Data synchronization for XML-DOM based collaborative applications

Advised by:
Dr. Henning Schulzrinne, Department of Computer Science,
Columbia University,
M/S 0401, 1214 Amsterdam Avenue
New York, NY 10027-7003
USA

Submitted by:
Govindakrishnan Kannan, Department of Computer Science,
University Of Kentucky,
Lexington, KY - 40508
USA.
e-mail: gkann2@cs.uky.edu

1. Abstract

The project provides a pragmatic solution to synchronize XML-data between applications that are collaborative in nature. This solution is intended for the applications that model data internally, in a Document Object Model (DOM).

2. Introduction

The basic requirement of the project is to demonstrate the synchronization of data between collaborative applications like shared-editors, shared-whiteboard, etc. that model their data internally as Document Object Model(DOM). Given that, the solution must capture operations normally performed on the DOM easily and suitably convey across to the peer collaborating application. Also the solution maintains a state – by doing a "book-keeping" of all actions related to a particular DOM node, in a session. The state may also be used for synchronizing another collaborating peer that either recovers from a crash (OR) has joined a session later.

3. Related work

Some existing tools that perform the XML differences and resolve them were studied. In particular the format of the XML files, that's generated by the "diff" part, was studied. Two of the tools that gave me better ideas were the MS XML Diff and patch tool, and the Delta toolset.

In particular, I learnt how the tools highlight the differences between XML documents; this lead me to think how a "diff" format for collaborative applications could be specified – for example a particular "node" may frequently be re-organized within the application's DOM, its attribute set may be changed continuously, etc. Since the solution is intended for collaborative applications where the changes (the differences of a shared-board, say) are to be transmitted quickly, efficiently I have modeled the diff internally as a DOM, with specific elements just enough to convey the changes appropriately. I have discussed the same in detail in the following sections.

4. Background

The XML-DOM, which forms the basis of the project, is a way of representing the hierarchical structure of documents. As the W3C puts it – "It's an API for valid HTML and well-formed XML documents. It defines the logical structure of a document and the way a document is accessed and manipulated. ".

5. Architecture

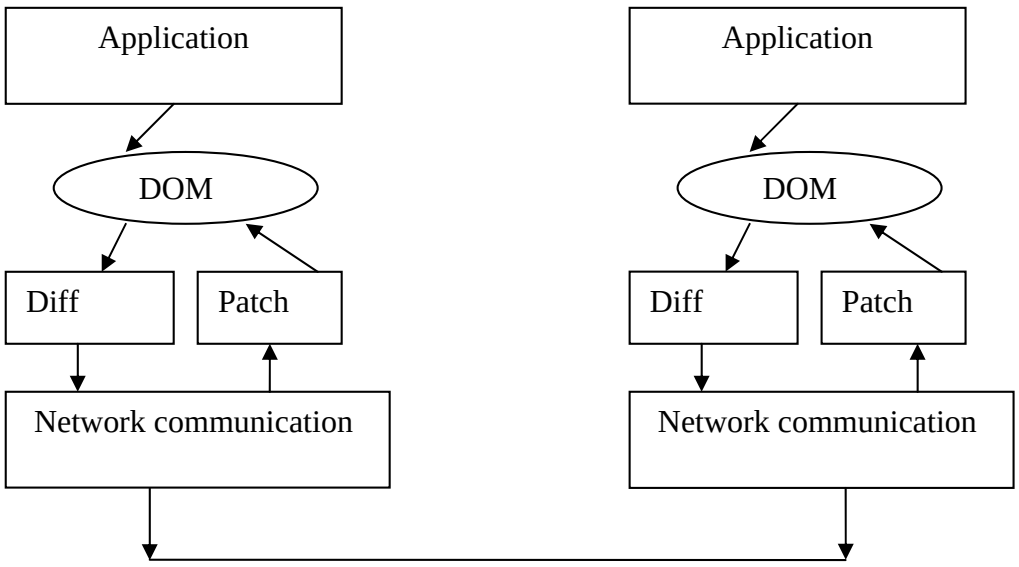
I have modeled the solution mainly having the following modules:

1) A "difference "generating module:
   1.1) this is the module that's delegated with the responsibility of maintaining the state of whatever operations that have been performed on the DOM.
   1.2) Using this "book-keeping" functionality above, the changes are suitably formatted as XML message(s), ready to be dispatched to the peer.

