Tutorial 3: ArgoUML and XML

ECSE 321 – Intro to Software Engineering
Electrical and Computer Engineering

McGill University

Winter 2009

Contents

ArgoUML

What is ArgoUML?

ArgoUML is a UML diagramming application written in Java and released under the open source BSD License. By virtue of being a Java application, it is available on any platform supported by Java.

Where and how to download?

You can get the most recent version of ArgoUML on their website at:

http://argouml.tigris.org/

From there you can run it through a Java Web Start (ie run it from your browser) or simply download the zip file directly and have a copy on your computer (recommended).

To run the application simply extract the downloaded zip file and run the *argouml.jar* file. It is recommended to always keep these numerous jar and other type of files in the same folder to avoid referencing errors when using the software.

Why use ArgoUML?

During the course of your project, you will be required to draw up numerous types of UML diagrams. From Use Case diagrams (see tutorial 5) to Class and Sequence diagrams, all of which help you and the client get a better grasp on where your project is headed and how every fits together. The primary reasons for using a specialized UML diagramming tool rather than paint are many:

- Flexible structure
- Standardized elements (arrows, actors, boxes, etc)
- Fancier interface

- Easily portable

ArgoUML is the suggested tool to be used for this course mainly because it is free and easy to use.

Available Features

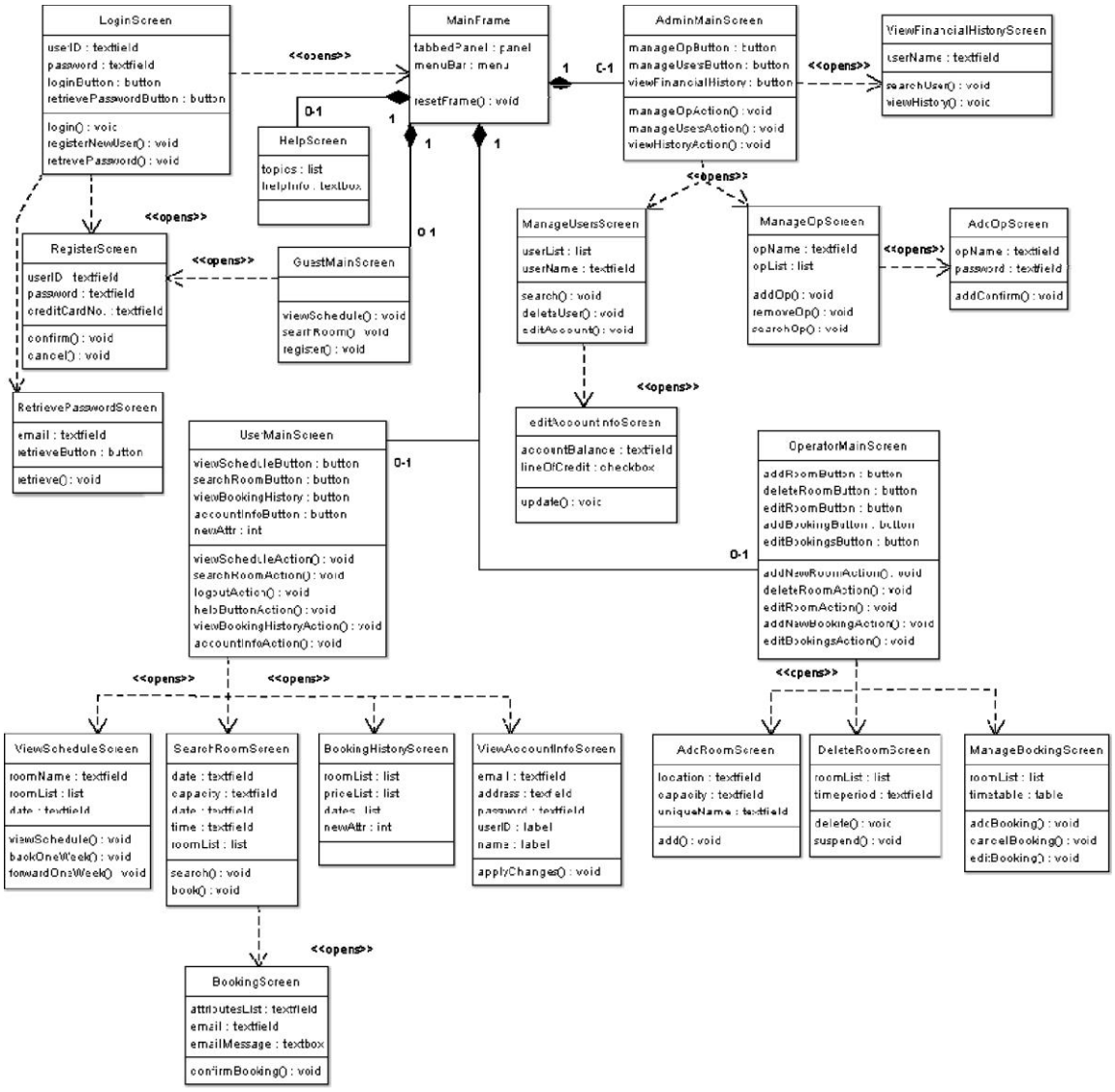
- All 9 UML 1.4 Diagrams supported
    - Most all types of UML diagrams are supported, all those used within the scope of this course are supported (except one)
- Platform Independent: Java 1.4+
    - You will definitely recognize the Java Look and Feel
- Export Diagrams as GIF, PNG, PS, EPS, PGML and SVG
    - Easy to past into design documents as pictures
- Available in ten languages - EN, EN-GB, DE, ES, IT, RU, FR, NB, PT, ZH
- Advanced diagram editing and Zoom
- Built in design critics provide unobtrusive review of design and suggestions for improvements
- Extensible modules interface
- Forward Engineering (code generation supports C++ and C#, Java, PHP4, PHP5, Python, Ruby and, with less mature modules, Ada, Delphi and SQL)
    - From well designed class diagrams can generate base code in Java, setting up classes and interactions between them automatically (time saving feature)
- Reverse Engineering / Jar/class file Import
    - From an existing project, can generate a rough class diagram of interactions (not very good)

