Acceleration Techniques for XML Processors

Biswadeep Nag

Staff Engineer

Performance Engineering

XMLConference

2004

Sun microsystems

We make the net work.

XML is Everywhere

- Configuration files (web.xml, TurboTax)
- Office documents (StarOffice, OpenOffice)
- Web content (transformed to HTML)
- Web services and application integration

XML Advantages

- Cross platform
- Human readable
- Self-describing
- Extensible

XML Disadvantages

- Verbose
  - Binary encoding (fast infoset)
- Expensive to process
  - Parsing is fundamentally costly

XML Parsing

- Vs. programming language parsing
  - Similar, but simpler
- Lexical grammar defined by XML 1.x spec
  - Lex, Yacc equivalent
- Semantics (types, constraints, etc) defined by XML Schema
- Current parsers handcrafted in Java, C

XML Document

```
<invoice>
   <lineitem lineid="1">
      <item sku="1234">
         <name> Widget </name>
      </item>
      <qty> 10 </qty>
      <price currency="USD"> 100 </price>
   </lineitem>
</invoice>
```

SAX Events Generated

- StartElement(invoice)
- StartElement(lineitem, lineid=1)
- StartElement(item, sku=1234)
- StartElement(name)
- Characters(Widget)
- EndElement(name)
- EndElement(item)
- StartElement(qty)
- Characters (10)
- EndElement (qty)
- StartElement (price, currency=USD)
- Characters (100)
- EndElement (price)
- EndElement (lineitem)
- EndElement (invoice)

XML Parsing Types

- Streaming
  - Push – SAX (Simple API for XML)
  - Pull – StaX (Streaming API for XML)
  - Small memory footprint
  - Sequential access
- Tree-building
  - Language independent – DOM
  - Java specific – JAXB
  - More memory consumption
  - Random access, modification

XML Processing Technologies

- Java API for XML Processing (JAXP)
  - SAX, DOM, XSLT and in future StaX
  - Basic (W3C standard based) XML parsing
- Java Architecture for XML Binding (JAXB)
  - Converting XML to Java objects and vice versa
- Java API for XML-Based RPC (JAX-RPC)
  - Foundation of web services
- All part of Java Web Services Developer Pack (JWSDP)

