INFORMATION SYSTEMS MODELS IN HIGHER EDUCATION

Cristina Mendonça

ABSTRACT
This paper intends to contribute to a better understanding of the process through which information resource, information technology, and organisation actors can contribute to the performance and quality of higher education institutions. Conceptual models will be presented and discussed.

KEY WORDS: APIs, Image processing, Motion capturing, Higher Education

1.INTRODUCTION

Institutions of Higher Education have to develop a viable strategic information resources management system and create some operating principles, which provide basis for consistent decision-making and resource allocation, and support efforts to reach their social mission and objectives. APIs integrates several concepts, such as competence, skill, expertise, and literacy. It should consider the characteristics of constructive learners, face attitudes regarding the usefulness of IT as an intensive effort to improve quality management Systems in the organisation, attempt to incorporate significant differences between prior knowledge, meta-cognition, motivation and the "learning style."

We specify flexible structures and we model quality of service goals, to allow comparing of the goal fulfilments along alternative execution options. We suggest a method to improve quality of service results for image processing tasks by controlling different execution options.

This is the most promising approach to realize seamless coupling between virtual environments and real world. The model is based on doing seamless mapping of human motion in the real world into the virtual environments. Motion capturing by computer vision techniques is applicable for such purposes (Antitiroiko, 2001).

We hope that these three models raise issues for an improvement of quality in Education, particularly in higher education, which is by the moment our aim.

We will discuss:

- A conceptual model for the performance optimisation of higher education as a function of Applied Information Systems (APIs).
- The quality of the image processing tasks by modelling them as flexible workflow processes.
- A virtual object manipulation system using a 3-D human motion sensing without physical restrictions.

2. APIs MODELS

2.1 PUSH MODEL APIs

In recent times the landscape of APIs and techniques for processing XML has been reinvented as developers and designers learn from their experiences and some past mistakes. APIs such as DOM and SAX, which used to be the bread and butter of XML APIs, are giving way to new models of examining and processing XML. Although some of these techniques have become widespread among developers who primarily work with XML, they are still unknown to most developers. Nothing highlights this better than a recent article by Tim Bray, one of the co-inventors of XML, entitled XML is too Hard for Programmers and the subsequent responses on Slashdot.

In a Push model the XML producer (typically an XML parser) controls the pace of the application and informs the XML consumer when certain events occur. The classic example of this is the SAX API, where the XML consumer registers call - backs with the SAX parser, which invokes the call - backs as various parts of the XML document are seen.

The primary advantage of push model APIs when processing XML is that the entire XML document does not need to be stored in memory only the information about the node currently being processed is needed. This makes it possible to process large XML documents which can range from several megabytes to a few gigabytes in size without incurring massive memory costs to the application. However it also means, that certain context and state information such as the parents of the current node, or its depth in the XML tree, must be tracked by the programmer.

Another issue with push model parsers is that many developers find call - backs to be an unintuitive way to control program flow. Tim Bray described call - backs as being non-idiomatic and awkward when used from his programming language of choice.

The following code sample uses the SAX API in the Apache Xerces Parser to display the content of the title and author elements of books that have the on-loan attribute set.

It shall be noted that, to register call - backs one needs to create a class devoted to handling events from the SAX parser, either by implementing the Content Handler interface or extending the Default Handler class.

2.2 PULL MODEL APIs

During pull model processing, the consumer of XML controls the program flow by requesting events from the XML producer as needed instead of waiting on events to be sent to it. This is very similar to the pseudo - code described in Tim Bray's post as the typical text - processing idiom. Like push model parsers, pull model XML parsers operate in a forward - only, streaming fashion while only showing information about a single node at any given time. This makes pull - based processing of XML as memory efficient as push - based processing but with a programming model that is more familiar to the average programmer.

Two notable pull model XML parsers are the .NET Framework's XmlReader class and the Common API for XML Pull Parsing. Programming using both APIs is fairly similar; one creates a loop that continually reads from the XML document until the end of the document is reached but acts solely open items of interest as they are seen.

The following code sample uses the .NET Framework's XmlTextReader class to display the contents of the title and author elements of books that have the on-loan attribute set.

Pull model parsers typically do not require a specialized class for handling XML processing since there is no requirement to implement specific interfaces or subclass certain classes for the purpose of registering call - backs. Also the need to explicitly track application states using Boolean flags and similar variables is significantly reduced when using a pull model parser.

2.3 TREE MODEL APIs

A tree-based API is an object model that represents an XML document as a tree of nodes. The object model consists of objects that map to various concepts from the XML 1.0 recommendation such as elements, attributes, processing instructions and comments. Such APIs provide mechanisms for loading, saving, accessing, querying, modifying, and deleting nodes from an XML document. The canonical example of a tree model API for processing XML is the W3C XML Document Object Model (DOM), which has inspired various programming language specific variations including JDOM and PyXML, for Java and Python respectively.

