University of Pretoria etd – Peer, E S (2005)

Chapter 5

Languages and Tools

"Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning." – Rich Cook

This chapter addresses various language and tool prerequisites for working with the software implemented for this research.

Section 5.1 introduces XML, which is used as a configuration and data representation language. Java and J2EE, which were chosen as implementation platforms are discussed in Sections 5.2 and 5.3 respectively. Section 5.4 presents the XDoclet tool, which enables attribute oriented programming. The JUnit framework, used for writing software unit tests, is introduced in Section 5.5. Finally, the chapter concludes with a brief summary in Section 5.6.

5.1 XML (eXtensible Mark-up Language)

XML (eXtensible Mark-up Language) is a recommendation by the World Wide Web Consortium (W3C)[1], for defining structured documents [53]. Structure is imposed on a text document by marking up the content with user defined tags. Figure 5.1 is an example of a simple XML document, a structured list of phone numbers.

Note that, given the proper choice of tag names, a document is reasonably self describing. It should be clear, even to somebody unfamiliar with XML, that the example

[1]http://www.w3c.org/XML/

University of Pretoria etd – Peer, E S (2005)


```
<?xml version="1.0"?>
<!DOCTYPE phoneBook SYSTEM "phonebook.dtd">
<phoneBook>
  <contact>
    <name>Joe Bloggs</name>
    <phone type="home">012-315-7834</phone>
    <phone type="cell">082-243-4244</phone>
  </contact>
  <contact>
    <name>John Doe</name>
    <phone type="home">012-514-1423</phone>
    <phone type="work">011-612-3431</phone>
    <phone type="cell">083-561-9542</phone>
  </contact>
  <!-- possibly more contacts -->
</phoneBook>
```

Figure 5.1: A Simple XML Phone Book Document

is a list of contacts in a phone book with their associated phone numbers. More importantly, because the document is structured, according to the `phonebook.dtd` document type definition, software can make sense of it too. The power of XML stems from the fact that standard tools can be used for manipulating any well formed document and that the grammar for a particular type of document can be defined and extended to suit its natural structure.

For example, the logical structure of a book can be broadly defined in terms of chapters, sections and paragraphs. DocBook [115], which defines an XML document type for marking up books and technical documentation, enables an author to write a book based on its natural logical structure. Since the book is just another XML document, the structure is machine readable and so standard style sheet templates can be used to transform the document into any format, in any desired medium.

Section 5.1.1 defines the syntax requirements for XML documents to be well formed. Next, document types and schemas are discussed in Section 5.1.2. Finally, the Document

University of Pretoria etd – Peer, E S (2005)

Object Model (DOM) is explained in Section 5.1.3.

5.1.1 Well Formed Documents

A document and its tags, more formally known as elements, must satisfy certain rules in order to be well formed [123]. Any well formed document is guaranteed to be parsed without error by a standard XML parser.

There are three simple rules pertaining to elements: i) there must be one and only one root element; ii) an opening tag must be followed by a corresponding closing tag, where matching is case sensitive; and iii) elements must be properly nested, so an opening tag which is outside the scope of a nested element must be closed in the same outer scope.

Elements may contain optional attributes, such as the type attribute in the phone elements in the example. Further, elements may be empty, in which case the element may be closed, using a shorter syntax, by suffixing the opening tag name with a forward slash, for example <element/>, instead of <element></element>. Empty elements may still contain attributes. Special cases include id attributes, which are used to associate a document scoped unique identifier with an element, and corresponding idref attributes, which are references that can be followed to elements identified by an id attribute.

Further, there are restrictions on the characters that may be used in attribute and tag names. Only alphanumeric characters, hyphens, underscores and periods may be used. Throughout a document, the literal strings "&" and "<" must be used in place of the "&" and "<" symbols respectively, otherwise they would be mistaken as mark-up. Similar string literals are defined for quote, apostrophe, and greater than symbols, but their use is optional. Another way to prevent character data from being processed as mark-up is to include it within a special CDATA tag, for example "<![CDATA[text that should not be processed]]>". Finally, comments are enclosed within the "<!--" and "-->" tags.

