

Aggregating Documents in SWiM

Zdravko Beykov

*Computer Science
Jacobs University Bremen
Campus Ring 1
28759 Bremen
Germany*

*Type: Guided Research Thesis
Date: May 9, 2008
Supervisor: Prof. Michael Kohlhase*

0. Executive Summary

Nowadays, we are living in a world where information is all around us. The internet is giving us the opportunity to explore a tremendous amount of data. Much of our difficulty lies in how to locate this data and with the advancement of technology this proves no easy task. Even when one finds the place where the information he searches for resides, he might not be able to narrow it down. Even if he succeeds in narrowing it down, the problem of understanding the idea closely linked to this information could arise. If some terms used in the explanation of the information are not present, then the user has to go through the process of locating each unknown node of this dependency separately and this would cost him a lot of time. The idea of the project is to ease the quest of finding the relevant data one searches for and include all the dependencies of the document that the user is unfamiliar with before the main idea. This whole info would be exported to a PDF file that would be ready for printing. Thus, the end product would be nothing more and nothing less than what one needs for understanding a certain topic.

1. Introduction and Motivation

The project would ease the extraction of printable data from a mathematical document collection in the semantic markup language OMDoc (3). It will be embedded into the SWiM wiki (4) as a Flash Applet and would let the user choose the dependencies that would have to be included in the final PDF, which would result from an XSL transformation (3). This project could be helpful for students learning a certain topic by filling gaps of their knowledge of prerequisite information.

This guided research would answer the question of how to utilize extracted mathematical information from the semantic wiki SWiM by including all the needed dependencies nodes into a single PDF file. Also, it will address the logical interconnection between semantic documents in order to get only relevant to the user data. This would be convenient, since in the end the user would have a complete PDF with only the needed information. It will be of great help for the learning of the material. The project would rely on several outside resources. The first one is the OMDoc-HTML XSL style sheet that would be used for displaying the selected nodes in HTML. Currently, this resource is available. The next one would be the OMDoc-TeX in the new XSL 2.0 format. It will be needed for converting all the dependencies into a simple TeX file that would later be transformed into PDF via LaTeX (3). The newest version of this file is not yet ready and therefore it will be worked on to the needed extent. The last needed resource would be the complete OMDoc version of the General Computer Science lecture notes used in Jacobs University for the course. Currently, they are available for the first part of the course, and soon the second part will be ready too.

2. Overview of technologies

2.1. XML

XML stands for Extensible Markup Language and is a general markup language that can be customized to serve a specific purpose. The structure of an XML document is quite simple syntactically – it consists of starting and ending tags of an arbitrary name that can surround other tags and text. Optionally, each tag can have certain properties defined. The syntax is the following: `<tag>`, `</tag>`, `<tag/>` - for a starting tag, ending tag and an empty tag respectively. Additionally, tags can have several properties which are declared in the following manner `<tag propertyName="propertyValue"/>`. The simplicity of the XML syntax allows for a big degree of freedom when declaring custom documents. Taking this one step further, the Document Type Definition (DTD) text file format was made. It has its own syntax to define a set of tags and their structure which are allowed in a certain XML document. In order to evaluate an XML document to a specific DTD, one should include a reference in the beginning of the XML in the `<!DOCTYPE>` tag which ideally should be the beginning of the document. Also, the DTD can define custom variables which are called entities and give a special meaning to their occurrence in an XML document. Their syntax is `&entityName;`. Good examples are the `>` and `<` entities in the context of HTML documents. They correspond to the symbols `>` and `<`, which if included explicitly would break the tags syntax structure and therefore are declared as entities. To facilitate the manipulation of XML in a programming language, standard API's have been developed. The Document Object Model (DOM) has the aim of representing an XML as a collection of native objects of a data structure of type Node. Each tag is such an object and has references to its parent

tag (if any), siblings, and attributes (1). Thus, you can navigate in native code through the whole XML document. The easiness of dealing with XML along with the degree of freedom one has to customize his own XML documents have made it one of the most popular document formats nowadays. It is the basis of many different files like XHTML, SVG, RSS, MathML and others (2). On the other hand, XML does have some disadvantages to other binary coded formats like slow processing speed and large file size. However, binary coded formats cannot be edited as easy as by opening any text editor. In conclusion, XML's advantages outweigh its disadvantages and for a reason it is recommended by the World Wide Web Consortium (2).

