

Reverse Architecting the Mozilla 1.0 Editor using the Focus Approach

Hatim Shafique (shafique@usc.edu), Sweta Gandhi (swetaumg@usc.edu) Sathoon Cheawpanitch(cheawpan@usc.edu) , Rahul Shrivastava (rshrivas@usc.edu)

Abstract

With the rapid development and frequent evolution of object oriented applications, many applications have lost focus of their actual goals due to architectural erosion and problems about further maintenance and evolution have arisen. The solution to these problems is proposed by the FOCUS approach [1]. In this paper we have applied the steps of the FOCUS approach on a subset of such an object oriented application - Mozilla 1.0 (the editor sub module), recovered its existing architecture and thereby used it as the basis of further evolution.

1) Introduction

Here we apply the various steps involved in the Focus approach to an open source application namely, the Mozilla 1.0. Due to the enormous size of the application we applied the Focus approach to the editor sub-module of Mozilla 1.0.

The current Mozilla 1.0 editor (referred to the 'editor' hereon) was designed to be a WYSIWYG compliant application for on-the-fly web page design [2]. It does not specify or follow any particular architecture. The architecture (if any) that had been proposed before building the application has either been eroded over time by the random integration of components while providing additional features to the module or the initial architecture itself was built by integrating the components as needed, without following a defined architectural pattern. Various other problems related to the current architecture like

- Complexity – With almost 150 classes, the editor is highly complex. To add to the complexity the initial architecture has just a bunch of classes interacting with each other to produce the desired result.
- Referencing – There is no specific style of referencing the function calls.
- Flexibility – Since the classes have a lot of interaction with each other and the referencing is done inside the classes itself, maintenance or further addition of functionality could be a laborious task. As before any component is removed the references done inside the classes of that component have to be handled.
- All these problems pile up to hinder the maintenance and further evolution processes.

Our focus here is mainly on the application of the various steps of the focus approach to the editor module. The paper is organized as follows section 2 covers the architecture recovery step, section 3 covers the system evolution and section 4 gives the conclusions, this is followed by the appendix and list of references.

2) Architecture Recovery

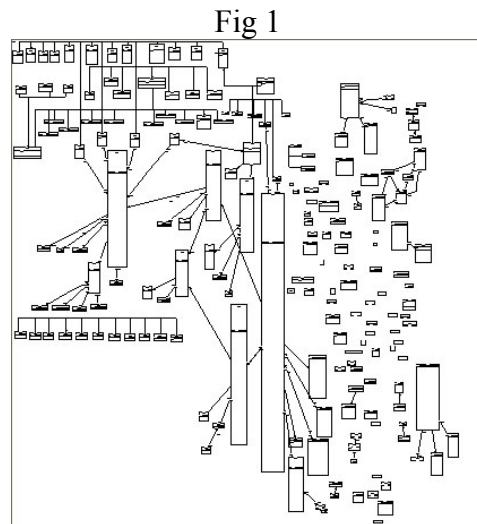
The architecture recovery process consists of the following 6 steps which at the end gives us a proposed architecture of the current editor.

a) ***Identifying Components***: The first step towards recovering the existing architecture is identifying the various components.

i) Generate Class Diagrams - To this end, we used Rational Rose to get the class diagrams of the Editor. Besides the editor module we reverse engineered many other modules related to the editor module, to determine if the changes made to the current editor would in any way effect the current working of the other modules. The results of our reverse engineering process showed that the editor module worked separately without any relevant interaction with the other modules and therefore we could apply the Focus approach to only the editor module.

Also using Rational Rose proved to be the bottleneck. The process of reverse engineering the whole of Mozilla1.0, and then extracting the editor module with its dependencies on other modules had to be changed, to reverse engineer the various modules separately and then see if they interacted with the editor module. Due to the available resources this became a very time consuming task.

Fig1 shows the class diagram extracted after reverse engineering the editor using Rational Rose.