5.1.2 Document Types and Schemas

Documents that conform to a given structure, constrained by either a DTD (Document Type Definition) or a schema, are known as valid documents. These constraints are enforced by the XML parser before an application sees a document. Validated documents permit software to make assumptions about the structure of a document, making XML

University of Pretoria etd – Peer, E S (2005)

processing software easier, and safer, to write.

```
<!ELEMENT phoneBook (contact*)>
<!ELEMENT contact (name, phone+)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ATTLIST phone
  type CDATA #REQUIRED
>
```

Figure 5.2: Phone Book Document Type Definition (phonebook.dtd)

Figure 5.2 provides the DTD for the phone book example. A DTD defines all the valid document elements and their relationships with their children. A suffix of "?", "*" or "+" after a child element name determines the number of children elements which may occur, namely, zero or one, zero or more and one or more respectively. Sequences are indicated by a comma separated list of children. Thus, the second line indicates that a `contact` element must consist of a `name` element followed by one or more `phone` elements. Legal attributes are defined by an `ATTLIST` description. The `PCDATA` type corresponds to character data that will be parsed for further mark-up, while the `CDATA` type is ordinary character data. Note that an attribute value may not be of type `PCDATA`, it will never be processed as mark-up. The `DOCTYPE` reference in the document instance specifies which element in the DTD should be considered as the root element.

Instead of using a DTD, an XML Schema [112, 13] can be used to define a document type. Schemas have several advantages over DTDs. Firstly, because the schema language is just another XML document, there is no need to learn a separate DTD language, and standard parsers and tools can be used to read and manipulate schemas. Furthermore, XML Schema has a more extensive type system that supports inheritance. Most importantly, because schemas are supported using namespaces, a single document can mix document elements from multiple schemas, simply by declaring multiple namespaces that reference different schemas.

The schema for the phone book example is presented in Figure 5.3. The `xmlns:xs` attribute in the root element defines the `xs` namespace. Thus, elements prefixed by `xs:` are instances of the `http://www.w3.org/2001/XMLSchema` schema. In this par-