2.2. HTML/JSP

HyperText Markup Language (HTML) is a language that is used to format web pages. It is a variant of XML, and has predefined tags that don't need a DTD, as web browser directly recognize them. Each page starts with the <HTML> tag, which should contain a <HEAD> and a <BODY> tag. The first acts like a header for references and definitions, whereas the second contains the content that is displayed. The tags inside <body> are used for formatting purposes. There is also a special type of an HTML page that does not have this tag, that allows the usage of so called “frames” which can subdivide one page into several pages. As HTML is only a markup language, there are no dynamic capabilities within it. In order to add some simple interactivity, the script language JavaScript can be used. However, it is client-executed and limited in functionality. Thus, if one needs to add a complex server side application, like connecting to a database before outputting an HTML page, a technology like JSP should be used. JSP stands for Java Server Pages and it uses the popular multi-platform language Java inside XML tags inside HTML pages. Those

tags are compiled and executed from the server before the page is sent to the user requesting it. Java is associated as slow and multi-platform code is not something of great significance to a server program as it is supposed to run only on that server. Still, JSP is free, it offers a small learning curve to users familiar with Java, and the Just In Time (JIT) compiler speeds up the execution to a great extent. Thus, JSP is one of the popular technologies in its field along with PHP and ASP.NET.

2.3. OMDoc

OMDoc is an XML based open markup format for mathematical documents. It is developed with the goal of web managing the huge amount of mathematical knowledge available nowadays. Unlike other document formats that store only representation of a mathematical object, OMDoc stores its semantic meaning and its relation to other OMDoc documents. Its `<ref>` element is used for reference. Its attribute “type” can have two values (OMDoc v1.2) – “include”, which means that an occurrence should be replaced with the one stated, and “cite” which states a relation to another element and is application specific (3). This tag is of key significance when determining the logical interconnection among OMDoc documents and will be focused on when dealing with finding dependencies in the project applet.

2.4. SWiM

SWiM is a semantic wiki for mathematical knowledge management. It is located at <http://swim.kwarc.info/> and uses a collection OMDoc's in its internal database (4). As a wiki, users can add and modify existing documents through its interface. It is developed in JSP to provide the rich functionality. It also allows mathematical representation of OMDoc documents via XSLT (Extensible Stylesheet Language Transformations). This is an XML format that specifies how to transform XML

documents into other XML documents or even other text documents in general. Although modern browsers allow XSLT transformations, their support is not perfect, and you rarely get identical results on all common browsers (5). Thus, SWiM does these transformations on the server. Another feature of the wiki is that it provides the context applet Ike Wiki that shows the dependencies of different documents among each other. It uses Flash for graphics, and gets the needed dependencies in an XML format processed by a JSP page.

2.5. Adobe Flash

Adobe Flash (previously Macromedia Flash) is a commercial software product that is used to create dynamic and interactive vector animations. You can produce SWF files which are relatively small in size due to internal compression and ready to be integrated in a web page due to the freely available, small in size and now available for many platforms Flash Player. Initially, Flash was more aimed at designers as it provided very limited scripting and many animation tools. However, it quickly became very popular due to a lack of competition on interactive vector animations for web pages. Developers were interested in a more feature rich scripting language, so after two revisions, ActionScript 3.0 is now available. Moreover, Adobe Flex SDK 2.0 allows to fully create SWF files from a command line compiler. Adobe still manages its commercial graphics IDE and doing complex animations is very hard to do only from source code, but for less graphical applets the Flex compiler is more convenient than the IDE from a developer's point of view. It even has the Flex component library which contains a library for GUI components. This compiler is developed in Java and is therefore freely available to for download to many platforms. Still, Flash has some disadvantages as an applet is very easy to

download and decompile directly into ActionScript code and graphics. Thus, big commercial products won't exist in Flash due to its insecurity. Also, the performance is slow due to the needed decompression of the applet, computation of vector graphics, and execution of the script via a virtual machine. However, Flash is yet to find a competition for making free, small and not so intensive graphically applets, due to the ease of creation and wide browser and system support.

3. Contextual Modeling

One of the most important parts of the applet that was to be implemented was the representation of the documents and the dependencies graphically. Intuitively, it can be done by the most common method of illustrating a graph – nodes would be documents and would be 2D geometrical shapes like circles and dependencies would be arrows pointing from node to node. Another approach thought of initially was to use the common Tree View control representation that would have documents that a node depends on as children and could be expanded by clicking on the “+” symbol.