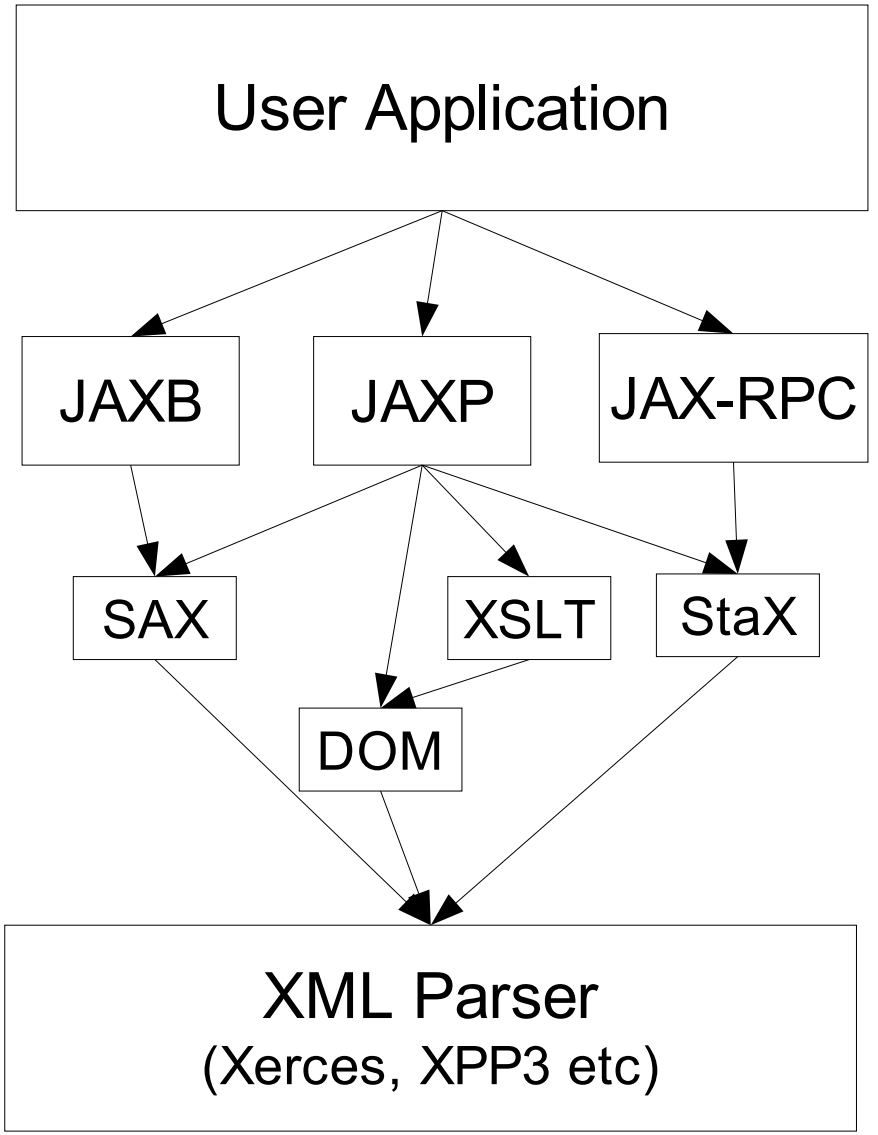
XML Acceleration Opportunity

- Diverse applications of XML
  - Parsing XML documents
  - Manipulating XML using Java programs
  - Remote web service calls via XML
- Yet one area that binds them all!
  - Basic XML parsing
- Current solutions focus on single application

XML Offload Engine (XOE)

- Handle common, routine aspects of XML processing, similar to:
  - TCP/IP offload engine (TOE)
  - SSL/Crypto accelerator
- Gains come from:
  - Acceleration – Specialized XML processing hardware
  - Offload – Using XOE processor instead of host cycles

