Chapter VIII
Native XML Programming: Make Your Tags Active

Philippe Poulard
The French National Institute for Research in Computer Science and Control (INRIA), France

ABSTRACT

XML engines are usually designed to solve a single class of problems: transformations of XML structures, validations of XML instances, Web publishing, and so forth. As the relevant operations or declarations are described with XML vocabularies, their respective engines have to face similar issues such as unmarshalling, and at runtime data handling. In order to address such issues, the author proposes an innovative and reliable XPath-based framework, active tags, that unifies XML technologies in a coherent system where various XML languages can cooperate. In this chapter’s approach, the authors focus on its type system that enhances the XML Data Model, specifically by allowing XPath expressions to be applied on non-XML objects, and on the ability of the engine to mix declarative languages with imperative constructs. This valuable feature is illustrated with the Active Schema Language, which allows the building of dynamic-content models.

INTRODUCTION

XML instances usually fall into one of the two following categories: XML documents and XML data structures; of course, hybrid instances that consist of a combination of these two categories can also be considered. However, a third category is worth mentioning: XML languages designed for processing purposes, such as XSLT (W3C, 1999b). Actually, since the emergence of XML technologies, numerous processing-oriented XML languages have been invented for specific purposes.

In this chapter, we will discuss native XML programming, that is to say, XML languages intended to be processed by a specific engine in order to solve a class of problem. This includes
templatizing languages that sometimes are not strictly spoken XML languages (they are not « well-formed » according to the XML specification) but aim to reach similar objectives, and declarative-oriented languages that are certainly the most specialized languages. Among these languages, we have:

- Those that are tightly coupled to a programming language such as Java (Ant, Jelly),
- Those that are rather language-independent (SCXML, XProc),
- Those that deal only with Web problematics (JSP/JSTL/Taglib),
- Pure declarative-oriented languages (XML Catalogs, J2EE deployment descriptors, schemata),
- Those that use technologies related to XML such as XPath (XSLT, Active Tags).

In fact, apart from the capabilities of the tools mentioned above, native XML programming is fundamental to process integration, in the same way that native XML querying (XQuery) is fundamental to data extraction and remodeling. As XQuery does not address natively Web problematics (although some proprietary extension of XQuery-based systems can deal with them), native XML programming acts as the glue between the HTTP server and the XQuery query. More generally, native XML programming can stand between the different components of an XML database application.

XML programming has the following advantages.

- First of all, we agree that whatever the nature of the underlying data sources, the best method for efficient data integration is to supply them in a common format: XML. Actually, every non-XML structure only has to be mapped to an XML representation. We will see later that we are not necessarily talking about the tag representation (markups in a file) of XML, but rather about the data model representation of XML (XDM). In any case, XML-related technologies such as XPath are likely to be used for handling these data. Conversely, tools such as Ant and Jelly that rely on the unified expression language (UEL) are very far removed from XML problematics.
- Secondly, due to its extensibility, XML is the best candidate for designing special purpose languages, including declarative languages: arbitrary complex processes can be expressed with very few tags, provided that they are backed by an appropriate implementation.
- Finally, as with XSLT, dealing with an XML language allows the designer of such a language to build XML structures by mixing XML literals among XML instructions (the so-called « Active Tags »), as do template languages.

The objective of this chapter is to give an overview of open issues in the field of native XML programming, and demonstrate that these issues can be addressed by a unification of XML technologies that promote the cooperation of multiple XML languages.

After recalling the origins that led to the design of XML-based programs, and drawing up a comparison table between different processing-oriented XML languages, we will introduce Active Tags (Poulard 2006; 2007), a general-purpose system for native XML programming that borrows the best from many technologies and tools like those mentioned and proposes an innovative and reliable framework that can host any new processing-oriented XML language. We will show with several examples that are runnable in RefleX (the reference implementation of Active Tags in Java) how to use Active Tags within Web applications or as standalone applications in pipelines, how to use several tag libraries simultaneously, and how to expose a tag as a macro-tag.
Related Content

Making Programming Exercises Interoperable with PExIL
www.igi-global.com/chapter/making-programming-exercises-interoperable-pexil/73172?camid=4v1a

Temporal OCL: Meeting Specification Demands for Business Components
www.igi-global.com/chapter/temporal-ocl-meeting-specification-demands/30577?camid=4v1a

XML and LDAP Integration: Issues and Trends
www.igi-global.com/chapter/xml-ldap-integration/27776?camid=4v1a

UML- and XML-Based Change Process and Data Model Definition for Product Evolution
Ping Jiang, Quentin Mair, Julian Newman and Josie Huang (2005). Software Evolution with UML and XML (pp. 190-221).
www.igi-global.com/chapter/uml-xml-based-change-process/29614?camid=4v1a