What's All This About Different Kinds of XML Parsers?

How Do XML Parsers Differ?

A. By the relationship between the XML parser and its client:

1. A **DOM** parser returns a "tree" representation of the XML document.

2. A **Push** parser calls client's methods with XML events.

3. A **Pull** parser returns XML events to a client on request.

B. By how data is returned:

4. Data-copying XML parsers copy all the information in the parsed XML document into objects, returned to the client.

5. In-situ XML parsers, as much as possible, indicate where data was found in the parsed XML document.

C. By what information is returned to the client:

6. Element structure and properties, and data content information.

7. Internal entity location and value information.

8. External entity information, and the ability of the client to participate in entity resolution.

……

There aren't necessarily clean boundaries between the different kinds of XML parsers:

9. Even the best in-situ parsers have to provide some information using objects.

10. Most pull model parsers revert to the push model when accessing external entity data.

XML Parsers: Power vs. History

Relationship between parsers:

1. Given either a push or a pull XML parser, you can easily build a DOM parser. Push or pull will give you XML events, and you just create DOM tree nodes to represent and link those events.

2. Given a pull XML parser, you can easily build a push parser: get the events from the pull parser and call the appropriate methods in the push fashion.

In both cases, the other way round doesn't really work. You can "fake" a pull or push parser based on a DOM parser, but you lose the low-footprint serial parsing of both those models. You can't really build a pull parser using a push parser (in the absence of coroutines, but that's another story.)

A bit of history:

In the realm of generally available parsers, SAX and SAX-like push model parsers came first. Using an object-oriented programming language, a SAX-like parser is the easiest to build.

DOM and DOM-like parsers came next. Given a SAX-like parser, a DOM or DOM-like parser is easy to build. Without a push parser, you have to reproduce all the logic of a pull parser within the DOM parser.

The next step is pull-model parsers. Pull model parsers are hard to build, because the current state of parsing, held in the tree in a DOM parser, held in the program state in a SAX parser, must be completely saved away between calls from the pulling client.

XML Parsers: Power vs. History

History vs. Power

It's not surprising that more powerful parsing models have come later on — that's the conventional direction of "progress". On the other hand, had we had freely-available pull parsers to start with, history would have been different.

In-situ vs. data-copying

In-situ is more powerful than data-copying: if the client doesn't care where the data is, either will do; if the client cares, only in-situ will do.

Historically: data-copying first, and in-situ later.

Anything next?

Pull-everything, in-situ XML parsers are about as far as you can go — they can do anything. So we're getting close to the end of this bit of history.

What's missing?

The current pull-model parsers only partially implement a pull-everything interface, and most are data-copying.

Why pull-everything? Extending the pull interface to entity-side events and giving the client full control over external entity resolution using a pull model puts fewer constraints on the client and widens the range of application of an XML parser.

In-situ parsing is a good fit for small-footprint devices (see Antonio J. Sierra's paper), but has other applications. For example, it allows you to use XML as the internal "native" format of data in an XML-aware text editor. Again, the idea is put the choices in the client's hands.