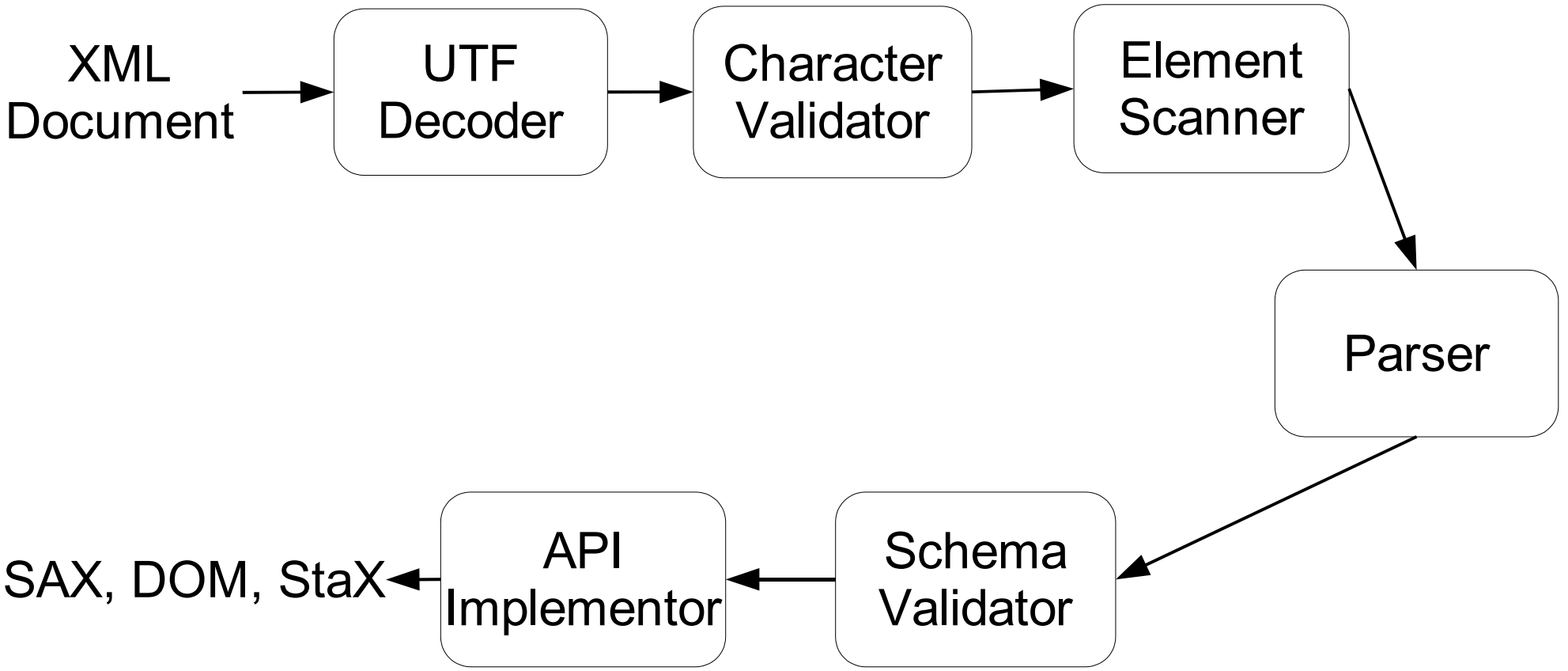
XML Application Stack



Java Standard API

XML Parsing API

XML Parser Architecture



XML Parser Details

Parser Module	Module Function
UTF Decoder	Decodes from UTF to Java Unicode chars
Char Validator	Checks that document has valid XML chars
Element Scanner	Tokenizes elements and attributes
Parser	Checks for well-formedness
Schema Validator	Validates XML document against XML schema
API Implementor	Implements W3C standard and Java APIs

XOE Characteristics

- Smart NIC
  - 10 Gigabit Ethernet
  - PCI-Express card
- Functions
  - TCP Termination (TOE)
  - HTTP Termination
  - SSL Acceleration
  - XML Acceleration
- Specialized Hardware
- Processor running software/microkernel