University of Pretoria etd – Peer, E S (2005)


```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="phoneBook">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="contact"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="contact">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="name"/>
        <xs:element minOccurs="1" maxOccurs="unbounded" ref="phone"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="name" type="xs:string"/>
  <xs:element name="phone">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute name="type" type="xs:string" use="required"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 5.3: Phone Book Schema (phonebook.xsd)

University of Pretoria etd – Peer, E S (2005)

ticular case, the namespace is a reference to the definition of the valid elements for an XML Schema document, which is also defined in terms of XML Schema. Default namespaces of documents can also be defined by schemas. Thus, in the phone book example, the `DOCTYPE` line can be omitted and the root element altered to read `<phoneBook xmlns="phonebook.xsd">`, where `phonebook.xsd` is the file containing the phone book schema.

5.1.3 Document Object Model (DOM)

The Document Object Model (DOM) provides a language neutral interface for manipulating XML documents programmatically [58]. XML documents are represented by an in memory tree based object structure, where nodes are defined for all possible components of an XML document, including elements, attributes, comments and free standing text. Since DOM bindings exist for all major programming languages, XML content to be accessed, processed and manipulated in a standard way on any platform.

As an alternative to the DOM, the Simple API for XML (SAX)[2] provides an event model interface for processing documents. SAX, which is an extension of the *Observer* pattern in Section 3.3.3, enables documents to be processed without the need to build a, possibly large, in-memory representation.

5.2 Java

Java is a modern, high level, general purpose, object oriented programming language [33, 59]. Programs written in Java are compiled into an intermediate language, known as byte code, which is interpreted at run time by a Java Virtual Machine (JVM). Benefits of Java include:

- **Platform and Vendor Independence:** A cornerstone of Java has always been the concept of write once, run anywhere. This goal has been achieved by virtue of the JVM, since only the underlying virtual machine need be ported to each platform where Java is supported. Supported platforms include Windows, Linux and MacOS. Further, the Java specification is guided by the Java Community Process

[2]http://www.saxproject.org/

University of Pretoria etd – Peer, E S (2005)

(JCP)[3], giving multiple vendors the opportunity to contribute and participate in the decisions that dictate the future direction of Java. Competing JVM implementations are available from multiple Java vendors, including Sun Microsystems, IBM, BEA[4] and the Blackdown project[5], ensuring diversity in the market place and the future safety of the Java platform. A completely free JVM implementation is also being worked on by the GNU Classpath community[6], along with a native Java compiler as part of the GNU Compiler Collection (GCC).

- **Garbage Collection:** Garbage Collection (GC) relieves a programmer from having to explicitly manage memory deallocation, resulting in safer code due to the reduced risk of introducing difficult to find memory leaks. GC is associated with at least some overhead, since an additional process must be executed from time to time to recycle unreferenced memory. Counter intuitively, in spite of this overhead, GC can have a net increase in the performance of an application[7]. For example, heap compaction performed by GC increases the likelihood of cache hits. Further, since GC only executes when memory is tight, programmes with a low memory footprint may never need to run a GC cycle. Another factor to consider is that smart pointer based reference counting techniques, which are typically employed to simplify memory deallocation in non-GC languages, can carry a much higher overhead than GC, since counters need to be updated for every assignment. Worse, reference counting techniques are dangerous because they cannot deal with circular references or anonymous objects. Finally, explicit destructors can be a significant performance overhead for stack allocated resources.

- **Java Foundation Classes (JFC):** The Java platform, which is guaranteed to be available on any compliant JVM, is defined in terms of the JFC. The JFC, or Java APIs, provide XML processing, Input/Output (I/O), Graphical User Interface (GUI) and networking services to applications. Further, since version 1.5 of the JFC, a type safe collections framework using templates is also provided. Also, the reflection API is a fundamental reason Java was chosen as an implementation

[3]http://www.jcp.org
[4]http://www.bea.com
[5]http://www.blackdown.org/
[6]http://www.gnu.org/software/classpath/classpath.html
[7]http://www.digitalmars.com/d/garbage.html

University of Pretoria etd – Peer, E S (2005)

language for this research. The JFC has been through many revisions, gradually improving its design, which is heavily based on design patterns. For example, I/O services such as buffering and compression are provided using *Decorators* (refer to Section 3.2.3) and the collections framework supports *Iterators* (refer to Section 3.3.2).

