

# Atlas: Serving Dynamic Courses Dynamically

Ian Cavers, Mik Kersten, and George Tsiknis

Department of Computer Science

University of British Columbia

**Subject Areas :** web-based learning, course tools, web-based application architecture, servlets,  
distributed applications, aspect-oriented programming

<b>1</b>	<b>INTRODUCTION</b>	<b>3</b>
<b>2</b>	<b>WEB-BASED COURSE TOOLS</b>	<b>4</b>
2.1	Methods of Serving Web-based Applications	4
2.2	Web-based Course Tool Requirements	5
<b>3</b>	<b>ATLAS: USER VIEW</b>	<b>6</b>
3.1	Student Interface	6
3.2	Dynamic Course Planning	8
<b>4</b>	<b>ATLAS: DEVELOPER VIEWS</b>	<b>11</b>
4.1	DSB Architecture	11
4.2	Technology	14
4.2.1	Java and Servlets	14
4.2.2	Objectspace Voyager	15
4.2.3	AOP and Patterns	16
4.3	Atlas/DSB Design and Implementation	17
4.3.1	courseTree Implementation	17
4.3.2	HTML Generation With webObjects	19
4.3.3	DSB Component Distribution	22
<b>5</b>	<b>DISCUSSION</b>	<b>24</b>
5.1	Performance Evaluation	24
5.2	Usability	25
5.3	Future Work	25
<b>6</b>	<b>SUMMARY</b>	<b>26</b>

## 1 Introduction

Following advances in web application technology, web-based course tools have evolved from simple hyper-linked pages composed of text and images to large systems composed of numerous interacting CGI programs and databases. The performance of these applications is often limited by the current state of web-application frameworks: a set of related CGI programs and files is not capable of composing a highly interactive, modifiable, configurable, and performance oriented application that enhances the web-based learning experience.

The Advanced Teaching and Learning Academic Server (Atlas) is an attempt to create a learning environment that more closely resembles the dynamics of a desktop-based application than its web-based counterpart. The key requirements of Atlas focus on a greater degree of interactivity and a dynamic user interface, as well as a dynamic course model capable of re-structuring course material based on student needs. The architecture that supports Atlas, called the Distributed Servlet Broker (DSB), was designed along with Atlas to allow for the high performance characteristic that a highly interactive web-application demands. The DSB is a distributed serving architecture capable of adapting to different network conditions, application loads, and web-server loads. The architecture can switch from local process, to distributed, and to parallel distributed serving modes at runtime. It can thus optimize application-serving performance and exploit the natural parallelism associated with web-server requests.

The dynamic user interface, dynamic user-based course generation, and runtime re-configurable persistent serving architecture make it possible for Atlas to offer a much higher level of interactivity with a performance that significantly exceeds much simpler CGI-based web course tools. The dynamic nature of the course content allows Atlas to present instructional material that is tailored to the individual user's aptitude and needs. The result is a moderate sized system that consists of approximately 200 Java™ [GJS96] classes and 50 packages. We first present this system from the point of view of its requirements as a web-based learning tool and the requirements common to web applications. Next, we present it from the point of view of the user, in order to depict the application's uses and usability. Finally, we present it from the point of view of the developer and software architect, in order to share our experience and use of development technology.

## **2 Web-based Course Tools**

Several years ago web browsers revolutionized the sharing and distribution of multimedia content across networks. To this date, the majority of available content is file-based, providing a static interface based on hyper-links. However, the introduction of new browsers and web server technology creates a potential to make the interface to this content similar to that of desktop-based applications. The Atlas project is an attempt to present an application in a web-based environment without sacrificing user interactivity and interface quality.

### **2.1 Methods of Serving Web-based Applications**

The main difficulty with serving web-based applications is the performance characteristic of the web and its browser interface. Several approaches to serving these applications exist, and each is distinguished by a different level of performance, scalability, and ease-of-use.

The simplest and still prevalent method of serving web content uses hyper-linked files. Given a well-organized web site, a large volume of content can be presented and readily referenced by means of files containing text, images, and links to other files. The interface in this case is given by static web pages, where the response to user action results loading a new pages. This approach limits the interactivity of web-based applications, but is suitable for applications that demand limited user interactivity. For example, a telephone number directory interface may constitute this sort of application. In this case, the user is able to browse a large, possibly cascading, index of numbers, and retrieve the desired one.

The shortcoming of the above method is the static interface. Even a simple application usually demands a very large number of states to be represented by the interface, which can require an unmanageable number of web pages. The solution is the use of dynamic pages, generated either by the web server or by the client's browser. Client-side dynamic pages are currently a combination of Java Applets™ [JA], JavaScript [JS], Cascading Style Sheets [CSS], or dynamic HTML [HTM], and provide an interface that not limited by network throughput or latency since the application resides in the client's browser. However, more sophisticated applications typically require state to be stored in a central database that is usually not accessible to client-side mechanisms for security reasons. The integration of a database with dynamic content is given by the Common Gateway Interface (CGI) [CGI]. With this interface the above mentioned telephone

directory can respond to a user's query with a database lookup, and programmatically generate a page with the resulting content.

The necessity for the Atlas/DSB architecture arose from the limitation of serving CGI-based applications. A CGI based web-application is characterized by a series of small programs which are able to take user input from a browser, access a database, and subsequently generate the resulting pages. However, this approach of initializing all of the application state when the server request is made not only creates a performance bottleneck, but distances the web-based application from a desktop-based application by limiting the scope of the interactions and computation that can be performed while responding to client requests. The solution to these problems is an application that persists and can maintain state across client requests, instead of exiting when a request is finished in the fashion of a CGI program. This sort of web-based application is much closer in nature to its desktop-based counterpart in that it acts as a whole and not as a collection of disjoint components.

### **2.2 Web-based Course Tool Requirements**

The requirements of Atlas are split into those that are particular to its domain, and those that are relevant to web-based applications in general. Due to the large amount of processing required with a typical web-based course tool request, performance measured by response time is a particularly important issue. The performance of a standard CGI-based architecture does not suffice in this respect due to the extra overhead of loading and executing a sizeable CGI program.

The success of Atlas depends on its performance and reliability. In order for a web-based application to be used reliably, uptime very near 100% is necessary. Performance, measured by the time that it takes to process a request, must be similar to or better than that of serving static files. Scalability, preferably involving no downtime, is just as important an issue in order to deal with inevitable traffic increases and extensions of functionality. The subsequent requirements particular to Atlas arise from both the functionality required of the application and usability issues:

- *Dynamic user-based course planning:* courses are to be presented in a dynamically customized format that meets individual student needs, aptitude, and learning habits.
- *Dynamic web interface:* the interface to the Atlas functionality must be user friendly, user configurable, and take into account discrepancies between web browser in order to offer a high level of interactivity and usability.

- *Minimal response time:* the processing associated with dynamic course planning and a dynamic web interface is considerable. Application usability can be drastically decreased if responsiveness is compromised; consequently performance is maximized by minimizing request response time.