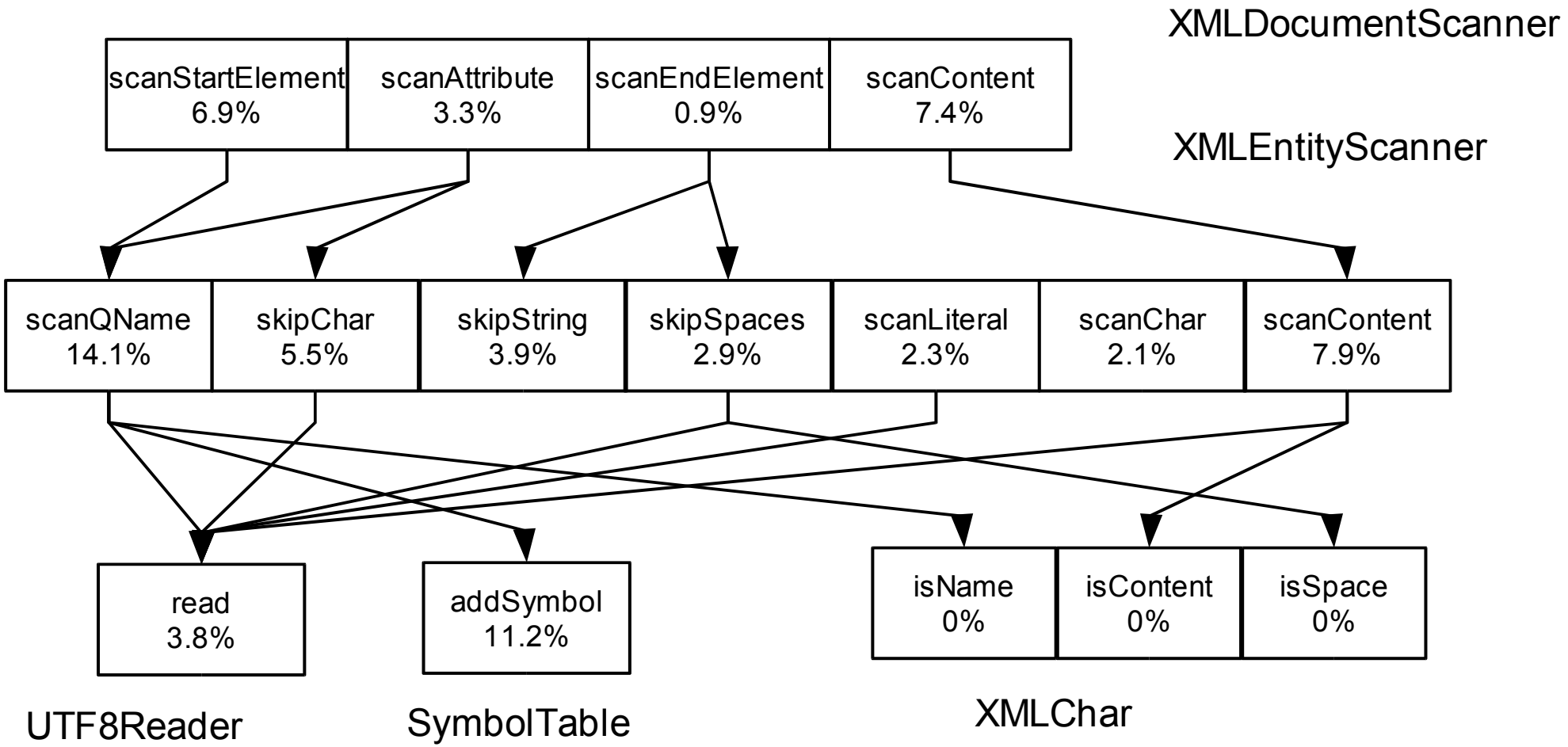
Design Considerations

- Address broadest use-cases
- XML & WS technologies still volatile
  - Avoid design obsolescence
- Keep most of the software stack unchanged
- Can fit into any host system
- Limited hardware design costs
- Use natural stratification of software stack