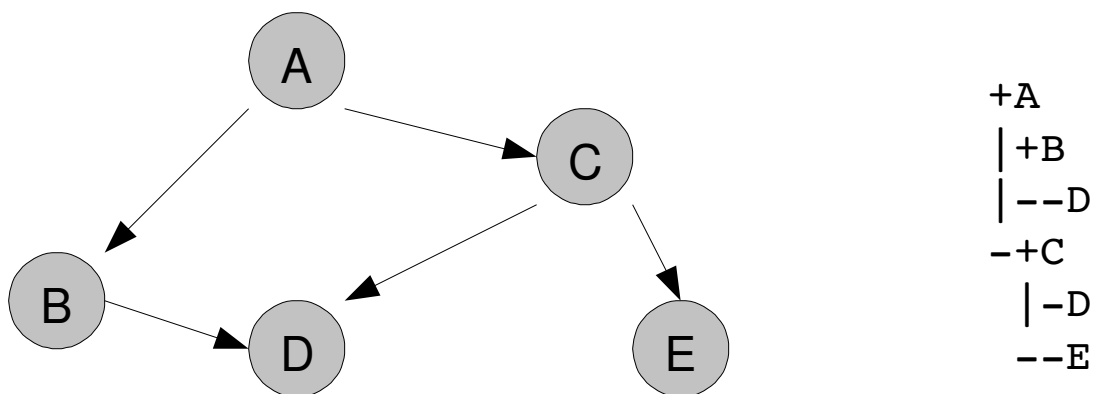


Figure 1: Graph Representation Ideas

However, it is clear from Figure 1, that if several nodes depend on a common other node, like in the example case where both “B” and “C” depend on “D”, the graph is

more suitable since the Tree View would contain “D” twice. As Flash is a web based technology that is suitable for such simple integrated graphics, it was chosen for the development of the applet.

The Ike Wiki that is hosted on SWIM makes a similar representation of dependencies. However, it is not suitable for keeping track of several nodes that are to be aggregated, because visually it spreads all nodes in order to optimize the space provided. For a better logical representation, a tree like graphic would be more suitable – the root is the main document, and as you go down you visit it's dependencies. That way, when you are given the option to include a node in the aggregated document, all the options would be leaves of this tree. Once, you choose a leaf, it goes up one level, and its children (its dependencies) are now leaves and are available for optional inclusion. Unfortunately, as Figure 1. showed, the dependency graph might not be a tree, so a compromise between the two ideas should be reached.