2) A "change resolver" module:
   This is the corresponding complimentary module that's delegated with the responsibility of executing the operations that have been communicated through the XML message(s).

3) A demonstrative usage of these modules, with a simple collaborative drawing program.
   3.1) the modules 1) and 2) are suitably interfaced to a third-party standalone drawing program ("HotSpot") – that has been modified from scratch to simulate collaborative behavior. This is done using TCP/IP sockets at both the sending and receiving ends.

So the entire solution can be picturized as follows:



I would like to emphasize that the modules do not have any knowledge of the underlying model of a collaborating application – for example it's possible that the application has a central server that takes care of routing the messages appropriately to other peers in the collaborating group.

6. Implementation details

1. Programming language and APIs

The entire programming has been done in Java (J2SE 1.4.2 beta) – since J2SE 1.4.1, the JAXP (Java APIs for XML Processing) is bundled with the SDK. The external libraries used are the Apache–Xerces2 APIs that provide the support of DOM event handling, upon which the solution almost exclusively relies on. The "HotSpot" drawing program has also been programmed in Java - whose functionality is extended to simulate the collaborative behavior - has been slightly modified to do away with the deprecated APIs it uses. Network programming is done using Java Socket API.

2. The Java classes:

A brief description of the following classes, that I have implemented are as follows:

a) DiffDomNode

This is where all the changes invoked by DiffListener are performed. It does bulk of the maintenance of the internal DOM maintained for generating the changes to be transmitted.

b) DiffListener

This class implements the MutationEventListener (a W3C Events specification) and identifies the type of operation and conveys them to the DiffDomNode class above.

c) DiifNodeInfo

This class encapsulates some useful information regarding the node that was affected by the application. It's used by DiffListener and XMLDiff.

d) XMLDiff

This is the entry-point to the "diff" module. It takes care of initializing custom- data structures, registering the event-listener (DiffListener) for the "Document" object handed down to it by the application. A Wrapper-function specifically for two DOM operations by name insertBefore (), replaceChild () is provided, which MUST be invoked if the application chooses to use these operations; the reason is outlined in the design alternatives below.

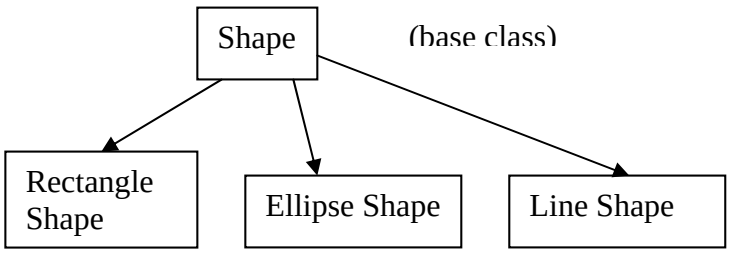
e) PatchListener

This class implements the MutationEventListener (a W3C Events specification) and performs the application specific operations in response to the updates the DOM made by XMLPatch.

f) XMLPatch

This is the entry-point to the "patch" module. It takes care of registering the event-listener (PatchListener) for the Document object handed down to it by the application. This class implements the SAX parsing module that parses the byte-stream sent by another peer. It updates the DOM object handed to it, that are subsequently tracked by PatchListener, and all application specific operations are to be performed here.

Apart from these classes, there are specific classes implemented for the sake of simulating the collaborative behavior of the “HotSpot” application. I provide a brief overview of the organization of the classes.



Each of the above derived classes implement the corresponding shape.

1. HotSpot
   This is what helps to manipulate the geometric object – ellipse, line, rectangle.

2. HotSpotMgr
   The hotspots are managed by this class, whenever the hotpsot for the respective object changes (when resized, or moved).

3. WBDOMMgr
   This class interfaces the diff and patch modules with the HotSpot Drawing program.

7. A brief description on how the “differences” are modeled

I have modeled the "differences”, that are to be transmitted, internally as a DOM (say diffDOM), and it is suitably updated by using a custom-data structure (a hash-table whose key is the xpath expression of the “node” that was affected by the application, captured using an event-listener as said above). The diffDOM is updated every-time a “node” is "affected" in the DOM that the application maintains internally, say appDOM. i.e.

1. Every-time a node is created, inserted/removed/replaced (OR) if its attributes change, an event is fired - that's captured by a listener.
2. A book-keeping of these affected-nodes and the actions that "affected" them is maintained using diffDOM and a custom data-structure.