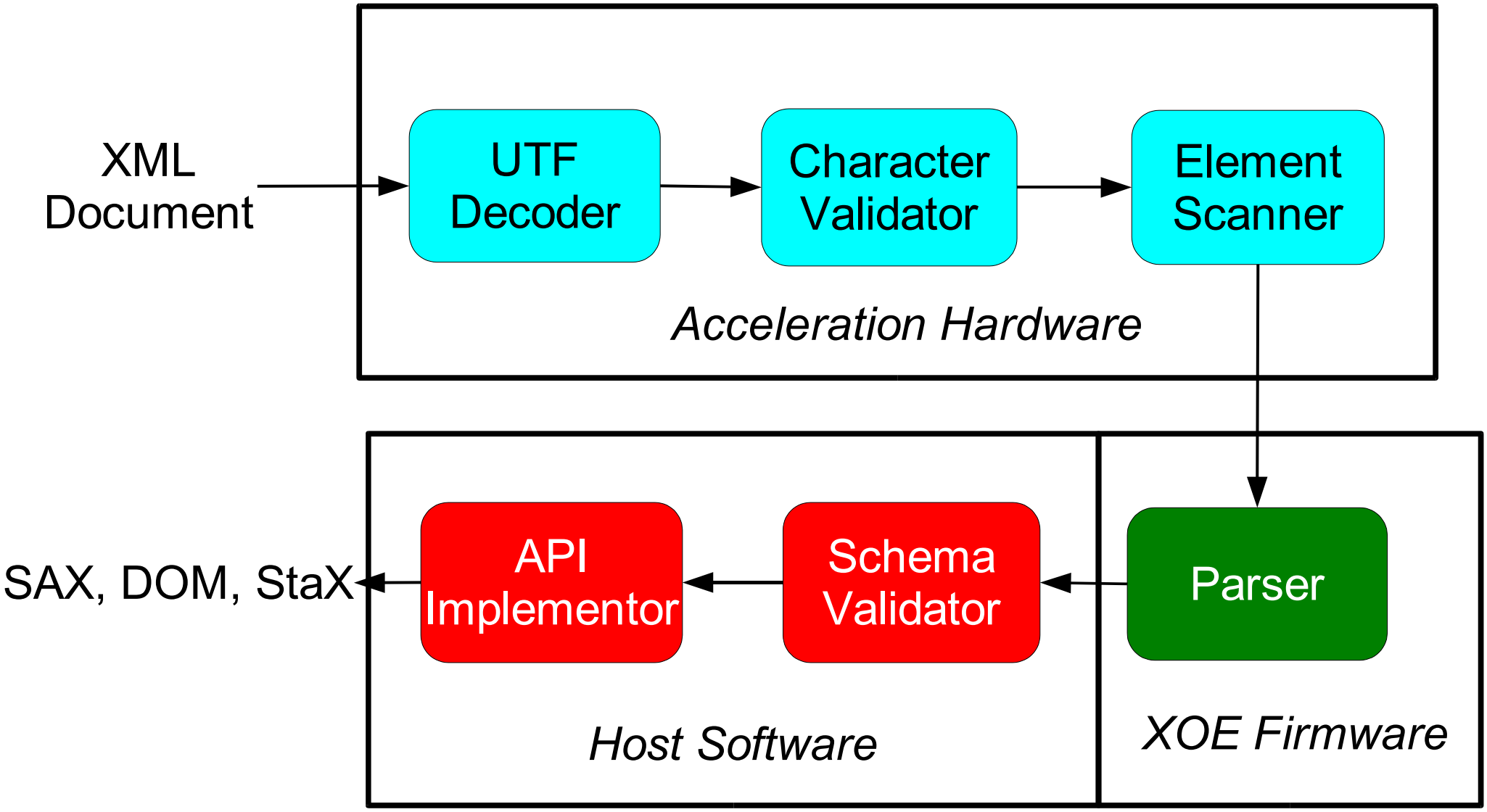
Design Decisions

- XOE Hardware
  - Low level parser functions
  - Relatively simple hardware
- XOE Firmware
  - High level functions (grammar checking)
  - Data structures requiring more memory
- Host Software
  - Applications APIs that may change
    JAXB, JAX-RPC, DOM, StaX, SAX
  - Steps that are done infrequently (Schema validation)