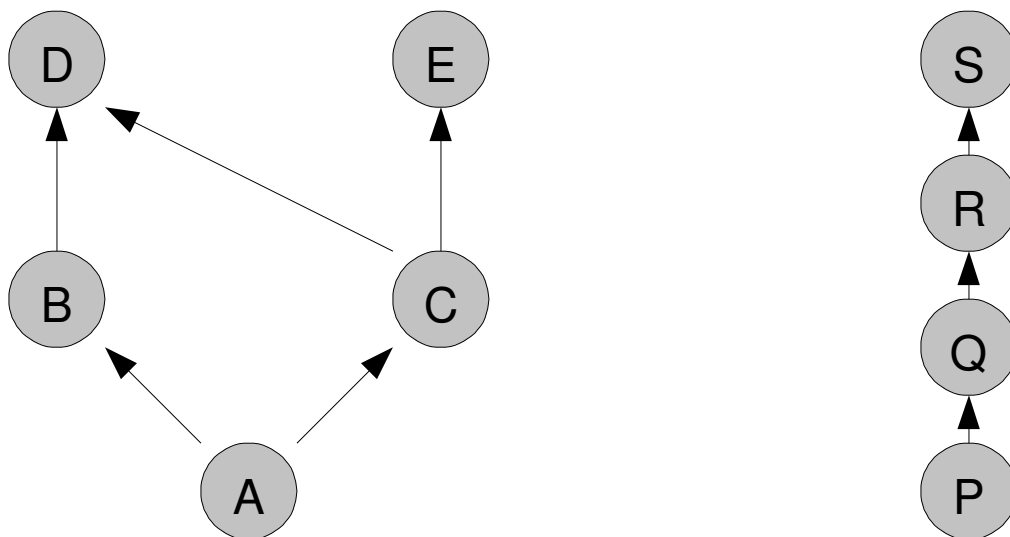


Figure 2: Representation Graph Structure of Aggregated Documents

A somewhat greedy approach of representing the graph would seem good in most cases. It goes as follows – the root is the lowest node, and as you go up, you traverse the dependence tree depth-first. Thus, you get the level at which each node is. For example, in Figure 2, the left case shows “A” as root at level 1, “B” and “C” - its direct dependencies at level 2, etc. Once, you have distributed the nodes equally vertically according to level, you equally distribute them equally horizontally – the more nodes at a certain level, the more crowded these nodes are. As the right example of Figure 2 shows, such a greedy distribution might not be very optimal in cases where each node has only one dependency, and the horizontal space is made minimal use of. However, the logical leveling of nodes similar to a tree representation is kept, keeping the convenience of choosing nodes to aggregate from the leaves of the tree only (which would be situated at the last level). So initially one would start only with the root and its dependencies as leaves and as one chooses one of the leaves their dependencies are now taking their place. Possible overcrowding both horizontally and vertically are possible, but as Flash has direct zoom in/out option, at any time, all nodes would be readable by using it.

Another advantage of choosing Flash for development of the applet is the ease of the integration of I/O functions from code. Input handlers can be directly associated with any graphical object, so selecting a node via the mouse is almost a trivial task to implement. The idea is that the user would be able to also remove nodes from an expanded tree (if he added them by accident). At first glance, two types of input would be needed for addition and removal of nodes. This would be some inconvenience if all input would be based on the mouse as the right click is reserved for the Flash built-in menu, and not all mice have more than two buttons.

However, if one can add nodes that are only leaves in the graph, so it is logical that if the remove is like an undo operation, one can remove nodes that are not leaves. Thus, you only need to left click, and whether you would select or remove a certain node depends on the position of the node. To avoid the mistake of removing a large amount of nodes just by one click to a node near the root, a restriction can be made to be able to click only on leaves (for addition), and nodes one level before the leaves (for removing). To help identify the leaves, the color of the last level nodes will be of different from the rest.

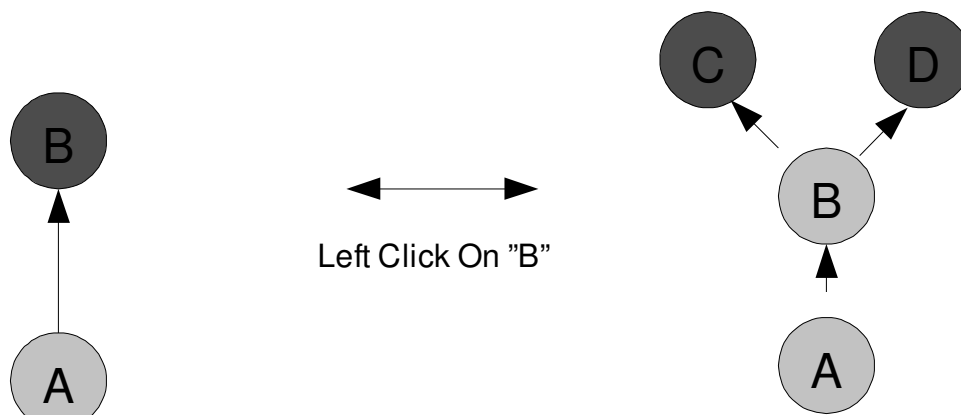


Figure 3: Selecting Nodes

Figure 3 illustrates this idea. On the left, “A” is to be included in the aggregated document whereas “B” is a leaf, so it is not, but can be chosen if left-clicked. If it is chosen, the user gets to the right diagram, where “C” and “D” are the dependencies of “B”, which are not included as they are leaves. The red background distinguishes the nodes which can be included. If the user decides he made a mistake by including “B”, he can click it again to remove it and get back to the left side of Figure 3.