### **3 Atlas: User View**

The web-based user interface and course traversal facilities define the user's view of Atlas. A high level of interactivity is the main focus of the user interface, which is a means for presenting both course content and a user-centered web-based environment that facilitates learning. The course traversal facilities are focused on a dynamic user model that enables Atlas to tailor courses to individual student needs.

#### **3.1 Student Interface**

An essential goal of Atlas is to provide students with an advanced web-based user interface that:

- Permits convenient access to the application functionality.
- Is able to adapt gracefully to changes and extensions of the application functionality.
- Dynamically customizes the user interface to adapt to the student's needs and browser/network performance.
- Permits the student to re-configure the interface based on his or her preferences.
- Provides a course-view tailored to both the dynamic course traversal and the standard course traversal (i.e. linear traversal through an instructor-defined path)
- Maintains information about the user's session in order to permit the user to resume their current session.
- Provides context-sensitive help.

The result is Atlas' web-desktop environment. This is the front-end to the Atlas application and contains links to all of the functionality available from the application. These links are separated into three categories:

1. *Atlas tools:* Atlas courses, online manual, and quick references.
2. *Collaboration tools:* online help and bulletin board provide users with help and means of interacting with other students and administrators.

## Atlas

3. *Desktop tools*: tools that are useful for the courses (calculator, HTML editor, color picker, etc.), preference setting to control the look and characteristics of the user interface, and control panels to control desktop properties.

Clicking any of the links available causes a corresponding window to be displayed. For example, Figure 1 illustrates the “Online Manual” window selected, with the existing manual sections accessible as links. This selected window contains icons representing the functionality requested, following the style of a desktop motif. If the application functionality is extended (e.g. if there are more Atlas courses or ‘tools’ added) icons are added to the corresponding windows. By means of the ‘preferences’ and ‘control panels’ icons the user can set desktop attributes effecting the look and feel (a selection of ‘desktop themes’ controls settings such as the background image and window colors), features such as the presence of the Java clock that resides on the desktop, or the use of active images for enhancing the interface, or changing their browser bandwidth setting. As will be detailed later, Atlas renders web pages based on user settings and browser attributes enabling the customizability available through the ‘preferences’ and ‘control panels’. Future versions of Atlas will extend this functionality to allow an even higher level of customization.

Once an icon has been selected by the user, the desktop will disappear if the functionality requires an entire screen to run (as is the case with the Atlas courses), or it will be swallowed by a window identical to the windows that contain the icons (as is the case with preferences).



Figure 1: Atlas Web-desktop view

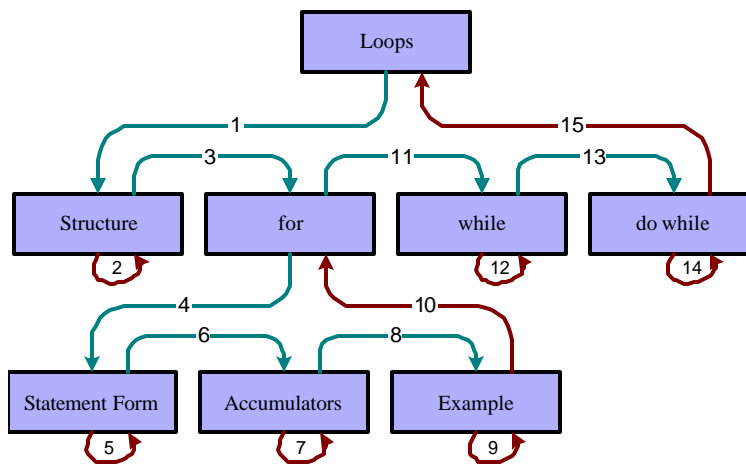
### 3.2 Dynamic Course Planning

The core of Atlas development has been focused on providing the user with an advanced course-navigation interface, and on the construction of the infrastructure necessary to support that interface. The approach that differentiates Atlas from the other approach to serving web courses [WCT] is the fact that Atlas dynamically creates a course based on a particular user model that is consequently tailored to meet the needs of an individual student. All Atlas courses are based on `courseTree`, a knowledge-base of course content.

A snapshot of `courseTree` taken from the C++ course takes the form depicted in Figure x. This structure represents a simple hierarchy, which can be traversed in different ways, representing the different “course navigation” modes of Atlas. The ordering of the default (linear) traversal strategy mode is depicted by the numbered arcs in Figure 2. Note that the traversal is broken down into independent steps, based on either presenting the contents of a node or moving to an associated node. The action of moving to a node when the current is completed takes the form of a move ‘up’ or a move ‘down’ the tree (moves ‘down’ are depicted as green arcs in Figure 2,

edges 1, 3, 4, 6, 8, 11, 13). A move ‘down’ represents the retrieval of the contents of a node for a user, while a move ‘up’ represents an interactive test that must be passed in order for the user to continue their traversal. If the test is not passed, the user is given a variant of it or the same test again. The traversal of the `courseTree` fragment depicted in Figure 2 takes some of the following steps:

1. The user hits the “next” button from the “Loops” node and is presented by the node that follows it in a depth-first traversal view of the course tree, i.e. the “Structure” node.
2. Since the “Structure” node has no children, the user is given a test for the node.
3. Once the test has been passed the user is presented the contents of the “for” node (note that they are not tested on this node until step 10, when they have learned the contents of all of that node’s children).
4. They are presented with the contents of the “Statement Form” node, and so on. . .



**Figure 2: courseTree Traversal Example**

A course can be traversed in one of two modes. The *browse* mode involves presenting the contents of a node when each of the ‘down’ arcs is encountered, and ignoring the ‘up’ arcs that present test material. An *active* mode involves presenting all of the arcs depicted in Figure 2 and imposes the concept of passing a course, since a user can not complete the last node without having successfully completed all of the tests of the previous nodes. The *active* mode is further

broken down into different strategies. The default strategy is presented in the traversal given above—every ‘step’ through the course is a result of the structure of the `courseTree`. More sophisticated strategies involve planning a better path based on the student’s requirements and aptitude. These typically involve pruning out nodes that the user already knows or pruning in nodes that the user needs to learn before they can proceed with the current course node. For example, a user may need to be introduced to sections of the UNIX course before they start working with C++ I/O. More information on traversal strategies is presented in Section 4.3.

Each `courseTree` node is composed of the following:

- *Contents.html*: one or more HTML files that contains the course material for the given node
- *Reference.html*: reference material associated with the node (i.e. tables of operators in the language section of the C++ course)
- *Test.html*: contains dynamic HTML interactive test for the node

As a result of splitting up the contents, reference, and test material for each node different ‘views’ of a course can be presented. For example, a test for the entire course can be generated from traversals that present only the “Test.html” contents. Similarly, a reference document for the course can be generated from traversals that present only the “Reference.html” contents.

