

# **A Sequence-based Type-aware Interface for XML Processing**

**Jaakko Kangasharju**

**Tancred Lindholm**

**Helsinki Institute for Information Technology**

**February 21, 2005**

## Introduction

- XML messaging, i.e. machine-to-machine communication is data-oriented rather than document-oriented
  - Small documents that represent programming language records
  - Mixed content rare
  - XML only a data interchange format
- Use of DOM duplicates data in memory, SAX awkward for constructing records

# XML as Sequences

- XML document represented as a sequence of (extended) XmlPull events

Event	Content
DOCUMENT START	value
DOCUMENT END	none
PREFIX MAPPING	namespace, value
ELEMENT START	namespace, name
ELEMENT END	namespace, name
ATTRIBUTE	namespace, name, value
CONTENT	value
TYPED CONTENT	namespace, name, value

## The XAS Java API

- TypedXmlParser, TypedXmlSerializer: interfaces to serialized form
- Event: type discriminator, namespace, name, value
- Event sequence: interface providing a sequential view
  - Three implementations: list, stream, serializer
- XmlReader, XmlWriter: element-level access to XML
- Compatibility APIs for SAX and DOM

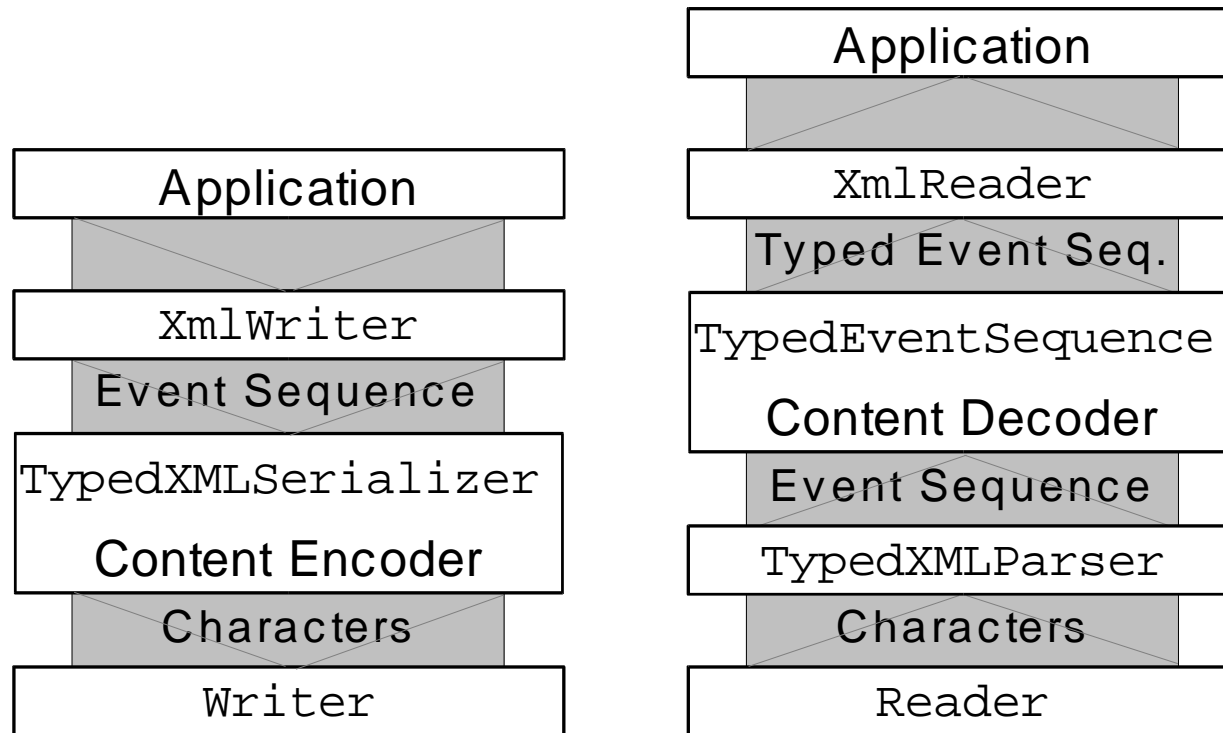
## Wrapped Event Sequences

- Event sequence interfaces allow wrapping of sequences, typically to elaborate more on input
- Filter sequence to delete unwanted events
- Transformed sequence to provide more general mappings
- A fragment of a document exists on only one wrapped level for lower memory use and proper streaming

## Handling of Typed Content

- Application-installable content encoders and decoders to handle arbitrary data types
- Primitive data must be handled at the lowest level
- Extended interfaces to support typed content in serializer and parser
- Mutually recursive `decode` to handle passed-in data type and generic `expect` to read type information from attributes

# The XAS Architecture



## Example: Compound Object Decoding

Encoded XML fragment:

```
↑<cp><i>1234</i><b>56</b></cp>
```

- Call `expect( "", "cp", reader )`

## Example: Compound Object Decoding

Encoded XML fragment:

```
<cp>↑<i>1234</i><b>56</b></cp>
```

- `expect` reads `SE( "", "cp" )` and type attribute, calls `Compound.decode`

`Compound.decode`



```
Integer i = (Integer) expect("", "i", reader);
```

```
Byte b = (Byte) expect("", "b", reader);
```

```
return new Compound(i, b);
```

## Example: Compound Object Decoding

Encoded XML fragment:

```
<cp><i>1234</i>↑<b>56</b></cp>
```

Compound.decode

```
Integer i = (Integer) expect("", "i", reader);
```

```
▷ i = 1234
```

```
Byte b = (Byte) expect("", "b", reader);
```

```
return new Compound(i, b);
```

## Example: Compound Object Decoding

Encoded XML fragment:

```
<cp><i>1234</i><b>56</b>↑</cp>
```

Compound.decode

```
Integer i = (Integer) expect("", "i", reader);
```

```
Byte b = (Byte) expect("", "b", reader);
```

```
▷ i = 1234, b = 56
```

```
return new Compound(i, b);
```

## Example: Compound Object Decoding

Encoded XML fragment:

```
<cp><i>1234</i><b>56</b></cp>↑
```

- `expect` reads over `EE( " ", "cp" )` and returns the object returned by `Compound.decode`

## Use Case: File System

- Synchronizing file system that serializes directory hierarchies as XML, originally used SAX
- XML used only because it is a standard way to represent structured data
- Possibilities for filtering and transformations simplified deserialization code
- Gradual migration possible using the SAX compatibility API

## Conclusions

- A sequential view with type awareness is very natural for many data-oriented XML applications
- Use of the event sequences as an in-memory representation with indexing on top could be competitive with existing tree-based APIs
- Simple layered design should help in maintaining and extending the API in the future

**Thank You!**