- **Tool Support:** Many high quality Java development tools are freely available. At least two good enterprise class development environments are available for free, namely Eclipse and NetBeans. The Javadoc tool, packaged with the standard Java SDK (Software Developer Kit), extracts special comments in the source code into a navigable HTML (HyperText Mark-up Language) format. XDoclet (refer to Section 5.4), originally based on Javadoc, can be used to generate various artifacts from meta-data embedded in special Javadoc comments. Debugging distributed and server side applications can be made simpler with a logging framework such as Log4j[8]. JUnit (refer to Section 5.5) is a unit testing framework for Java. The build process of complex Java projects is script-able using the Apache Ant[9] build tool.

  These are only the tools that have been used, or are being considered, for this research. There are many other free third party tools, frameworks and APIs available for Java, supported by a diverse Java community.

- **Performance:** Java is still plagued by the stigmatism of poor performance due to early and immature implementations of the JVM. This situation is further exacerbated by the intuition that interpreted languages with additional GC overheads must have inferior performance to natively compiled languages. Modern HotSpot [1] JVMs, however, have dramatically improved the performance of Java, to the point where it is comparable and in certain circumstances superior in performance to natively compiled languages such as C/C++ [25, 95]. HotSpot JVMs sport state of the art generational GC algorithms, speculative run time optimisation using dynamic profiling, and Just In Time (JIT) compiling of critical code, known as hot spots, to instructions optimised for the local processor. Numerous micro-benchmarks [72, 76, 23, 70] have been conducted, which show Java performance to

[8]http://logging.apache.org/log4j/docs/index.html
[9]http://ant.apache.org

University of Pretoria etd – Peer, E S (2005)

be on par with other languages.

A simple benchmark, called NastyPSO, was performed around the time the decision to port the implementation code used in this research to Java was being made. NastyPSO[10] is a quick and dirty implementation of a simple particle swarm optimiser (refer to Section 2.4.1) in C#, C++ and Java. To make the benchmark fair, no language specific libraries are used. For example, the random number generator used in the code is implemented by NastyPSO in each language. Thus, the only differences between the implementations are syntactic. Further, no OO features of are used, purely testing the number crunching ability of each language. The source code for NastyPSO is made available so that the results presented in Table 5.1 can be verified independently by the reader.

Table 5.1: NastyPSO Performance

Language	**Compiler / VM**	**Time (seconds)**
C++	Intel Compiler (-O3 -march=pentium4)	391.3
C++	Intel Compiler (-O3 -march=pentium4 -mp)	570.6
Java	Sun HotSpot VM 1.4.2.03 (-server)	584.6
Java	IBM VM 1.4.1	584.8
Java	Sun HotSpot VM 1.5.0_beta1 (-server)	600.8
C++	GNU Compiler 3.3.2 (-O3 -march=pentium4)	742.8
C++	GNU Compiler 3.3.2	754.0
Java	JRockit 8.1	756.8
Java	GNU Compiler (GCJ)	934.4*
Java	Blackdown 1.4.1 (-server)	945.0
Java	Sun HotSpot VM	992.4*
Java	Sun Classic VM	1596.5*
C#	Mono 0.28	2572.9

Times recorded are the CPU scheduled time given by the Unix `time` command, so the results are invariant to varying load on the machine. Unfortunately, some parameters,

[10] http://cilib.sourceforge.net/NastyPSO/

University of Pretoria etd – Peer, E S (2005)

which are hard coded in the implementation, have changed since the time the benchmark was performed and were not properly recorded. Further, times suffixed by an asterisk have been interpolated based on a run conducted several months earlier, where the versions of the compilers and virtual machines were not recorded. The scaling was performed relative to the performance of the Sun HotSpot (Server) VM, which was a common denominator in both sets of results, even though the versions may not have matched. That said, conclusions about the relative performance of the implementations are still valid, even though the times may not exactly match those produced by the current version of the code.

The first conclusion evidenced by the results is that the choice of JVM can have a measurable performance difference. In fact, selecting the server parameter of the Sun JVM made the difference from one of the worse performing configurations to one of the best. The server JVM performs more aggressive run time optimisations at the cost of slower startup times, making it suitable for long running processes. Surprisingly, the free GNU compiler was unable to match the best JVM performances, even under very heavy optimisations for the platform. The Intel[11] compiler was able to outperform the best Java configuration, however, if the compiler was forced to reject optimisations that may affect the floating point precision then this difference was not significant. C# was tested under the free Mono platform and was found to perform significantly worse than any of the other configurations. Microsoft's[12] implementation of the .NET platform was not tested, since it is not platform independent.