3. An optimized diffDOM is generated, that excludes certain operations. i.e. if a sub-tree is created and later removed, two things are done:

a) Initially the nodes of this sub tree as they are inserted, due to the event-listener, are stored in diffDOM.

b) Later all the nodes in this sub-tree are deleted from diffDOM. This is because we don't need to send both the information like:

"1) create the sub-tree" and "2) delete sub-tree".

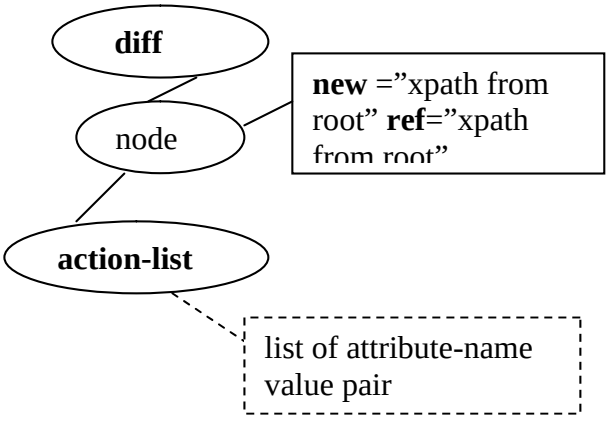
There are two possibilities how an application might use a DOM:

1. The application MIGHT use some standard XML file and manipulates the DOM representation of it, in a session, so relying only on the event-model it's not possible to trace the actions that "affect" the appDOM. i.e. if as sub-tree of such an appDOM is re-organized in a session, I won't already have a "reference" to these nodes, just as I would have one, had they been created dynamically?
2. Create a DOM afresh and manipulate it.

I have considered both these cases. As suggested all nodes are identified by an XPath-like path, as shown in the figures below. I observed the following advantages of modeling the "differences", as DOM:

1. It can be serialized easily, when its time to transmit/announce the same.
2. Easy to manipulate, this by itself is an advantage of DOM.
3. Easy to maintain - all operations can be performed using the standard tree operations provided by the DOM API.

7.1An example "diff" format

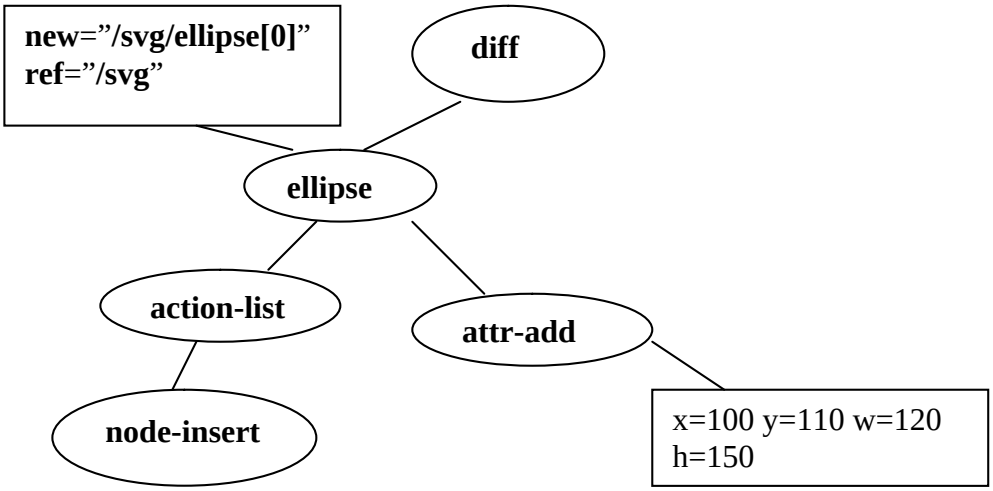


1. action-list
   Contains the smallest number of actions that have been currently performed on the node; i.e.-e:- insert/remove/replace/insert-before (till now I see that at most only action will be recorded even though several actions are performed as suggested earlier.)
2. A list of < name, value > pairs (if available) from the appDOM (an attribute insertion/deletion/modification is not recorded as an action; ONLY the latest set of attributes of that are "imported" directly from the corresponding appDOM, into this node.)
3. Apart from that an attribute-list that specifies the XPath-like location of this node are required: a) "new" - has the XPath-like location of this node that's recently been created by the application. b) "ref" - has the XPath-like location of a reference node. This is required to reconstruct at the other end - it tells us where this nodes is inserted with reference to the "parent-node" OR with reference to a "sibling-node".