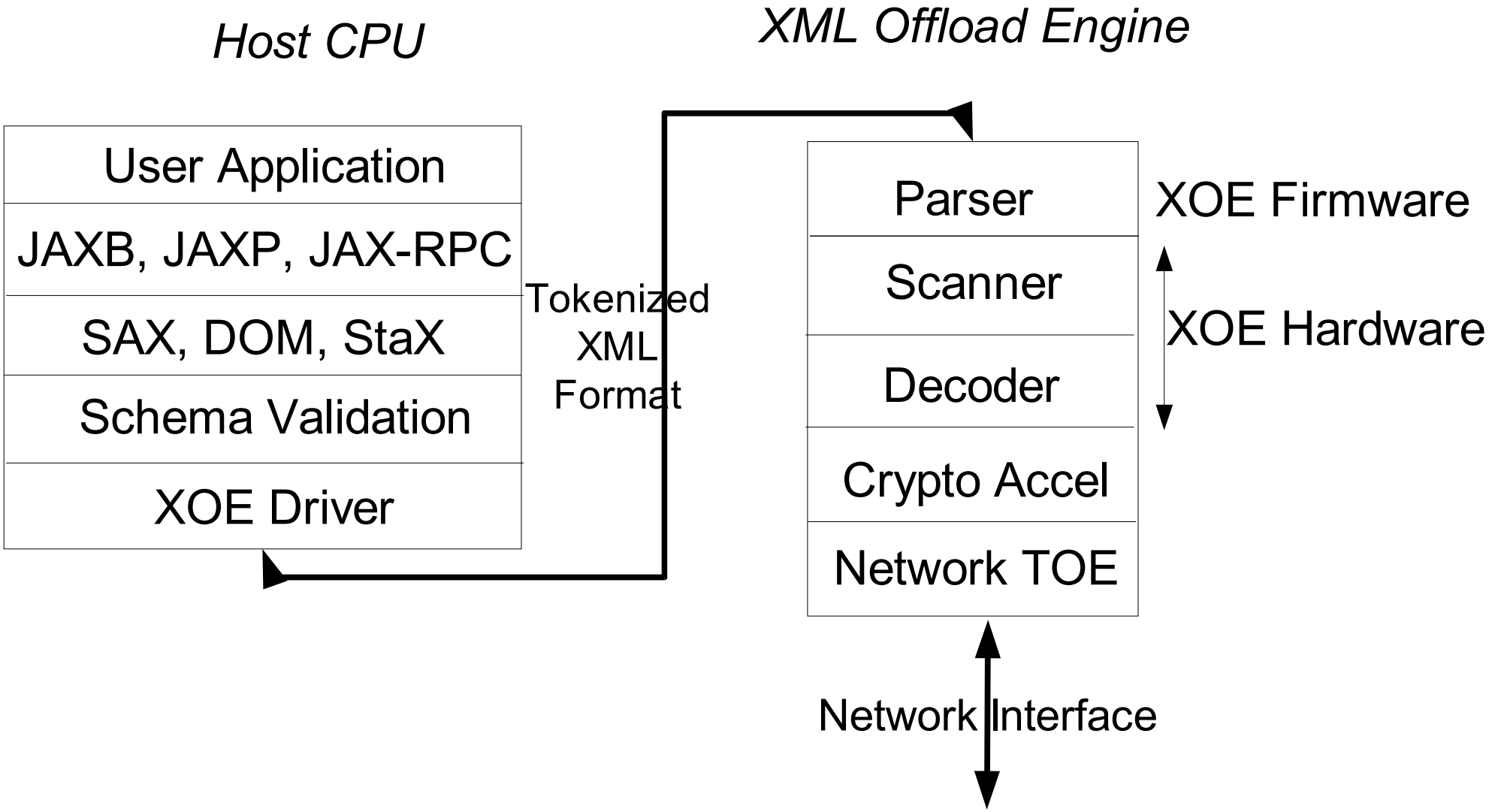
Xerces Call Graph



XML Offload Architecture



XML Acceleration Architecture



Hardware Design

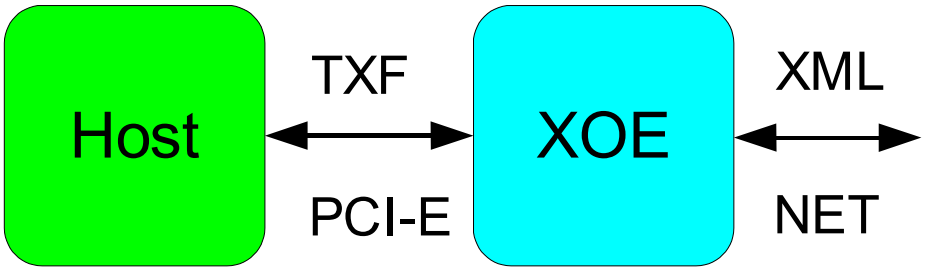
- Pipeline Architecture
  - DMA data from local memory
  - UTFDecode
  - Check valid XML chars
  - Tokenize into elements & attributes
  - Store in symbol table (hash implementation)
  - Compose tokenized format
  - DMA back to memory for sending to host

Firmware Design

- Control and direct XML accelerator h/w
- Implements XML grammar
- Well-formedness checking
  - Each begin tag matched with an end tag
  - Tags are properly nested
- Stack structure with ptrs to symbol table
- Compare endElement ptr to stack top