Unfortunately, OO polymorphic method calls are still expensive, even in C++ although less so than Java, making object based polymorphic numeric types expensive, particularly in the tight loop applications needed by CI algorithms. Fortunately, object in-lining [18] may provide a solution to this problem in future. Object in-lining is a compile time optimisation that essentially unpacks code into a calling class whenever polymorphism is not required, so a developer can write clean OO code while leaving the hard work of making it perform well to the compiler.

[11] http://www.intel.com
[12] http://www.microsoft.com

University of Pretoria etd – Peer, E S (2005)

5.3 Java 2 Enterprise Edition (J2EE)

Java 2 Enterprise Edition (J2EE) is centred around Enterprise Java Bean (EJB) technology, enabling the development of scalable multi-tiered enterprise class applications [8]. EJBs are software components that are managed in the context of an application server container. The container forms the interface between EJB components and the underlying platform, providing caching, clustering, security, session, transaction, and persistence management services.

An EJB comprises three essential components: i) an application interface; ii) a home interface; and iii) an implementation class. The application interface, also known as a business interface, specifies the services that a bean provides to clients. Programming to an explicit interface with no direct knowledge of the implementation means that the implementation can be switched without affecting any clients. The Java Naming and Directory Interface (JNDI) provides an additional level of indirection, making implementation classes configurable at application deployment time. Thus, EJB clients are not aware of the implementing class details, they are only exposed to an abstract JNDI name for the implementation providing the service. Primarily, home interfaces are responsible for managing the life cycle of individual beans, providing methods for locating and creating them. Beans are destroyed by calling a remove method directly on an instance. Services that apply to more than one particular bean instance are also provided by the home interface, making those services analogues for class scope, or static, methods. Further, EJB interfaces for local and remote clients are differentiated in J2EE, so different subsets of a bean's services can be provided to local and remote clients. Finally, an implementation class for an EJB provides the code behind both the home and application interfaces.

The J2EE architecture is layered, cleanly separating different responsibilities into separate layers. At the lowest level, the persistence layer, discussed in Section 5.3.1, is responsible for managing the storage of application data. Above that, the application layer, in Section 5.3.2, is responsible for handling all the application logic, also known as business logic. The presentation layer, discussed in Section 5.3.3, provides the interface to the user. In general, separating the architecture into even more layers is possible, if it makes sense to do so in the context of the application. The purpose of the layers is to improve maintainability of the code by decreasing the dependencies between layers,

University of Pretoria etd – Peer, E S (2005)

changes to one layer should at worst affect the layer immediately above it. In addition, the separation of application and presentation logic means that the same application logic can be used for multiple presentation mediums. For example, a rich GUI client and a web interface, both separately implemented in the presentation layer, should share the same application logic. Finally, the deployment of J2EE applications is discussed in Section 5.3.4

5.3.1 Persistence Layer

Two types of persistence EJB exist in the J2EE specification, Container Managed Persistence (CMP) beans and Bean Managed Persistence (BMP) beans. BMP beans require persistence logic to be manually coded by the developer, while CMP beans delegate persistence logic to the application server container.

Persistence EJBs, also known as entity beans, present an OO view of an underlying relational database [30], or indeed any data store. Although the object relational mapping need not necessarily be a one-to-one correspondence with the underlying database tables, each entity bean instance typically represents a single row in a relational database table. Each column corresponds to a property of the CMP bean, where a property has its usual OO definition of a field with an accessor, or get method, and a mutator, or set method. Relationships are represented by collection valued properties. These relationships are typically bidirectional, with many-to-many relationships being supported by collections on each side of the relationship. Figure 5.4 illustrates how the one-to-many relationship between between a customer and a number of accounts would be represented by entity beans.

Note how the home interface, only shown for the customer entity, can be used to locate existing- and create new entities. More importantly, for CMP beans, it is not necessary to provide implementations for any of the methods, they are simply declared abstract, and the private fields are omitted. The container provides all the necessary functionality to query the underlying database and ensure that the interface works as expected. The database is automatically updated whenever collections are manipulated or a mutator is called. Further, CMP beans can have a significant performance advantage over hand crafted database interactions, due to entity caching and preloading.