The following elements appear within the “action-list”

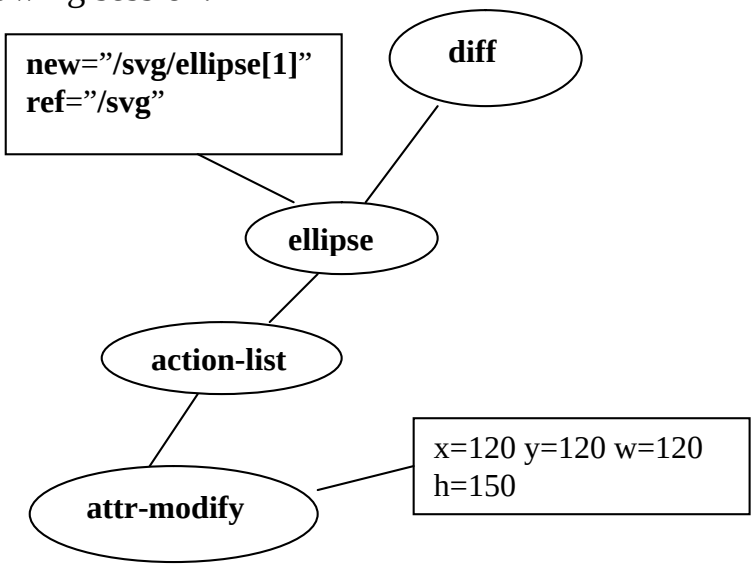
1. node-insert –
   When a new node is inserted into the DOM
2. node-remove –
   When an existing node is deleted from DOM.
3. node-insertBefore –
   When an existing node is inserted before another node.
4. node-replace –
   When an existing node replaces another node.
5. attr-add, attr-modify, attr-delete
   When an attribute is added, modified or deleted from a node.

7.2 An example of how the “diff” is generated initially.



The corresponding XML message:
<diff><ellipse new="/svg/ellipse[0]" ref="/svg"><action-list><node-insert/><attr-add x="100" y="110" w="120" h="150"/></action-list></ellipse></diff>