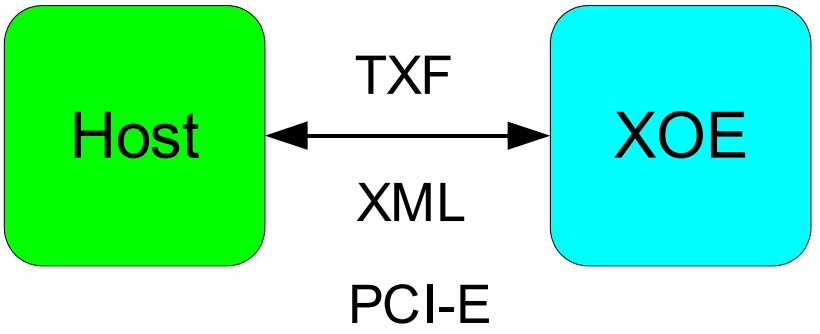
XOE Operating Mode 1

- XML from/to wire (web services)
  - Inline mode
  - SOAP / JAX-RPC
  - Requires TCP termination
  - HTTP termination (proxy)
  - Possibly SSL



XOE Operating Mode 2

- XML from/to host CPU
  - Coprocessor mode
  - Local file-system or generated by CPU
  - XML document processing (JAXP, JAXB, XSLT etc)



Data Transfer Performance

- PCI-Express interconnect
- Bulk / throughput transfers
- Avoid latency for short messages
- Buffering required
- Sharing between multiple streams
  - Saving parser state in firmware
  - Stopping at consistent points

Tokenized XML Format

- Close to binary format
- Used to pass data between host & XOE
- XML assumed to be valid & well-formed
- Cheaper to process than plain XML (> 2X)
- Compression of element and attr names
- Already converted to Java Unicode
- Fixed length strings

XML Document

```
<invoice>
   <lineitem lineid="1">
      <item sku="1234">
         <name> Widget </name>
      </item>
      <qty> 10 </qty>
      <price currency="USD"> 100 </price>
   </lineitem>
</invoice>
```

TXF Symbol Table

Symbol Id	Symbol
1	invoice
2	lineitem
3	lineid
4	item
5	sku
6	name
7	qty
8	price
9	currency

TXF Document

Code		Length	Value	
1	*StartElement*	-	1	*invoice*
1		-	2	*lineitem*
3	*AttrName*	-	3	*lineid*
4	*AttrValue*	1	1	
1		-	4	*item*
3		-	5	*sku*
4		4	1234	
1		-	6	*name*
5	*ElemContent*	6	Widget	
2	*EndElement*			*/name*
2				*/item*
1		-	7	*qty*
5		2	10	
2				*/qty*
1		-	8	*price*
3		-	9	*currency*
4		3	USD	
5		3	100	
2				*/price*
2				*/lineitem*
2				*/invoice*

XML Security

- Different from transport level (SSL)
- Encrypting parts of SOAP messages
- Digital signatures
- Requires Base 64 encoding

Example Encrypted Document

```
<invoice>
   <lineitem lineid="1">
      <item sku="1234">
         <name> Widget </name>
       </item>
       <qty> 10 </qty>
        <cipherData>
              hIOnMM9QhInpIe6D35cMW1prhJ
        </cipherData>
   </lineitem>
</invoice>
```

XWSS in Development

- Current implementation (Apache-based) uses non-standard APIs
- Future JSRs
  - JSR 105 – XML Signatures
  - JSR 106 – XML Encryption
- Very poor performance (10X slower)
- Requires iteration through XML parsing and crypto engine