University of Pretoria etd – Peer, E S (2005)

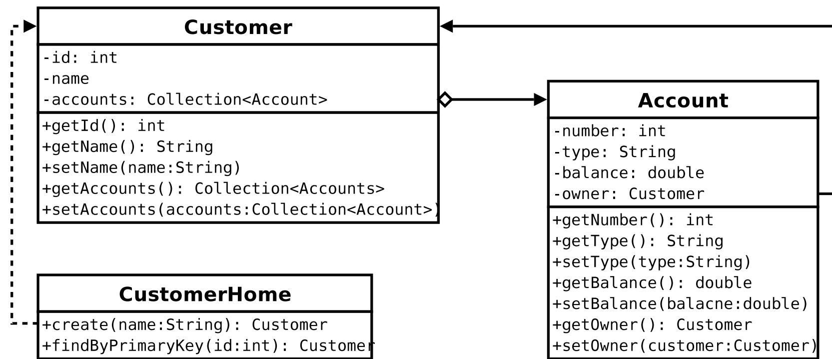

Figure 5.4: EJB Entity Relationship

5.3.2 Application Layer

Three types of application layer EJBs exist, message driven beans, stateless session beans and stateful session beans.

Message driven beans provide an asynchronous interface for clients accessing application layer objects via the Java Messaging Service (JMS), typically using XML based messages. A message driven bean's interface consists of a single on message method, which must unpack the message and do something sensible with it; perhaps calling other application layer beans or sending off other messages as a result.

Session beans typically present a session *Facade* [4] (refer to Section 3.2.4) to clients, which only exposes those parts of the system which are interesting to a given client. Some clients may require application state to be stored over multiple synchronous requests. For example, a shop application would need to store the contents of a shopping cart for the duration of the user session. A J2EE application server container automatically handles session state by creating a new instance of a stateful session bean for each client. Sessions that do not require state should use stateless session beans, enabling the container to share a single instance amongst multiple clients, if it is more efficient to do so.

Having the server container manage message and session beans means that applications can easily be scaled up over multiple servers. For load balancing, an application server simply needs to ensure enough stateless beans are instantiated for a particular service to saturate the given hardware. Fault tolerance is achieved by ensuring that stateful beans are distributed to one or more backup servers. All this is achieved without the explicit knowledge of the developer, making it easier to write scalable, fault tolerant

University of Pretoria etd – Peer, E S (2005)

applications.

5.3.3 Presentation Layer

The presentation layer presents a developer with many choices. The interface presented to users might be a heavyweight rich client implemented using the JFC or it may be a highly accessible web application powered by a combination of any number of existing presentation tier frameworks. For example, Struts[13] with JSP (Java Server Pages)[14] or the recently released JSF (Java Server Faces)[15] framework. It could even be a very thin layer that simply forwards messages to an underlying message driven bean, perhaps implementing an electronic mail interface.

GUIs should make use of the Model View Controller (MVC) [4] architectural pattern. The model, which represents data or functionality behind the user interface, is accessed via session beans in the application layer. A view is responsible for presenting its model to the user and returning control to the controller after the user takes action. The controller then determines the next view based on the current view and the action taken by the user. In the case of Struts, the controller is implemented by a single Servlet[16] which directs application flow between various views which are implemented by JSPs.

5.3.4 Deployment

The real power of J2EE stems from the ability to customise an application at deployment time without altering any source code. Depending on the application server, this deployment configuration, also known as a deployment descriptor, is usually specified in one or more XML documents. The following are some of the most important configurable aspects of J2EE applications:

- **Security:** J2EE provides a declarative security model based on the Java Authentication and Authorisation Service (JAAS)[17] specification. User and role based access rules for beans and their individual methods are declared in the deployment descriptor. The container performs run time security checks for each method

[13] http://struts.apache.org/
[14] http://java.sun.com/products/jsp/
[15] http://java.sun.com/j2ee/javaserverfaces/
[16] http://java.sun.com/products/servlet/index.jsp
[17] http://java.sun.com/products/jaas/

University of Pretoria etd – Peer, E S (2005)

call and throws a security exception if a client attempts to call any unauthorised method. The open source JBoss[18] application server provides this functionality by wrapping EJBs inside a security *Proxy* (refer to Section 3.2.5), which performs any necessary checks before delegating requests to the actual bean.

- **Entity Relational Mapping:** Even though the container can provide the implementation for database interactions, it is still necessary to inform the container about the type of database, along with table and column names onto which entities are mapped. Further, the entity methods that participate in relationships need to be declared.

- **Transactions:** EJB containers are capable of providing full ACID (Atomicity, Consistency, Isolation and Durability) transaction support [30]. Transaction boundaries are specified in the deployment descriptor for beans and methods. For methods, a transaction is opened at the start of a method call and is closed again when the method exits normally. If an EJB exception is thrown then the transaction is rolled back with no side effects. The Container may also perform deadlock detection and roll back transactions that cause deadlock. The isolation level of transactions is typically also configurable. Transaction support is also provided using a *Proxy* in JBoss.

