Programming for XML

Daniela Florescu
Oracle
dana.florescu@oracle.com

Donald Kossmann
ETH Zurich
kossmann@inf.ethz.ch

1. BACKGROUND AND MOTIVATION

There are many emerging applications for XML. Examples are inter-application data exchange (e.g., Web services), streaming data (e.g., sensor networks), blogs (e.g., RSS and Atom), meta-data management (e.g., configuration files), and Scientific data (e.g., linguistic and protein databases). There are several reasons for the use of XML in these applications. XML is vendor and platform independent. XML dissociates data from schema which made XML attractive for new applications that require flexible schemas. Last but not least, there are a large number of tools; among others, editors for XML and XML schema (e.g., XML Spy or Stylus Studio), transformation languages (e.g., XSLT and XQuery), XML wrapper generators, and design tools (e.g., UML to XML converters).

Although there are many tools availalbe, an open question is the right programming paradigm to process XML data. Today, the most popular solutions are based on extensions to existing programming languages (e.g., Java, Python or PHP) with XML-specific libraries and APIs. Such libraries either represent the XML data as a virtual tree, or they read the XML data in a streaming (push or pull) fashion. This approach has the obvious problems that arise from the impedance mismatch between the XML type system and the type system of the host language. Moreover, the code written in such programming languages cannot be (easily) optimized using traditional techniques; good performance, scalability, and service-level guarantees is difficult to achieve for such programs on large datasets. Recently, several proposals for new programming languages have been made in both industry and the research community. One prominent example is Microsoft's XLinQ language. Another prominent example of XML processing in Web-based applications is AJAX (Asynchronous Java Programming with XML). In academia, XL, XStatic, Links, and several other languages have been proposed. All these solutions follow different philosophies and address critical design questions in different ways.

SIGMOD 2006, June 27–29, 2006, Chicago, Illinois, USA.
Copyright 2006 ACM 1-59593-256-9/06/0006 ...$5.00.

This tutorial gives an overview of the current generation of programming languages for data-intensive XML applications. Furthermore, this tutorial compares the possible solutions based on a few comparative practical criteria. The tutorial shows how each solution addresses the design questions in different ways and gives the tradeoffs in terms of capabilities and optimizability of these languages are.

2. OVERVIEW OF TUTORIAL

The outline of the tutorial is as follows:

1. Introduction to XML: why? where? when? how?
2. The need for processing XML data
3. Alternative solutions to XML data processing
   (a) A comparative study of XML APIs (tree vs. stream, push vs. pull, read-only vs. updatable, Infoset vs. PSVI)
   (b) Adding APIs to existing programming languages
   (c) Mapping XML into structures of existing programming languages (e.g., EntityBeans)
   (d) Extending existing programming languages with XML support
      i. Extending scripting languages (e.g., JavaScript, PhP)
      ii. AJAX
      iii. Extending C# with XML support (XLinQ)
      iv. Extending SQL with XML support (SQL/XML)
   (e) Extending XSLT and XQuery with side-effects and procedural support
4. Comparative study of alternative approaches
   (a) Functional completeness
   (b) Productivity improvement
   (c) Potential for automatic optimization
5. Open issues
   (a) Declarative vs. imperative XML programming
   (b) The usefulness of static type systems
   (c) Integrating semantic search, classification and inference in traditional query and programming languages
   (d) Support for asynchronous communication, continuous queries and push data
6. Conclusion