After the user is done selecting the nodes, he can then choose to aggregate

them. To keep the input only mouse based for convenience, a clickable button “Aggregate” would be added. Another addition will be the ability to view the clicked document, as SWiM can XSLT transform an OMDoc to HTML. The applet can be in a window with two frames. The left frame would contain the applet, whereas the right frame will contain the translated document that was last selected. Also, as there are different types of dependencies, the user will need to know which dependency is of which type, so rectangular labels will be added. Finally, to avoid “jumpy” animation, there will be a short motion tween each time there is a change in the state of the graph. Summing it all up, the final decided interface's look is the following:

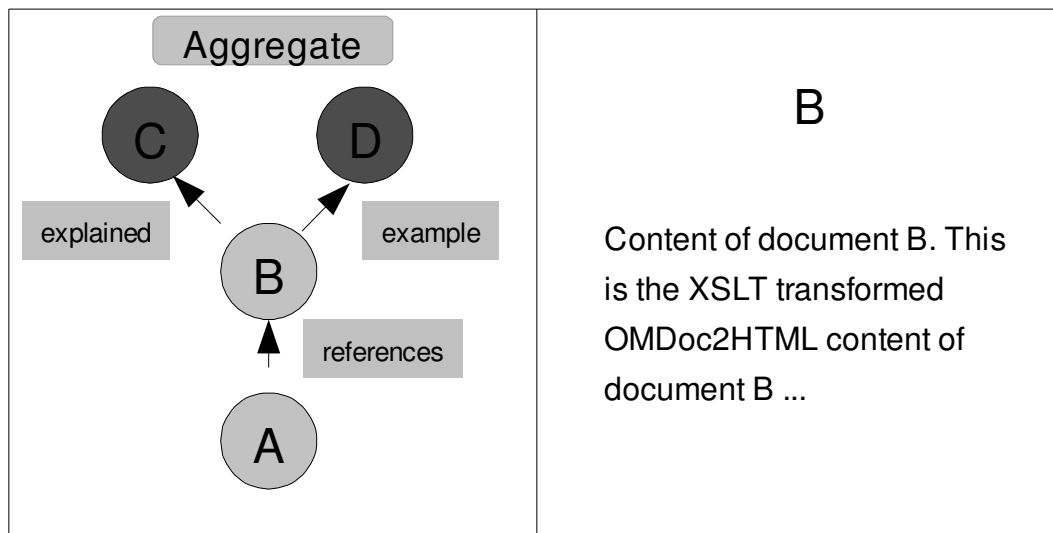


Figure 4: Final Interface Design

The “Aggregate Button” action will linearly combine all of the nodes in the tree, starting from the last level. The idea is to first include the non-dependent nodes, and gradually all of the others to reach the root node at the end. Thus, if the user is to read the document from beginning to end, he will reach the root node only after he has read all of its dependencies (and the dependencies of their dependencies, etc.) The server will then XSLT transform this aggregated OMDoc to TeX, which is a

markup language which can easily be translated to PDF. After that, the user gets the result – one PDF document, containing all the dependencies needed to be familiar with the root document, including the root, ready to be printed and read.

4. Implementation

The implementation of the approach requires coding in ActionScript 3.0 and JSP. Following the structure of SWiM, an additional tab is added, which opens a two-frame window, which will contain the applet on the left and the XSLT-ed document on the right. Due to the internal workings of SWiM, the applet cannot directly access the OMDoc sources, so the dependencies of the nodes had to be passed to the applet via a JSP XML request. Finding all the dependencies once a node is given is a trivial Depth First Search (DFS) via keeping track of already visited nodes (7). In short, the algorithm is a recursive function with a global hash map indicating whether a node (represented as its URI string) was already visited or not. If it was visited, the function returns. If it wasn't, it is appended to the XML response string just like all its dependencies. Next, the function is called on each of its dependent nodes. Another graph algorithm will be used when aggregating the documents into a single OMDoc – Breadth First Search (BFS) (7). This algorithm traverses the visited nodes level by level, and we want to include the nodes by levels in decreasing order in the aggregated document. Thus, we include them in reverse traversal of the BFS and send the URI sequence as an XML message to a JSP server page to get the aggregated document. The same technique of XML communication between JSP and Flash is used in the Ike Wiki. As both Java and ActionScript 3.0 have internal handling of XML via libraries, sending and receiving messages at both ends is not difficult to implement. Not going into detail, the communication is just two messages: one

contains all nodes and their dependencies and is requested by Flash from JSP on creation of the applet. The other message is from Flash to JSP, sending the sequence of OMDoc URI's to be combined. The OMDoc-to-HTML document will be generated by modifying the already existent JSP page context.jsp to allow not only current but also custom context OMDoc to be displayed, and for the OMDoc-to-PDF transformation, a new JSP page will be added.

The majority of implementation is required for the applet. In order to aid its development, several Unified Modeling Language (UML) diagrams have been created.