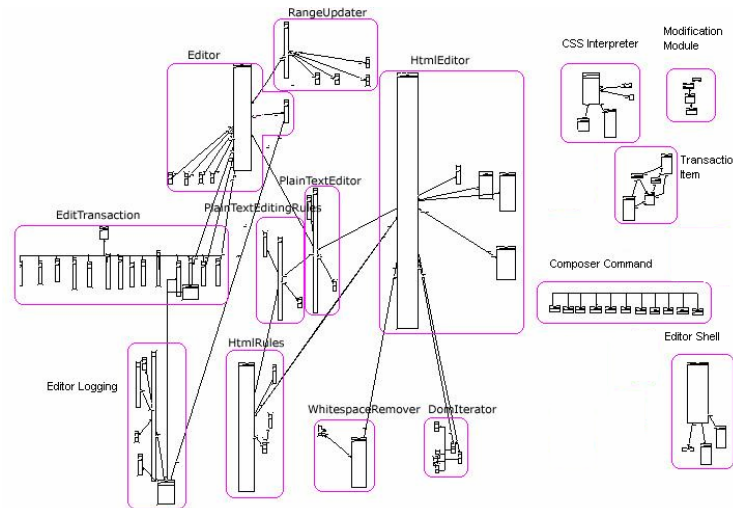
The class diagram recovered consisted of about 150 odd classes both connected and unconnected.

ii) Group related Classes - The above class diagrams were grouped into related classes using the following criteria. [1, 3]

- Classes isolated from the rest of the diagram were grouped into one grouping
- Classes related by generalization were grouped into 1 group as were classes related by aggregation and composition.

iii) Package classes into components – The classes identified in the previous step were packaged into abstract components, which were further grouped together into a higher level of components. Fig 2 shows the packaged components.

Fig 2



b) ***Propose Idealized Architectural Model:*** The initial idealized architecture for the Editor was proposed in this step based on the component diagram (Fig 3). For this we analyzed various classes of the editor i.e. the various properties they had, the function calls they made, and also made ourselves familiar with the various views of the editor namely HTML view, the normal view, the tag view and the preview view.

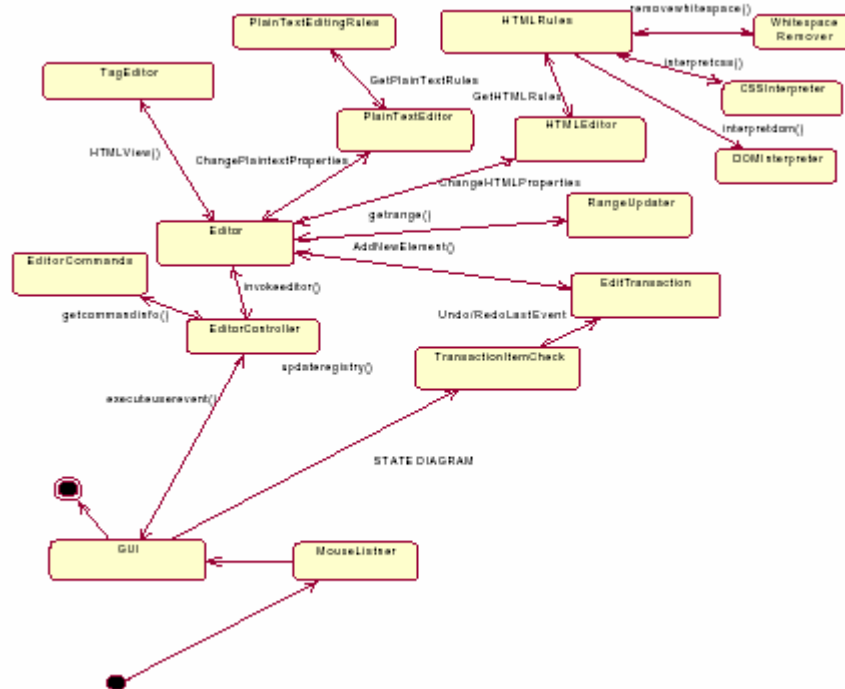


Fig 3

The proposed architecture is in the C2 style [4] due to the various advantages offered by this type of style for a GUI application. These advantages have been discussed later in section 2 (e).

Many parts of the architecture do not completely encapsulate all the classes recovered from the reverse engineering step but it does capture the main logic and user level properties of the editor.

- c) **Map Identified Components into Idealized Architecture:** This step involved mapping the above component diagram into the C2 architecture. Fig 4 shows the architecture in the C2 style after the mapping step.