Shortcomings

- No undo feature.
    - Very frustrating, but must be contended with
- Sometimes models cannot be re-opened. Incrementally save-as-copy.
- Sequence diagrams not functional
    - Needed for this class, whoever these are the easiest diagram to work out by hand

GUI Class Diagram Example

RMSS – Fall 2007



XML

What is XML?

The Extensible Markup Language (XML) is a general-purpose markup language. It is classified as an extensible language because it allows its users to define their own tags. Its primary purpose is to facilitate the sharing of structured data across different information systems, particularly via the Internet. It is used both to encode documents and serialize data.

For the context of this course it offers a simple structure to act as a database. From the sample below, it is easy to see that XML is structured like a tree, which allows for easy parsing when programming (more on that later)

Sample File

Booking

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- edited with XMLSpy v2007 (http://www.altova.com) by Me (None) -->
<UserBookingsDatabase>
	<bvarba />
	<jess />
	<mtavar>
		<event>
			<name>Testycool</name>
			<type>Anniversary</type>
			<organizer>Michael Tavares</organizer>
			<date>07/12/2006</date>
			<start>0</start>
			<end>1</end>
			<emails>michaeltavares321@hotmail.com;</emails>
			<price>8.0</price> <room>Champion_Lecture</room>

		</event>
	</mtavar>
</UserBookingsDatabase>
```

Advantages of XML

- It is text-based.
- It can represent the most general computer science data structures: records, lists and trees.
- The strict syntax and parsing requirements make the necessary parsing algorithms extremely simple, efficient, and consistent.
- XML is heavily used as a format for document storage and processing, both online and offline.
- It can be updated incrementally.

- The hierarchical structure is suitable for most (but not all) types of documents.
- It is platform-independent, thus relatively immune to changes in technology.

Disadvantages of XML

- XML syntax is redundant or large relative to binary representations of similar data
- XML syntax is verbose relative to other alternative 'text-based' data transmission formats.
- No intrinsic data type support: XML does not provides a specific notion of "integer", "string", "boolean", "date", and so on.
- Expressing overlapping (non-hierarchical) node relationships requires extra effort.

Correctness

There are two types of correctness in XML:

1) Well formed
    a. An XML document is well formed if it adheres to all default syntax, for example opening tags without closing tags make up a non well formed document
2) Valid
    a. A Valid XML adheres to all other semantic rules such as DTD. For example a custom modifier that is not defined will result in a invalid XML file.

There are different kinds of parsers which function on either one of these principles

Syntax

Tags

XML is structure very similarly to HTML. All element are encapsulated between tags. As seen in the example, we have an opening tag and closing tag:

```
<event>
</event>
```

Root and declaration

We also have a root tag, in our example `<UserBookingDatabase>`, which encapsulates all other sub tags. Prior to this root tag, we can have an optional XML declaration which, for example, can contain versioning information or dependencies.

Comments

Comments are held within two distinct tags:

```
<!-- Comments here -->
```

Nowhere within your comment can there be '--', this result in a non-well formed document.

Attributes

Additional markup is used to structure the contents of the XML document. The text enclosed by the root tags may contain an arbitrary number of XML elements.

```
<name  attribute="value">content</name>
```

Nested tags

Tags can be nested however, similar to nested logic in programming, you cannot have overlapping nesting. Here is a bad example:

```
<p>Normal <em>emphasized <strong>strong
emphasized</em> strong</strong></p>
```

Empty elements

Although you can represent an empty element as a concatenation of both the opening and closing tags:

```
<name></name>
```

XML has syntax to display this in a shorter manner:

`<name  />` Or `<name/>`

However an empty tag can still contain attributes.

```
<name  attribute="value"  />
```

Entity References

XML provides two methods for referring to special characters: character entity references and numeric character references.

An entity reference is a placeholder that represents that entity. It consists of the entity's name preceded by an ampersand ("`&`") and followed by a semicolon ("`;`"). XML has five pre-declared entities:

- `&` & ampersand
- `<` < less than
- `>` > greater than
- `&apos;` ' apostrophe
- `&quot;` " quotation mark

Ex: `<company_name>AT&T</company_name>`

If more of these entity references are needed, you can define them in the DTD

```
Ex: <!DOCTYPE example [ <!ENTITY
            copy "&#xA9;">
            <!ENTITY copyright-notice "Copyright &copy; 2006,
            XYZ Enterprises">
        ]>
        <example>
            &copyright-
        notice; </example>
```

Numeric character references

These references are very similar to entity ones, but instead of using a word to describe what the reference is, we use the Unicode hex or decimal value of that character.

```
Ex:     <company_name>AT&#38;T</company_name>
        <company_name>AT&#x26;T</company_name>
```

JDOM

JDOM is an open source Java-based document object model for XML that was designed specifically for the Java platform so that it can take advantage of its language features. It allows to structurally store data in a familiar way tat works easily programmatically. You can either parse information from an XML file or create your own tree which can then be exported to a custom XML file.

Ex:

```
<shop name="shop for geeks" location="Tokyo,
  Japan"> <computer name="iBook" price="1200$" />
  <comic_book name="Dragon Ball vol 1" price="9$"
  /> <geekyness_of_shop price="priceless" />
</shop>
```

If you want to collect this information into separate variables in Java, you'd proceed like so:

```
SAXBuilder builder = new SAXBuilder();
Document doc = builder.build(new FileInputStream("foo.xml"));
Element root = doc.getRootElement();
// root.getName() is "shop"
// root.getAttributeValue("name") is "shop for geeks"
// root.getAttributeValue("location") is "Tokyo, Japan"
// root.getChildren() is a java.util.List object who has
3 Element objects.
```

To create the same file using Java code you'd want to do this:

```
Element root = new Element("shop");
root.setAttribute("name", "shop for geeks");
root.setAttribute("location", "Tokyo, Japan");
Element item1 = new Element("computer");
item1.setAttribute("name", "iBook");
item1.setAttribute("price", "1200$");
```

```
root.addContent(item1);
// do the similar for other elements
XMLOutputter outputter = new XMLOutputter();
outputter.output(new Document(root), new
FileOutputStream ("foo2.xml"));
```

References

ArgoUML

- http://argouml.tigris.org/
- http://en.wikipedia.org/wiki/ArgoUML

XML

- http://en.wikipedia.org/wiki/XML
- http://en.wikipedia.org/wiki/JDOM
- http://www.jdom.org/