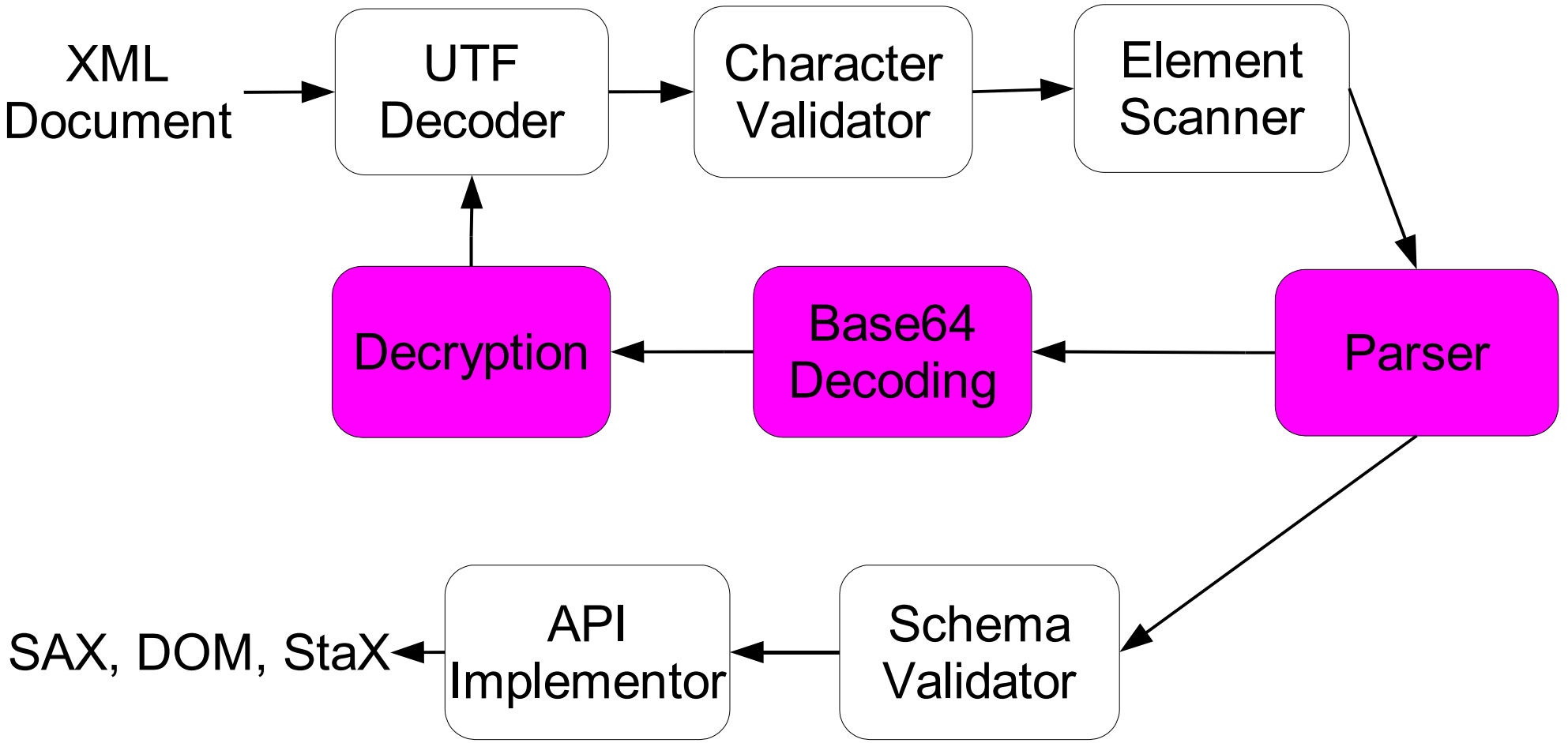
XWSS Opportunity

- Co-location of:
  - Crypto Accelerator
  - XML Parser
- Combination has major advantages
  - All processing happens in XOE
  - No round-trips to host
  - Transparent to host and application software
  - Huge performance gains

XML Security Details

- XML hardware tokenizes XML
- Does Base 64 decoding
- Firmware recognizes cipherData tag
- Calls crypto engine with encrypted data
- XML produced fed back to XML hardware
- Resulting tokenized XML combined with original document

XML Parser Architecture



Summary

- XML processing is expensive on general purpose CPUs.
- Using an XML Offload Engine
  - Many basic XML parsing functions accelerated by special-purpose hardware.
  - Generic, pre-processed, tokenized XML document produced by firmware.
  - Specific APIs implemented by host software
- Combining with crypto accelerator produces major benefits in XML security