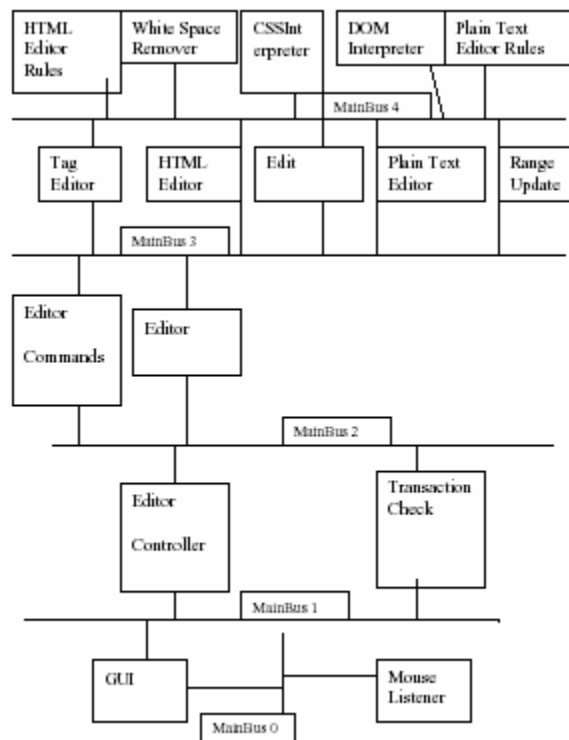


Fig 4

- d) **Identify Key Use Cases:**

Next, to make be sure that the architecture we proposed in the step above satisfied all the key cases; we identified some key cases that would be:

- i) Unaffected by our new architecture: HTML parsing, Copy, Paste, Undo, Redo and other normal text editing functionalities were all tested on the proposed architecture.
- ii) Corresponding to new requirements: Our primary aim was to extend the Editor to facilitate Concurrent Editing. For this we tested the proposed architecture for concurrent

editing, to understand whether we could support the same with added the additional components, while maintaining the architectural integrity.

e) **Analyze Component Interactions:** This step involved the generation of sequence diagrams for each of the cases identified in the previous step. Fig 5 shows one of the use cases namely Change Properties.

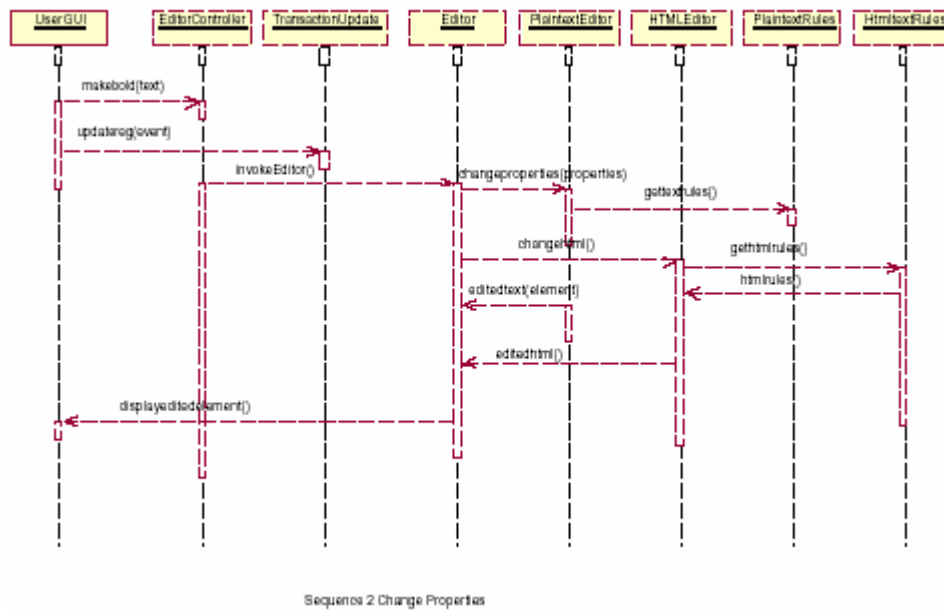


Fig 5

e) **Generate Refined Architecture:** While going through the key cases in the above step, we identified the key components in the architecture, we realized some of the components, could be removed as on referring the Mozilla1.0 site, we saw that those components were no more in use, and so were made obsolete by later additions and hence could propose our intermediate architecture shown in Fig. 6.

In this step many stand-alone components were grouped together as we realized that they were merely contributing to the complexity of the architecture and should have been encapsulated within one of the components of the editor architecture.