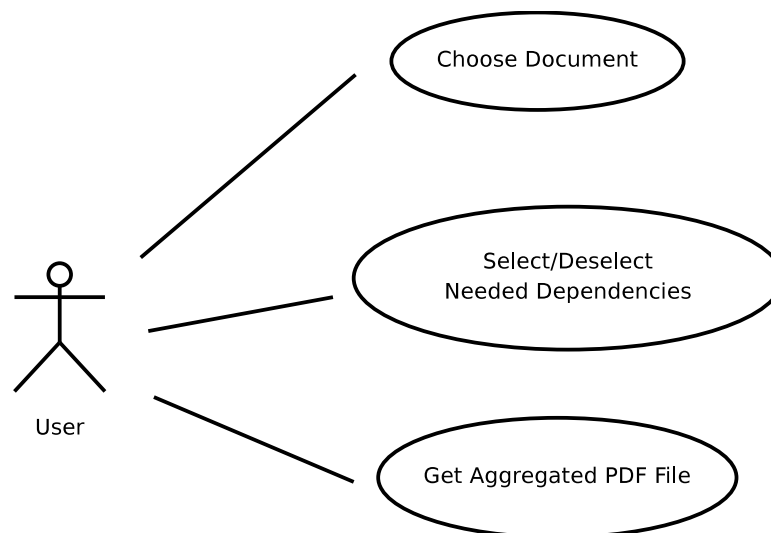


Figure 5: Applet Use Case Diagram

This represents the use case diagram and is self-explanatory. What follows is the more complicated class diagram:

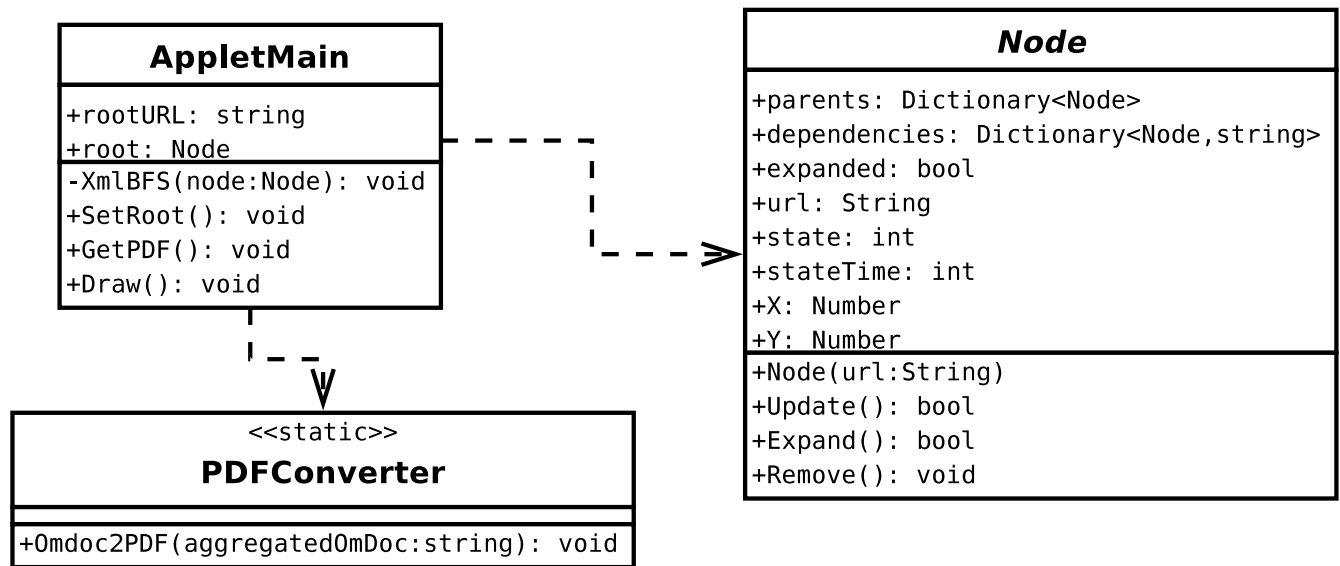


Figure 6: Applet Class Diagram

The PDFConverter is a static class that has only one method that takes the XML aggregated message to be sent to the JSP, which is obtained from AppletMain's XmlBFS function, called when the “Aggregate” button is pressed and GetPDF is called. The graph is represented by a collection of Node objects. Its properties X, Y show their graphical coordinates in the applet, and the state and stateTime variables are needed to keep track of the transition state of the graph. Last, but not least, there's a UML sequence diagram:

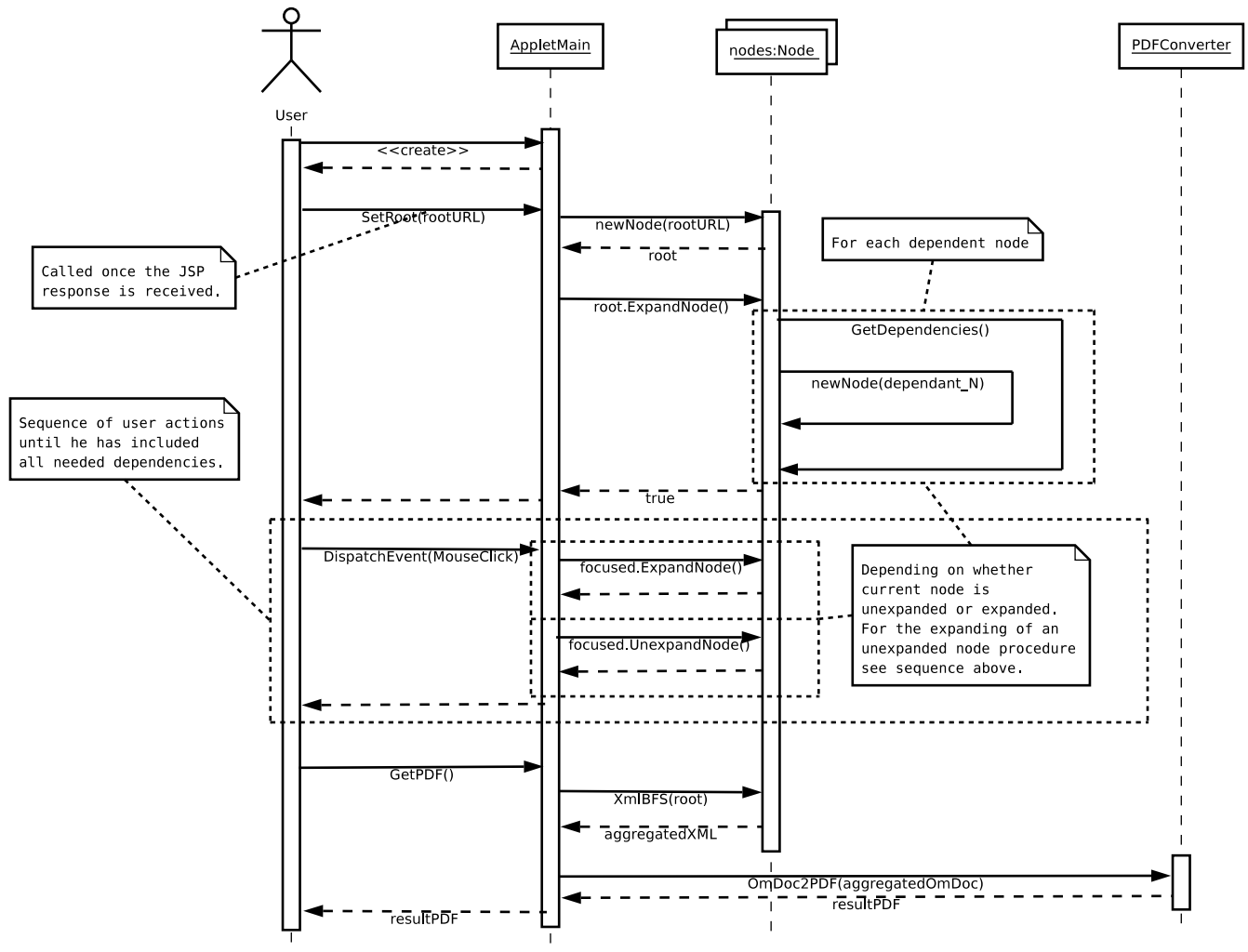


Figure 7: Applet Sequence Diagram

The sequence diagram traces how a typical applet execution will proceed. The system ActionScript 3.0 calls are not included. Drawing of shapes is handled via the Sprite class and is done each time an EnterFrame event is issued. Hierarchical organization of drawn objects are handled using DisplayObjectContainers, and input is handled via MouseClick events.