The user interface for the course is presented in Figure 3. The course navigation bar, generated by traversing and a course and retrieving all of the node titles, is visible on the left and is present if “browse” mode is used. Navigation, mode switching, and help buttons are visible above the course contents window.

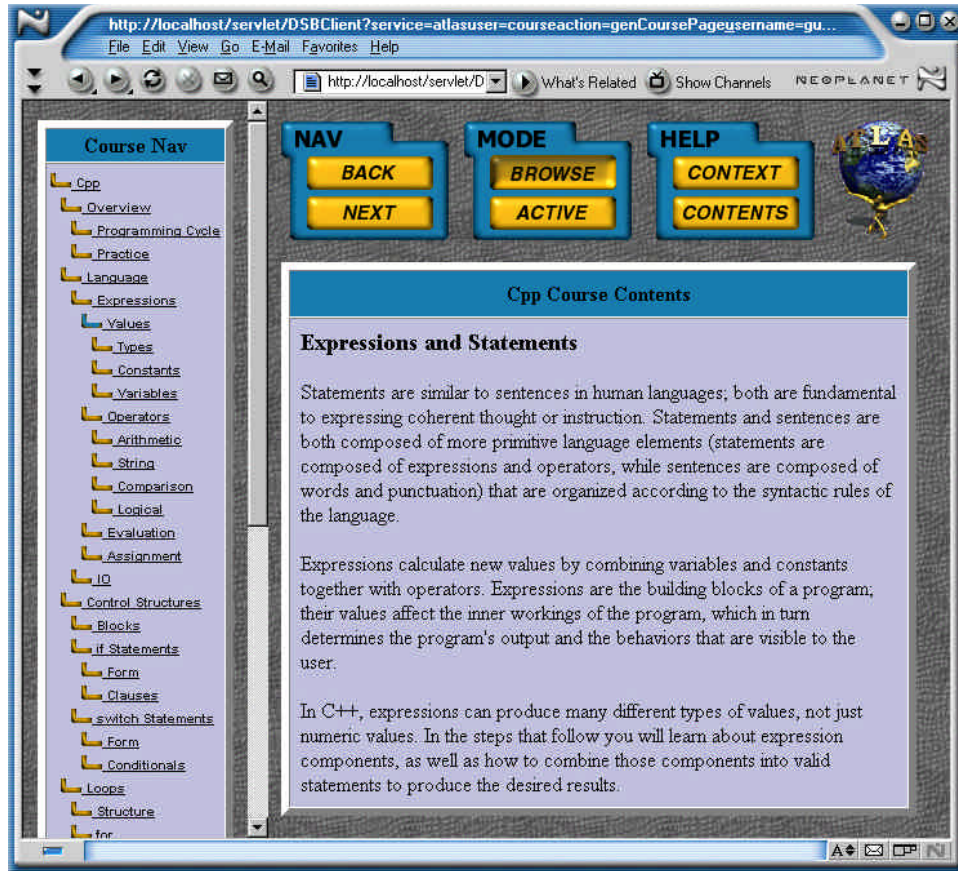


Figure 3: Atlas Course View

## 4 Atlas: Developer Views

Whereas the user's view describes the Atlas functionality, this section explains the experience of developing that functionality and corresponding serving architecture, the new technology used, and some of the implementation details that set Atlas apart from similar web-based applications.

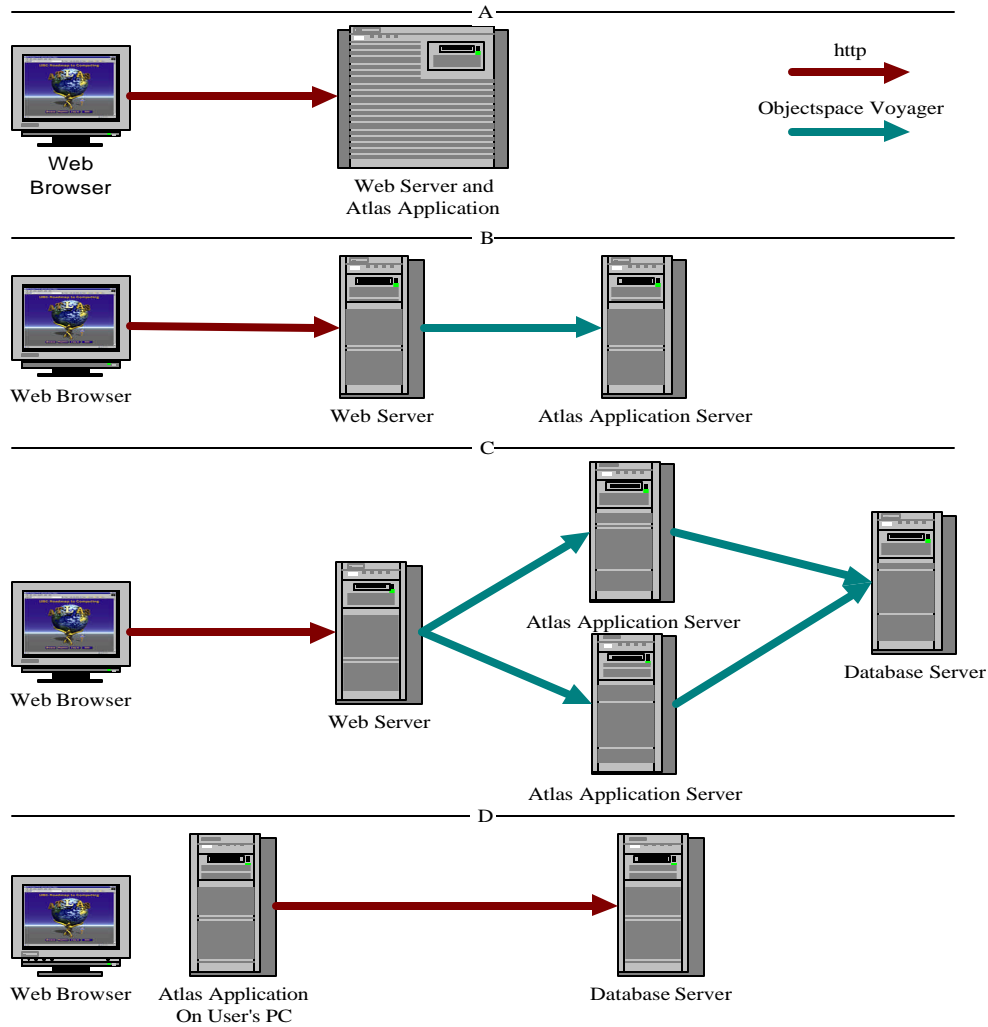
### 4.1 DSB Architecture

We wanted to build Atlas to be economically configurable for the different kinds of educational computing environments in which it might run. Consequently, we set the following two requirements. First, we required Atlas to run in different configurations based on performance requirements, hardware availability, and network conditions in order to optimize performance. Second, we required the system to be scalable at runtime in order to minimize downtime. The result is the Distributed Servlet Broker (DSB), which is the layer between the Atlas application and the web server that handles network configuration modes and the distribution of Atlas.