7.3 An example of “diff”, when further changes occur during the course of a drawing session.

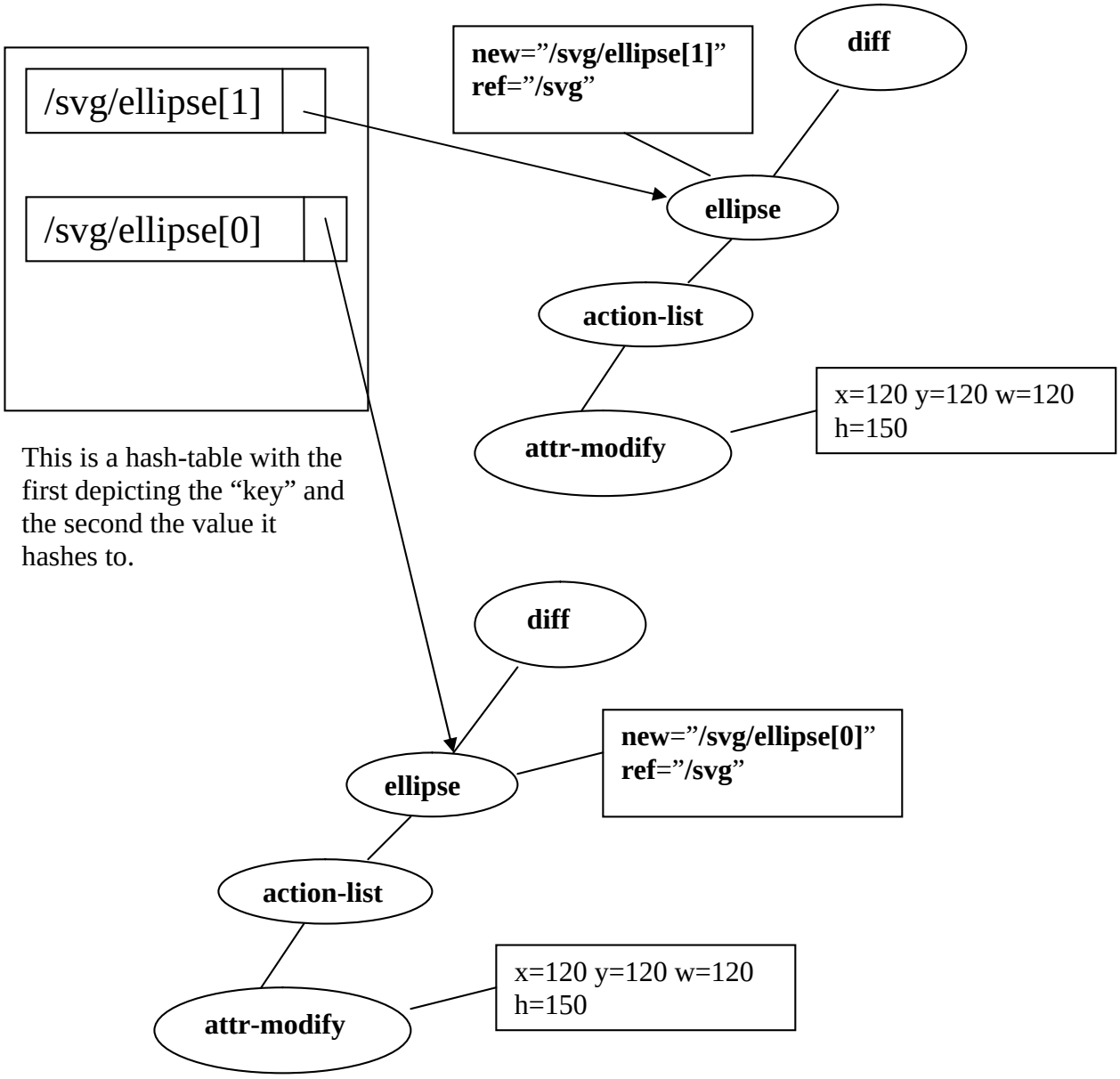


The corresponding XML message:

<diff><ellipse new="/svg/ellipse[1]" ref="/svg"><action-list><attr-modify x="120" y="120" w="120" h="150"/></action-list></ellipse></diff>

8. Notes on custom-data structure

The data-structure used in the XMLDiff, DiffListener, DiffDomNode classes deserves emphasis. Its basically a hash-table that can be potrayed as follows:



The first entry is the key of the hash-table - the XPath expression of the location of the target-node and the reference node. The value that will be looked up is the reference to the Diff-DOM node as shown.

9. Typical flow of control in the demonstrative application:

a) The application hands over a Document object it uses to construct the DOM upon instantiating XMLDiff.
b) A server runs in a separate thread, and a client runs in the main thread, as a part of this application, instantiates XMLPatch.
c) Whatever drawing operations are performed on-screen are captured by the DiffListener object instantiated in (a) and appropriate XML-based message is constructed.
d) The XML-message in (c) is dispatched to another peer in a different machine thru' the client.
e) The server gets the XML-message – the end of which is signified by closing the socket at the sending end – and hands it over to XMLPatch. Within it the XML-message is parsed using SAX - due to faster response less memory-intensive operations - and the DOM is updated corresponding to the actions specified in it.

10. Design Decision

a) A wrapper function for replaceChild(), insertBefore() is included which must be invoked if the application were to use these. It's more a design decision than a limitation; the reason is that I found it tough to decide an action based solely on the Event-model. For example replaceChild() removes the existing node and inserts a node designated, so it's hard to determine if the application genuinely deleted a node and inserted some other node – though a clever mechanism could be used to infer. But the Wrapper performs exactly the same operation as the original, and it works fine.

11. Useful Enhancements

a) I started to design as a generic solution, but I need to have a broad understanding of several applications and commonly occurring scenarios. So I reckon that a good API would be a better solution.
b) Design decision (b) could be re-tooled, perhaps in a future release.

12. Acknowledgements

I have referred the following sites:

1. www.w3c.org – DOM specifications
2. www.xml.apache.org – Apache's Xerces2 APIs
3. http://www.javaworld.com/javaworld/jw-12-1996/jw-12-howto.html
   - The HotSpot drawing program, used for the demo purposes.
4. http://java.sun.com/webservices/docs/1.1/tutorial/doc/index.html
   - Sun's Java-XML tutorial.
5. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnxmlnet/html/xmldiff.asp
   - Microsoft XML Diff and Patch tool.
6. http://www.deltaxml.com/
   - The Delta XML compare, merger and synchronize tool.