5. Critical Evaluation

Initially, the design of the applet was different. Before getting to know the internal workings of SWiM, it was planned that an absolute URL of an OMDoc document can be obtained, and the Flash applet can calculate its dependencies on its

own through simple XML tag/attribute checks. However, as it turned out, the OMDoc collection is only internally visible, and the dependency calculation had to be done via a SWiM's JSP. Additionally, it was expected that since most common browsers support client side XSL transformations the server wouldn't have to bother with it. However, the browser's support for that isn't perfect, so the server does it. Ideally, it was thought that the server wouldn't have to do much – just pass the XML link, and the applet/browser would handle almost everything. Thus, if in the future SWiM's internal structure changes to expose the OMDoc sources and the browsers have standard and identical support for XSL to HTML transformation, almost all of the work can be done client side, so the wiki server would get minimal load. This is very much needed in case the wiki gets a high amount of traffic, something that, as one of its developers, I am hoping for.

Over the process of designing and implementing the applet, I've learned many things about the new internet technologies, especially Flash and JSP. Flash and JSP can seamlessly work together using XML messages. Also, both are free and multi-platform. However, during my research I found two technologies that might be more suitable for the job in the future– Microsoft's .NET ASP and Silverlight. The first is a substitute for JSP, whereas the second is thought by some as the next Flash, as it has the same functionality, but better programming support and faster execution. The surprising thing is that both of these technologies can be developed in a single programming language – C#. This is Microsoft's answer to Java, and claims to be a better version of Sun's language. Thus, nothing can be sure except the web technologies are to undergo the biggest developments in the near future.

6. Future Work

As the aggregation applet is done, the PDF conversion still needs development along with integration with the applet. In the next week most probably it will be done to finalize the implementation part of the project. However, if tweaking is needed for one reason or another later on, be it user preferences or change of the SWiM's internals, I am willing to modify the applet for the better development of the semantic wiki. Moreover, I am keeping the source of the applet open, and anyone would be able to take a look at it for educational purposes or even modify it, as long as he gives me the proper credit.

7. Conclusion

Being at the end of a creation of the applet, I am looking forward to the practical usage of it by the future wiki users. The implementation process was an educational journey for me and I am hoping that it will inspire further extending of its ideas by other developers. For now, the applet is staying <http://swim.kwarc.info/> and probably will undergo minor changes in the following weeks.

As for the future of semantic wikis, it looks bright. Wikipedia, the worlds largest wiki is the site that is globally ranked at number 7 by amount of traffic it gets (8). If it had all its mathematical knowledge in a semantic format it could only further benefit. Learning from that site would be eased a lot. Combining that with an applet to extract and aggregate dependencies, one would bother only with material he needs in order to understand a given topic, and find it very fast and easy. Thus, a semantic web would ease assimilating information through such wikis and applets to a great extent. Therefore, I am expecting the traffic to SWiM to increase significantly, once it has enough content.

8. References:

- (1) Refsnes Data (2008). *XML DOM – Node Object*. Retrieved 9 May 2008 from http://www.w3schools.com/dom/dom_node.asp
- (2) Wikipedia (2008). *XML*. Retrieved 9 May 2008 from <http://en.wikipedia.org/w/index.php?title=XML&oldid=210753931>
- (3) Michael Kohlhase. *OMDoc – An open markup format for mathematical documents*. [Version 1.2]. Number 4180 in LNAI. Springer Verlag, 2006
- (4) Christoph Lange. *SWiM A Semantic Wiki for Mathematical Knowledge Management*. Technical Report. Jacobs University. <http://kwarc.info/projects/swim/pubs/tr-swim.pdf> 2007
- (5) Wikipedia (2008). *XSL Transformations*. Retrieved 9 May 2008 from http://en.wikipedia.org/w/index.php?title=XSL_Transformations&oldid=208205834
- (6) Wikipedia (2008). *Adobe Flash*. Retrieved 9 May 2008 from http://en.wikipedia.org/w/index.php?title=Adobe_Flash&oldid=211278344
- (7) Preslav Nakov, *Programiranje++ Algoritmi*; 3rd Edition. ISBN 954-8905-06-X. Sofia 2005
- (8) Alexa Internet (2008). *Global Top 500*. Retrieved 9 May 2008 from http://www.alexa.com/site/ds/top_sites?ts_mode=global&lang=none.