Four run-time configurations, or *network contexts*, constitute the running modes of the DSB. In the *Single server* context (see Figure xa) client requests are handled as threads within the web server, and the Atlas application is run as a Servlet in the web server. In the “web and application server” (Figure xb) mode, the load of processing Atlas requests is removed from the web server by distributing the application server. The “parallel application server” (figure xc) mode takes distribution a step further by splitting up the load of handling client requests among several network nodes, and maintaining a centralized database. This mode exploits the inherent parallelism associated with HTTP requests and balances the distributed load in order to optimize performance. These three modes address hardware availability, hardware configurations, and application load concerns. However, the role of the web for computation should not be ignored. If a sufficiently high bandwidth exists between the Atlas application and web browser (client), the application is able to migrate the objects associated with its functionality into a Java Applet in the client’s browser in order to minimize communication and protocol overhead; this is referred to as the Applet context (Figure xd).

Typically, the DSB application is configured to run in one of these network contexts. However, it is possible to reconfigure a context dynamically (i.e. by adding more machines to the “parallel application server” mode), or to change context at runtime (i.e. by switching from the “single server” to the “parallel”). The intention of the DSB is to optimize the web serving architecture for the requirements of the application while taking into account the hardware dedicated to the web application. This approach of software-based scaling can make effective use of previous generation commodity machines rather than necessitating expensive server hardware to gain application performance and load handling ability [Bre98].

# Atlas



Network Context	Web Requests	Hardware Requirements	Network Conditions	Benefits
<b>Single Server</b>	As thread within web server	Single web server	Low to moderate web server and Atlas load	Minimize communication overhead by running in a single process
<b>Web and Application Servers</b>	Separate process on separate machine	Web server and application server	High web server and moderate Atlas load	Off-loads the web server
<b>Parallel Application Servers</b>	Multiple processes on multiple machines	Web server, multiple application servers, and database server	High Atlas load	Exploit parallel nature of http requests
<b>Applet</b>	Within an applet in the client's web browser	Single web server	High bandwidth to web browser	Minimize communication between web client and server

**Figure 4: Distributed Servlet Broker Network Context Modes**

## 4.2 Technology

Designing and building a modifiable, extendable, and performance oriented application necessitates the aggressive use of new technologies such as Java Servlets [Srv] and Aspect Oriented Programming (AOP) [KLM+97]. The use of these technologies was largely in response to the problems identified by the original C++ Atlas implementation which used a custom abstraction layer over the CGI interface that permitted Atlas to run as a persistent application while taking web browser requests in the form of CGI requests. This approach worked almost identically to Java Servlets, but developed before the Servlet specification was released. It was dropped in favor of the more sophisticated Servlet. Since Servlets are only supported in the Java programming language, a move from C++ to Java was made. The change in languages was also encouraged by the development benefits associated with the Java language [GJS96].

The distribution layer in the original C++ implementation was a remote method call layer built on top of a UNIX IPC layer. Bugs and problems with this implementation were particularly problematic in terms of development because they involved the challenging task of distributed debugging. A much simpler solution to distribution was used in this implementation, relying on Objectspace's Voyager [Obj]. In order to capture some of the specific issues that were challenging in designing and implementing Atlas the new paradigm of AOP was employed. For the object-oriented portions of the system pattern-based design was used.

### 4.2.1 Java and Servlets

Java Servlets are a server-side Java interface intended for creating dynamic web applications. Servlets are analogous to CGI programs, but take an object-oriented approach and differentiate themselves by existing as persistent objects in the web server, rather than being executed as separate programs in the fashion of CGIs. A Servlet is a Java class that implements the `javax.servlet.Servlet` interface, meaning that it provides a method that will be executed by the web server when a request to the Servlet is made. Any parameters associated with the request (passed with the `GET` or `SET` mechanisms in CGI programs) are packaged into a request object provided to the method, while any HTTP response-related parameters are placed in a response object. Servlets offered an very significant improvement over the CGI method due to both their performance increase that results from running as threads rather than processes, and by

providing a more rationalized object-oriented paradigm for building the server-side objects. They also facilitate sophisticated applications by providing the information hiding and abstraction principles that are a benefit of Object-Oriented programming. For example, Atlas/DSB is a complex application (200 class) wrapped by a single Servlet interface.

We have noticed numerous benefits developing with Java over C++. The most significant of these benefits experienced in the development of Atlas/DSB involved the ease of debugging and lack of problematic runtime errors (such as “array index out of bounds”, pointer related, or memory reference related errors). The strict object-oriented nature of Java encouraged a careful design leading to a more modular structure. Java “packages” facilitated this modularity and were used extensively.

#### **4.2.2 ObjectSpace Voyager**

Numerous different technologies were evaluated for distributing objects and requests in the Atlas/DSB environment. The customized approach of the original C++ implementation was dropped in favor of something more robust. Java Remote Method Invocation (RMI) [Rmi] was a prime candidate due to its integration with the Java language, but was found to be unnecessarily complex in moving regular objects to function in a distributed mode, and had a discouraging performance characteristic. OMG’s CORBA [Cor] approach was more complicated to set up and implement with than Java RMI, and offered little benefit other than compatibility. Since Atlas/DSB did not need to inter-operate with other applications compatibility was not a requirement. Instead, a much more streamlined and transparent approach, offered by Objectspace Voyager [Obj] was used. The Voyager application allows the distribution of Java objects by employing an object request broker to which objects are bound. These objects can then be used by another Java program if that program specifies a socket connection linking its own Voyager object request broker to the one containing the remote objects. This process only involves a few custom lines of code, and permits almost completely transparent use of remote objects. Voyager makes use of the built-in features of the Java language, such as interfaces, in order to minimize the ease of using and migrating distributed objects, and resulted in much more understandable distributed code that does not suffer a performance hit.

### 4.2.3 AOP and Patterns

Atlas is composed of a sizeable code base, and as such requires special care in the design stage in order for it to remain understandable and modifiable. “Gang of Four” design patterns [GHJV95] were aggressively used in order to provide commonly understood and easily extendible abstractions within the application’s architecture. For example, the process of building web pages took the form of a “Builder” pattern, with concrete builders for the “course” and “desktop” views among others. Another instance is the “Chain of Responsibility” pattern that was used for passing around Servlet requests to be manipulated by different handlers such as a web page rendering handler and a database update handler. The patterns were particularly useful for documenting parts of the system architecture and for providing a convenient means of both locating the positions at which classes and packages would have to be added to extend the application’s functionality and providing a tested and reliable infrastructure for those packages.

