL, or exception */
int count = es.elementCount("/a"); /* 3 */
esDOM esa = es.getesDOM("/a");
test = esa.get("/a"); /* "test3" */
```

Other Efforts

- Types:
  - Wire format compression
  - Serialization/parsing automation
  - Application specific ‘compiled’ encoding
  - XML Libraries: SAX/DOM
- SXML – Scheme S-expressions
- Bin-XML/BiM/BiX, WBXML, XER
- Millau, BOX, DSDL, LAML, StAX
- Xtalk, Xmill, ZML, XML short-tagging

Outlook

- Designed to be suitable for eventual standardization & wide use
- Implementation of current design in progress
  - Immediate use in large, high volume projects
  - C++, Java; later Python, perl, php4, etc.
  - Experiment with first-class support in Python
  - Open Source license (LGPL or other commercial-use-allowed license)
  - Restrictions of all material for companies/subsidiaries which have been convicted of anti-trust law violation
- Past implementations:
  - esDOM/Xerces (Java, PKI/validation intermediation)
  - esXML-lite (C++, MPEG4 player intermediate profile scene graphs)
- Soliciting comments, improvements, collaboration
- Goals and requirements are solid, current structure open to drastic change / competitive open development