- **Application Server Configuration:** The configuration, pertaining to a given application, for the application server is usually also specified in the deployment descriptor. For example, the caching and preloading behaviour for entities is configurable in JBoss. Clustering strategies and other performance related settings, such as bean instantiation policies, can also be configured.

5.4 XDoclet

XDoclet[19] is a free attribute oriented programming tool, which can be used to generate artifacts from annotations embedded as special Javadoc comments in source code.

XDoclet is an invaluable tool for EJB developers, enabling them to automatically generate any required interfaces and deployment descriptors directly from an annotated

[18] http://www.jboss.org
[19] http://xdoclet.sourceforge.net

University of Pretoria etd – Peer, E S (2005)

implementation class for an EJB. For example, to mark a method for inclusion in the application interface, a developer need only include an `@ejb.interface-method` annotation in the Javadoc comments preceding the method. Declaring JAAS access rules for a method can be achieved by prefixing the method with an `@ejb.permission` tag followed by the appropriate user or role based permissions. Similar tags are defined for declaring entity relation mappings, transaction boundaries and application server specific configurations.

The recent syntax enhancements for annotations in Java 1.5 means that future versions of XDoclet may move their annotations out of Javadoc comments into the actual code. An advantage of proper annotations will be the ability to query these attributes using the standard Java reflection API. For example, it would be possible to query security annotations before calling a method, where currently the only way to determine these permissions is to attempt the operation and catch the security exception that might be thrown.

XDoclet is more general than simply an EJB tool, with tags defined for various other applications, including the Spring framework, Hibernate, JDO, Axis, Struts and JSF amongst many others.

5.5 JUnit

Unit testing is the practice of performing automated tests on units of code, typically testing the behaviour of the public interface of individual classes. The fact that the tests are automated is the most important factor. Automated tests are easy to run, meaning they can be scripted into the build process to give early warning of something getting broken during code maintenance. This safety net gives developers more confidence to work on the code, particularly when maintaining code they did not write, since even small changes can be tested against the entire test suite, rooting out any unexpected side effects. If the tests pass then chances are nothing got broken, assuming the tests are representative of the required behaviour.

Tests should be maintained in tandem with the code. The XP (eXtreme Programming) paradigm [11] advocates writing a complete test suite for a unit before writing its implementation, so that passing all the tests becomes the measuring standard for the completeness of the implementation.

University of Pretoria etd – Peer, E S (2005)

Unit tests serve another important purpose, namely documentation. Unlike comments which can easily fall out of synchronisation with the code implementation, automated tests immediately show any discrepancies that need to be addressed. Unit tests implicitly document the intended behaviour of the code, since that is precisely what they are testing.

JUnit[20] is a free framework that facilitates unit testing in Java. Figure 5.5 illustrates the Test *Composite* (refer to Section 3.2.2) employed by JUnit.

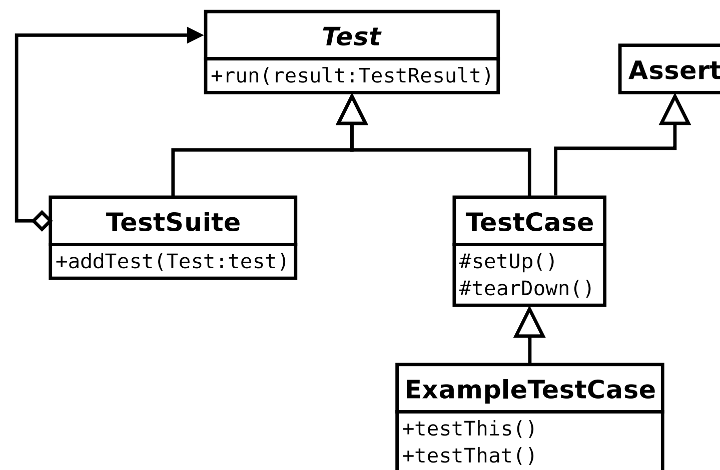

Figure 5.5: JUnit Composite Test Framework

Graphical and command line tools which are capable of executing a `Test`, which may be an entire suite of tests, are provided. The `TestSuite` composite can be used to build a hierarchy of test cases that mirrors the package hierarchy of the software, with one `TestCase` dedicated to each class being tested. Adding new tests for a class is made trivial, only requiring the developer to write another method prefixed with the string "test". The JUnit framework uses the Java reflection API to introspectively call each test method in turn. The `setUp()` and `tearDown()` methods are called by the framework before and after each test method respectively. These methods can be used to configure a fixture that is available to all the test methods. Various methods for testing assertions are inherited in via the `Assert` class. Assertions that fail are gathered into a test result and are reported by the tool after all the tests have been executed.

[20]http://www.junit.org

University of Pretoria etd – Peer, E S (2005)

5.6 Summary

XML and Java were introduced as languages used in the development of CILib and CiClops. In particular, Java was motivated as an appropriate choice of implementation language due to its platform and vendor independence, garbage collection, the Java foundation classes, good tool support and high performance.

Next, an overview of the J2EE framework, which is used by CiClops, was presented. J2EE provides powerful services, such as container managed persistence and transactions, to applications built using EJBs.

Finally, the XDoclet tool and its role in easing EJB development was discussed, followed by a brief introduction to the JUnit testing framework.