AOP is a new programming methodology that allows the design and implementation of the program to capture cross-cutting concerns [KLM+97]. Cross-cutting concerns are those that touch numerous places in the object model, and as such can not be modularized as classes. An example of one such concern is the debugging code inserted during the development of a distributed system such as Atlas/DSB. Lines of code are often added to methods of classes in order to output the signatures and parameter values of those methods. Removing this tracing code is tedious since it is spread throughout the system’s classes. The tracing concern can be captured as a single aspect, and then be ‘weaved’ into all the methods and classes in the system that require tracing. This modularization allows the tracing concern to be extended, say to handle a GUI that presents traces from several nodes in a distributed system. Such a tracing module was developed and used for building Atlas/DSB. Currently AOP programming is done with an extension to the Java programming environment called AspectJ [LK95].

The major benefit of using AOP for Atlas/DSB was the modularization of the DSB “network contexts” as aspects of the system depicted in Figure 4. This allowed Atlas to be implemented and tested without concern for the distributed and parallel nature of the application. In other words, the application development was greatly simplified by focusing explicitly on the “single server” context and ignoring distribution-related issues. Aspects were then used to extend the application functionality to work in a distributed context, addressing issues such as context sensitive variables such as file handles, the distributed objects used through Voyager, and

concurrency issues. Separating the network configuration and distribution concerns into a module and allowing Atlas to be de-coupled from that module and run independently in a simpler default context was the most significant method used to accelerate the development of the application.

### 4.3 Atlas/DSB Design and Implementation

The Atlas requirements of dynamic user-based course planning, dynamic web interface, and performance necessitated a series of design and implementation decisions. The key decisions are focused on the DSB which is responsible for application performance, the `webObjects` library responsible for making object representations of HTML pages, and the `courseTree` knowledge base responsible for course navigation.

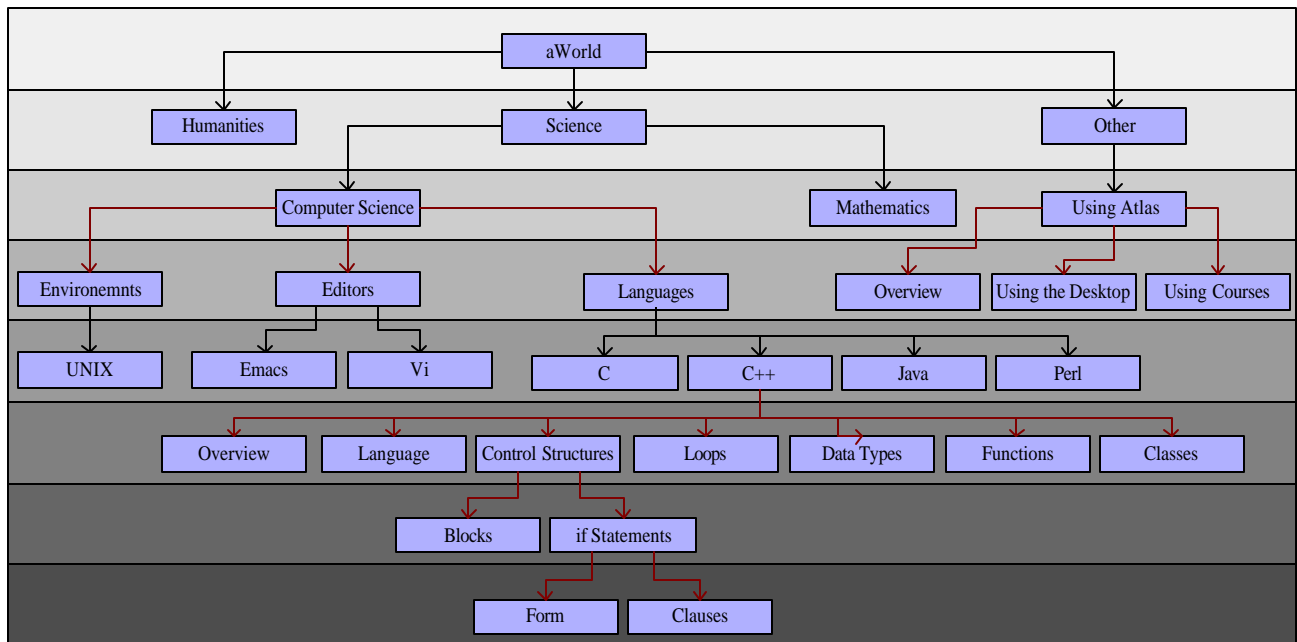
#### 4.3.1 `courseTree` Implementation

User-based course planning is achieved by two means: the `courseTree` knowledge base represents the format of the courses in a way that enables the specified means of navigation, and a course planning component makes up a portion of the Atlas `courseManager` package responsible for retrieving course contents. The `courseTree` knowledge base is implemented as a directory structure rather than a database in order to simplify modifications and permit offline manipulation. Each node in `courseTree` is represented as a directory, and contains the contents associated with the node as well as any child directories. The child directories can take one of two forms: those that are ordered and those that are unordered. Ordered children impose a total ordering on the navigation order through the children, and associate pre-conditions with given nodes. For example, in Figure 5 the “Overview” child of the “Using Atlas” node must be completed before the “Using the Desktop” node is reached, and as such its completion is a pre-condition to the “Using the Desktop” node.

The `courseManager` component uses a Strategy “Gang of Four” pattern [GHJV95] in order to plan the user’s traversal through the tree. Two strategies are currently implemented:

- *Linear Strategy*: assuming a root course node (i.e. “C++”) does a depth-first traversal presenting the contents of each node when traversing with the direction of the arrow Figure 5, or ‘down’ the tree, and presenting the test material when an traversing against the direction of the arrow, or ‘up’ the tree.

- Planning Strategy:* first, assess user’s requirements, pervious knowledge, and aptitude (with a questionnaire and related tests). Next, determine root node of user’s course (i.e. if the want to learn about C++ select the “C++” node). Finally, determine if they have satisfied any pre-conditions necessary for them to have traversed to that node (i.e. for “C++” “Computer Science->Environemnts” and “Computer Science->Editors” must have been completed) from either aptitude tests or information stored about pervious traversals through `courseTree`. If they have not satisfied all of the pre-conditions proceed to traverse the tree from the first node in the list of pre-conditions node and stop when the goal node is reached (i.e. the C++ node in the C++ course). Proceed to traverse the tree until the root node is completed. Note that this strategy is being extended with functionality that is able to prune leaf nodes based on the user’s learning ability in order to speed up traversals of familiar pre-condition material.



**Figure 5: Partial `courseTree` Snapshot**

The above strategies take as parameters the user’s current position in the `courseTree` tree and the user model and return a corresponding node. The node is then rendered with the `webObjects` component which reads the contents of the node from disk. Extending the planning abilities of Atlas is facilitated by the “Strategy” pattern. The developer need only sub-class the

superclass associated with traversal strategies in order to implement a more sophisticated or better suited strategy.