Typically tree model APIs load the entire XML document into memory, and thus do not limit users to forward-only access of the XML data. This prevents traditional tree model APIs from being used in situations where large XML documents have to be processed. Although it is possible to build optimised tree model APIs that only load portions of an XML document as needed, such APIs are not in widespread usage.

The following code sample uses the Apache Xerces DOM API to display the contents of the title and author elements of books that have the on-loan attribute set.

There are numerous advantages to this approach. First, the memory footprint of XML data can be reduced because information isn't being stored as nodes and textual data but as classes and programming language primitives. A DOM node that represents a <foo> element that contains a numeric value as text is more memory intensive than its counterpart foo class with an integer field. In particular, the memory footprint is better if you are able to turn lots of leaf values into primitive valued fields, or if you can do away with parent and sibling pointers. Second, it is more convenient to perform calculations on certain types of data such as numbers or dates as native programming language constructs than it is to interact with them as string values stored in nodes. But, third, the most compelling argument is the improved ease of use. It's no longer necessary to navigate the XML tree to access the information but instead one can simply access data as fields and properties of an object.

Object to XML mapping technologies have certain limitations that prevent them from replacing traditional methods for accessing XML data. Most of these technologies cannot represent all the information in an XML document with full fidelity. Many do not preserve processing instructions and comments. Similarly mixed content is problematic to map to objects since the tendency is to map element and attribute nodes to objects and text nodes to the values of fields or properties in said objects. Although the order of elements is significant in an XML document, this typically cannot be enforced on objects. Most object oriented languages do not have a way of expressing that in a book class the *title* field precedes the *author* field, although one could use ordered collections to get around this problem.

It seems natural that one would process XML using a language that is designed for processing XML as opposed to going through traditional programming languages. For performing complex operations on XML data, all of the aforementioned techniques suffer from either being too cumbersome, require too many lines of code, or do not handle all of XML. In such cases, the wise decision is to go with a language, which natively understands how to process XML to do the heavy lifting and invoke that from the target programming language. Examples of languages specifically designed for processing and manipulating XML include XPath, XQuery, XSLT, and Xtatic.

There are various sites where one can try out sample XQuery expressions, including QEXO XQuery Sandbox and Microsoft's XQuery demo site.

3. SERVICE-ORIENTED ARCHITECTURE (SOA)

A service-oriented architecture is essentially a collection of services. These services communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity. Some means of connecting services to each other is needed.

Service-oriented architectures are not a new thing. The first service-oriented architecture for many people in the past was with the use DCOM or Object Request Brokers (ORBs) based on the CORBA specification. For more on DCOM and CORBA, see Prior service-oriented architectures (new window).

If a service-oriented architecture is to be effective, we need a clear understanding of the term service. A service is a function that is well - defined, self - contained, and does not depend on the context or state of other services. See Service (new window).

The technology of Web services (new window) is the most likely connection technology of service-oriented architectures. Web services essentially use XML (new window) to create a robust connection.

A basic service-oriented architecture diagram can show a service consumer at the right sending a service request message to a service provider at the left. The service provider returns a response message to the service consumer. The request and subsequent response connections are defined in some way that is understandable to both the service consumer and service provider. How those connections are defined is explained in Web Services explained (new window). A service provider can also be a service consumer.

4. AN AVATAR MOTION CONTROL BY BODY POSTURES

The goal is to do seamless mapping of human motion in the real world into virtual environments. We hope that the idea of direct human motion sensing will be used on future interfaces. With the aim of making computing systems suited for users, a computer vision based avatar motion control was developed. The human motion sensing is based on skin - colour blob tracking. The method can generate realistic avatar motion from the sensing data. The framework is addressed to use virtual scene context as a priori knowledge. It is assumed that virtual objects in virtual environments can afford avatar's action, that is, the virtual environments provide action information for the avatar. Avatar's motion is controlled, based on simulating the idea of affording extended into the virtual environments.

A 3D representation for visualizing large software systems is developed. The origins of this representation can be directly traced to the *SeeSoft* metaphor. This work extends these visualization mechanisms by utilizing the third dimension, texture, abstraction mechanism, and by supporting new manipulation techniques and user interfaces. By utilizing a 3D representation we can better represent higher dimensional data than previous 2D views.

These prototype tools and its basic functionality have applications to particular software engineering tasks, which contribute to a quality approach for I formation System Models in Higher Education.

BIBLIOGRAPHY

Antitiroiko, A. V. et al. R. (2001): “Information society competencies of managers: conceptual considerations”, *In search of a human-centred information society. Edited by E. Pantzar, R. Savolainen & P. Tynjälä, Tampere*