### **4.3.2 HTML Generation With webObjects**

In order to support the server-side dynamic HTML generation requirements of Atlas, web pages needed to be generated using an object model rather than the string manipulation that was done in the C++ version of Atlas. The `webObjects` library was constructed in order to support this functionality and is used to create object representations of HTML pages. The UML [BJR98] representation of the classes associated with the `webObjects` library are presented in Figure 6. The example of one such page is shown in Figure 7, where the same page is presented in HTML rendered in the web browser, standard HTML, and with the according object model.

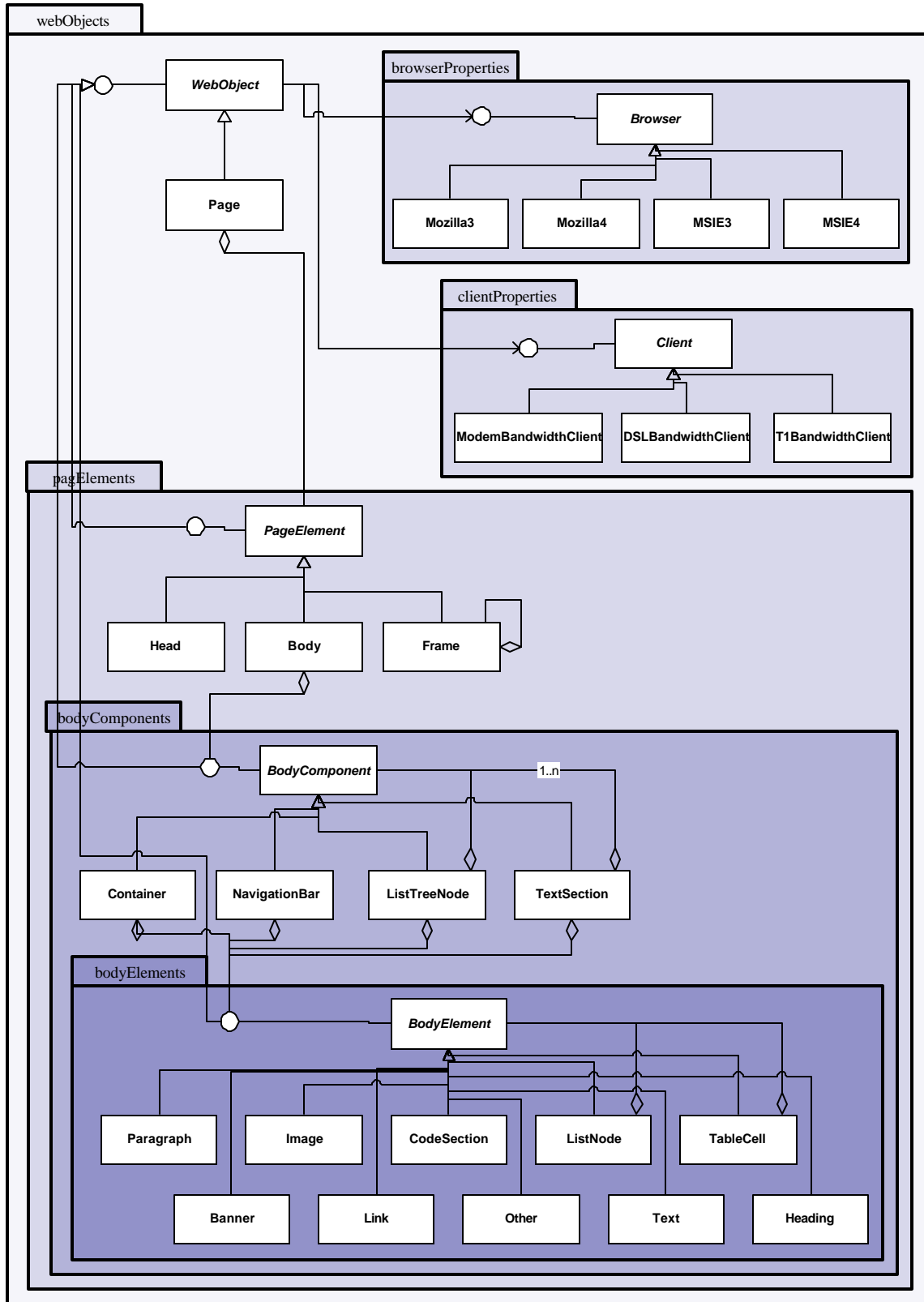


Figure 6: UML of webObjects static structure

## Atlas

The `webObjects` library offers several benefits over the use of HTML strings or files for representing dynamically generated web pages:

- Modularizing at the `bodyElements` level permits convenient manipulation of commonly used elements such as lists and tables. These elements are easier to create with an API than they are with raw HTML since default attributes and layout characteristics can be assumed, and because an API results in code that is more readable than string manipulation.
- At the `bodyComponents` level commonly used structures can be re-used and stylistic attributes can be captured. These components are often re-used in web pages. By modularizing them attributes such as layout and look-and-feel can be controlled, and code duplication is minimized.
- At the `pageElements` level properties of HTML pages such as scripts and frame sets can be conveniently manipulated with the provided API.
- Aspects (not shown in Figure 6) can be used to extend the look-and-feel attributes of the web-page object model in order to encapsulate the look-and-feel attributes of the entire site, or of the pages generated for a particular user.

The printing methods that must be implemented by every class in the `webObjects` hierarchy must also take as parameters the objects part of the `browserProperties` and `clientProperties` packages in order to print HTML that is tailored to the destination browser and any client settings such as bandwidth requirements.

Another benefit that results from this modularization is concerned with sending dynamically generated web pages to the browser. Conventionally, a page is constructed as a string and printed to the output stream associated with the browser. However, the `webObjects` library avoids using strings for web pages, and allows each element associated with a page to print itself to the browser directly when it is rendered. When a method that requests printing is invoked on the top level `Page` a reference to the browser output stream is passed as a parameter, and the `Page` object prints its contents to the browser by asking each contained object to print itself, forming a transitive closure of the references in the structure associated with the page. Since this avoids any strings being printed it results in increased efficiency.

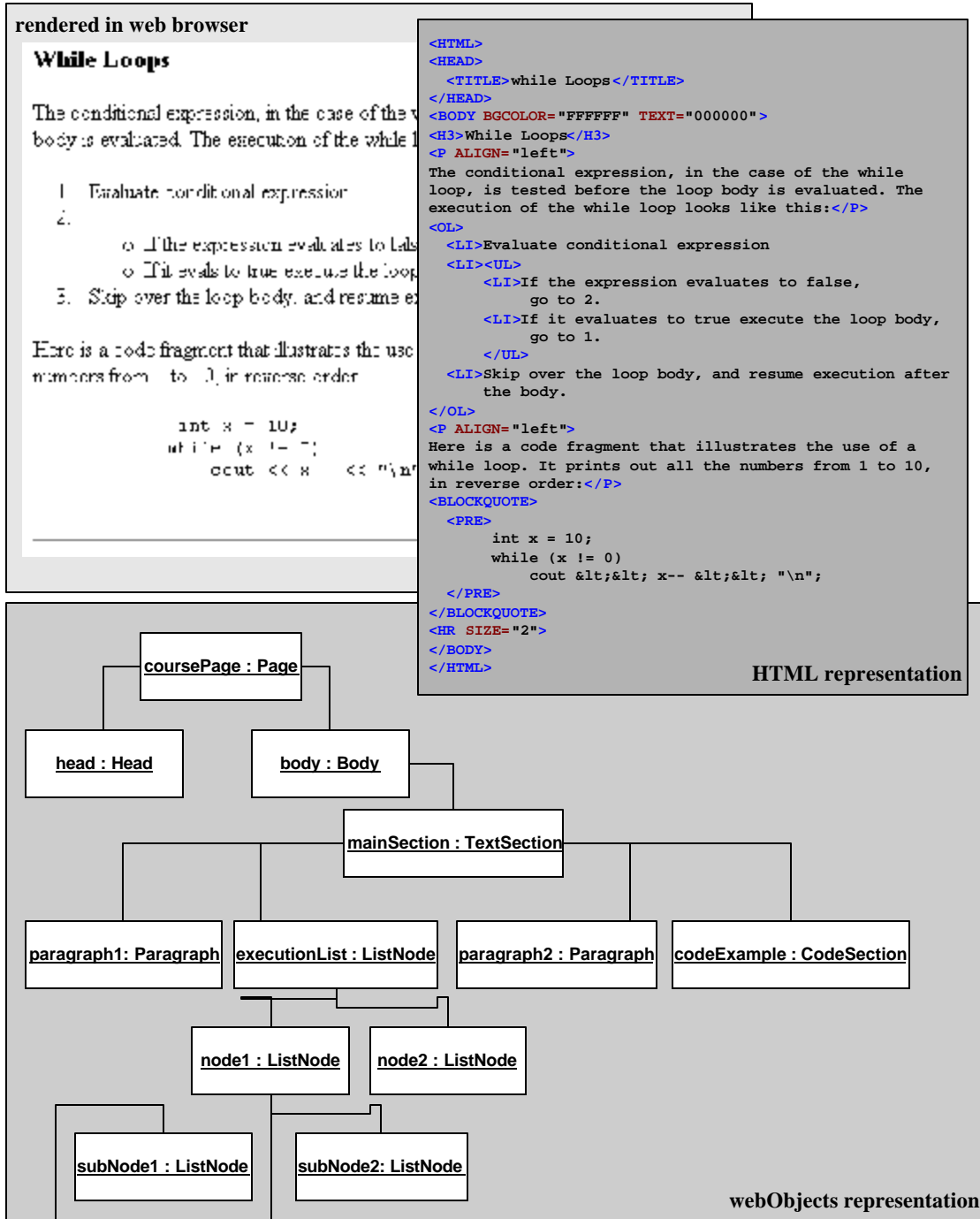


Figure 7: Three Views of a Page Made with webObjects

### 4.3.3 DSB Component Distribution

The Distributed Servlet Broker is the distribution and network configuration framework for Atlas, and is responsible for ensuring that Atlas runs in a configuration that is optimal given the associated load and hardware availability. The use of AOP [KM99] has permitted the complete

## Atlas

de-coupling of Atlas from the DSB framework, whereas very tight coupling was present in the original implementation. The result is that Atlas is treated as a component distributed by the DSB. In Figure 8, the Atlas component is referred to as the `AtlasService`, and is shown in two of the four network configuration modes (note that the database component is referred to as `DSBNexus`).

The `DSBClient` is the component that handles the requests passed to the web browser. These requests are then delegated to the `DSBServer` by means of a registry that performs load balancing if necessary. The `DSBServer` delegates the request to the `AtlasService` if that was the service requested, and the `AtlasService` employs the database contained in `DSBNexus` (`DSBNexus` is transparent to remote use by means of distribution aspects [KM99]). The component-based implementation of the DSB has not only permitted performance optimizations for the Atlas application, but also allows other applications to make use of the DSB architecture. The `AtlasService` component only distinguishes itself by implementing the Servlet interface, and therefore any Servlet can be distributed with the DSB (although it would be restricted to using a database contained in `DSBNexus` in the “parallel server” mode). Since the `AtlasService` component implements the Servlet interface it can be used directly in the web server as any other Servlet, and can thus function without the presence of the DSB.

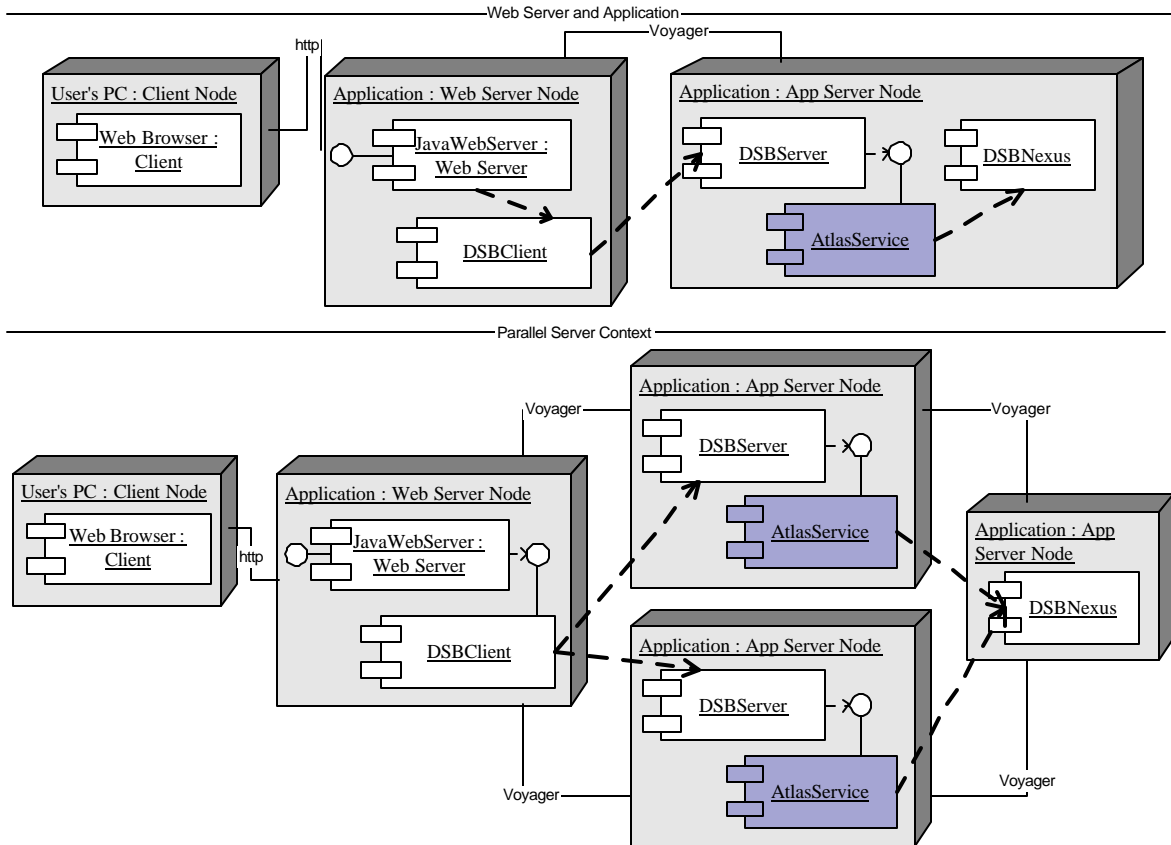


Figure 8: DSB UML Component Diagram

## 5 Discussion

### 5.1 Performance Evaluation

In the early stages of development, the use of new technology and the Java language resulted in serious concern regarding the performance of Atlas. However, a careful design of the web page generating architecture, the optimizations resulting from the DSB, and the nature of the Servlet technology resulted in an application performance that far exceeded expectations. Although actual metrics are not available at this time, rudimental tests showed Atlas/DSB to be an order of magnitude faster than comparable C and C++ CGI-based applications, and two orders of magnitude faster than comparable Perl-CGI applications. The tests were performed with Atlas running as a “pure” Java application on the Java Web Server [Jws] under Microsoft Windows NT™. The performance increase is largely attributed to the fact that Servlets are spawned as threads within the web-server process, and thus do not require significant state to be initialized at request time, unlike CGI-based approaches.

## 5.2 Usability

Atlas/DSB is currently in an alpha stage of testing, is in the process of being released to the public. As such, results are not yet available from the usability of the application. However, the quality and configurability of the interface, along with the performance of the application, has resulted in positive user responses. Future work will be based on collecting more user comments and tailoring the application to better suit its audience.

## 5.3 Future Work

The majority of the Atlas development to date has focused on implementing an architecture that would fulfill the immediate requirements and scale well to future uses. The implementation of extensions to the functionality presented here can be made at a fraction of the cost of the original implementation, and will result in an application that is much more usable and useful. Some of the essential focuses for future work are as follows:

- *Implementing better course traversal strategies:* the `courseTree` tree specification was designed with an additional feature, the exploitation of emergent cross-cutting associations, that is not currently being used. This feature relies on each node being linked to a general knowledge tree (for example, the C++ “while loops” section is linked to the “imperative languages loop-test on top” section). This sort of linking in turn allows associations to emerge between relevant nodes, and these associations can be incorporated into courses. For example, if a user expressed interest in learning both the Java and C++ programming languages, they could be prompted to view the syntax for Java’s “for loops” after they had completed the according C++ node.
- *Providing course administration and course authoring tools:* currently courses are translated into the `courseTree` format manually with the aid of some rudimentary tools. However, since the format imposes a strict hierarchy on the course, it may be useful to adopt as part of an authoring tool. Automatic converters from other course formats also need to be developed since the quality of the Atlas application depends on the size of the `courseTree` knowledge base.

## **6 Summary**

Although the Atlas project is still young and has not been tested with a significant user base, it shows promise as both a novel paradigm for course serving and as an inexpensive, high-performance web serving architecture. By using a familiar desktop-base interface that is configurable by the user, ease of use is enhanced. Adopting a dynamic notion of a course that tailors itself to changing user needs leads to courses that are better suited to the user. Planning the user's traversal through a course enables the concept of passing a course, which can facilitate online instruction. The DSB approach to serving architecture both enables the computation required by the interactivity of Atlas and permits the application to handle a large user load and future scalability needs without necessitating expensive hardware.

## **Acknowledgements**

Gail Murphy contributed to the development of the DSB architecture and to the use of Design Patterns and AOP techniques in developing Atlas/DSB.

## References

- [BJR98] G. Booch, I. Jacobson and J. Rumbaugh. *The Unified Modeling Language User Guide*. Addison-Wesley, 1998.
- [Bre98] E.A. Brewer. Delivering High Availability for Inktomi Search Engines. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, p. 538, 1998.
- [Cgi] The CGI Specification web page: <http://hoohoo.ncsa.uiuc.edu/cgi/interface.html>
- [Cor] OMG CORBA home page: <http://www.corba.org>
- [Css] Cascading Style Sheets level 1 web page: <http://www.w3.org/TR/REC-CSS1>
- [GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Publishing Company, 1995.
- [GJS96] J. Gosling, B. Joy, and G. Steele. *The Java Language Specification*. Addison-Wesley, Reading, Massachusetts, 1996.
- [Htm] HTML 4.0 Specification web page: <http://www.w3.org/TR/REC-html40>
- [Ja] Java Applets web page: <http://java.sun.com/applets/index.html>
- [Js] JavaScript 1.1 Language Specification: <http://home.netscape.com/eng/javascript>
- [Jws] Java Web Server web page: <http://www.sun.com/software/jwebserver/index.html>
- [KM99] Mik A. Kersten and Gail C. Murphy. "Atlas: A Case Study in Building a Web-based Learning Environment using Aspect-oriented Programming", Technical Report TR-99-04. Submitted for Publication.
- [KLM+97] G. Kiczales, J.Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J.Irwin. Aspect-oriented programming. In *Proceedings of the 11<sup>th</sup> European Conference on Object-oriented Programming (ECOOP'97)*, pp. 220-242, 1997.
- [LK95] C. Lopes and G. Kiczales. Recent Developments in AspectJ™. In *ECOOP '98 Workshop Reader*, 1998.
- [Rmi] Java Remote Method Invocation web page: <http://java.sun.com/products/jdk/rmi>
- [Wct] WebCT web page: <http://www.webct.com>
- [Srv] Java Servlets web page: <http://java.sun.com/products/servlet/index.html>
- [Obj] ObjectSpace web page: <http://www.objectspace.com/Products/voyager1.htm>

## Contact Information

Ian Cavers  
Dept. of Computer Science, UBC  
201-2366 Main Mall  
Vancouver BC CANADA V6T 1Z4  
Voice: (604) 822-4327  
Fax: (604) 822-5485  
E-mail: [cavers@cs.ubc.ca](mailto:cavers@cs.ubc.ca)

Mik Kersten  
Dept. of Computer Science, UBC  
201-2366 Main Mall  
Vancouver, BC CANADA V6T 1Z4  
Voice: (604) 822-2964  
Fax: (604) 822-5485  
E-mail: [mkersten@cs.ubc.ca](mailto:mkersten@cs.ubc.ca)

George Tsiknis  
Dept. of Computer Science, UBC  
201-2366 Main Mall  
Vancouver BC CANADA V6T 1Z4  
Voice: (604) 822-2930  
Fax: (604) 822-5485  
E-mail: [tsiknis@cs.ubc.ca](mailto:tsiknis@cs.ubc.ca)