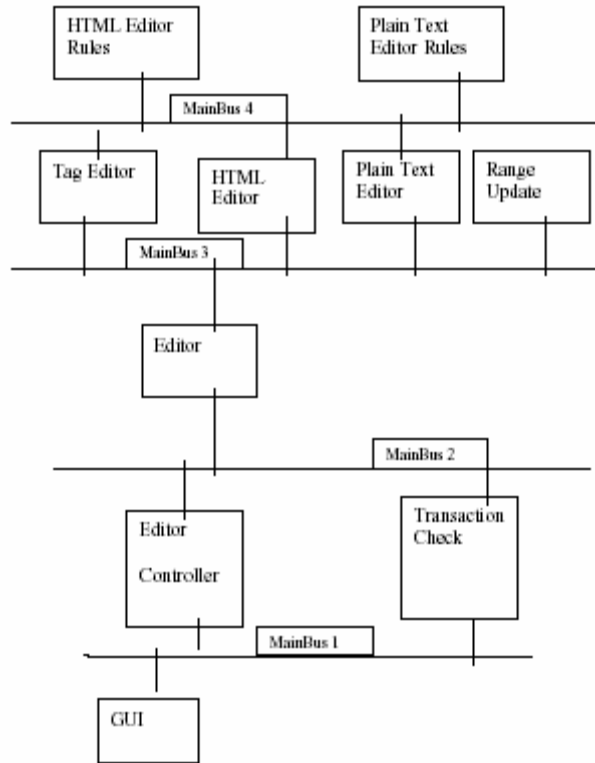


Fig 6

3) System Evolution

In the system evolution phase we evolve the system recovered in the architecture recovery step, refine the architecture and add the additional functionality of concurrent editing.

a) **Proposed Idealized Architecture Evolution**: At this point we considered alternative architectures that could be proposed for the new Editor.

i) Pipe and Filter Architecture:

Reason for consideration: the implementation of the classes could be mapped to a pipe and filter architecture as the data from one component is used as an input into another component (unidirectional data flow) and as mentioned in one of the problems the components do not interact with each other directly, only specific components interact with the other components.

Problems: Since the new component, which we propose to add, involves concurrency this architecture is not well suited as bidirectional flow of data is required. Finally, pipe and filter architecture requires that the processing of commands should be conducted in batches which means one of the components would keep waiting for the input if the component interacting with it is busy doing some other tasks.

ii) C2 Architecture:

Reasons for consideration: The main reasons for choosing the C2 Architecture are

- The new functionality we propose to add namely Concurrent Editing can be easily achieved by the C2 style while it cannot be implemented with the pipe and filter architecture or the existing architecture.
- Also the current Editor consists of individual components and the interaction between the components is of type request/reply, one of advantages of the c2 architecture is that it is best suited for this kind of interaction
- Furthermore since all the components do not have any referencing within the components itself and they interact with each other using the connectors between them, each component is a standalone, components can be added and removed at runtime, this adds to the flexibility of the architecture.
- The C2 architecture is also easier to integrate and supports such GUI based applications and is thus best suited for big application where different people implement individual components.

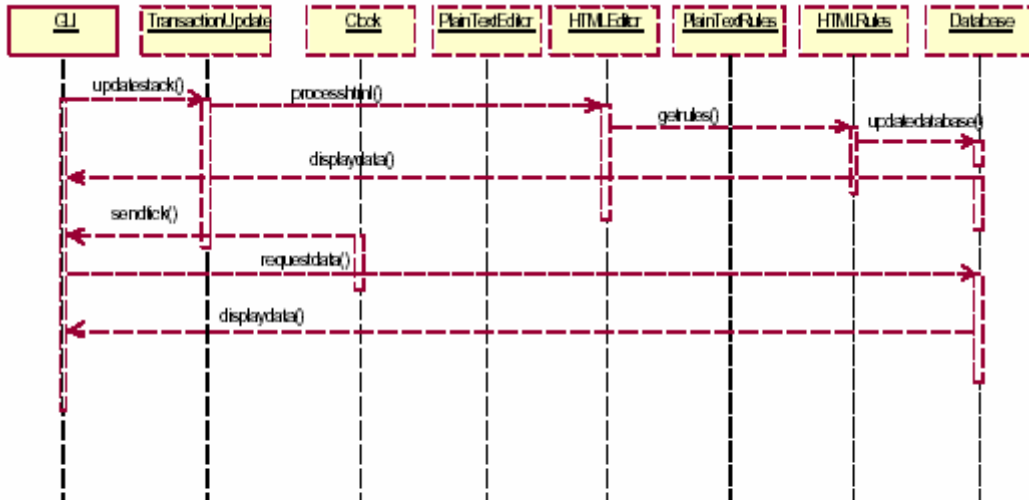
b) **Add/Modify Components:**

We propose to add a new functionality into the existing Mozilla Editor namely Concurrent Editing which enables two users sitting on different machines using the editor to edit the same file at the same time. The changes made by any one of them will be reflected in the others' window.

One of the possible advantages of this is to ease the process of building a web page with users sitting on different machines without having to continuously send and receive the updated documents to each other.

For this purpose we added a new component a database/server to keep a buffer of the most recent file open in the editor by any of its clients.

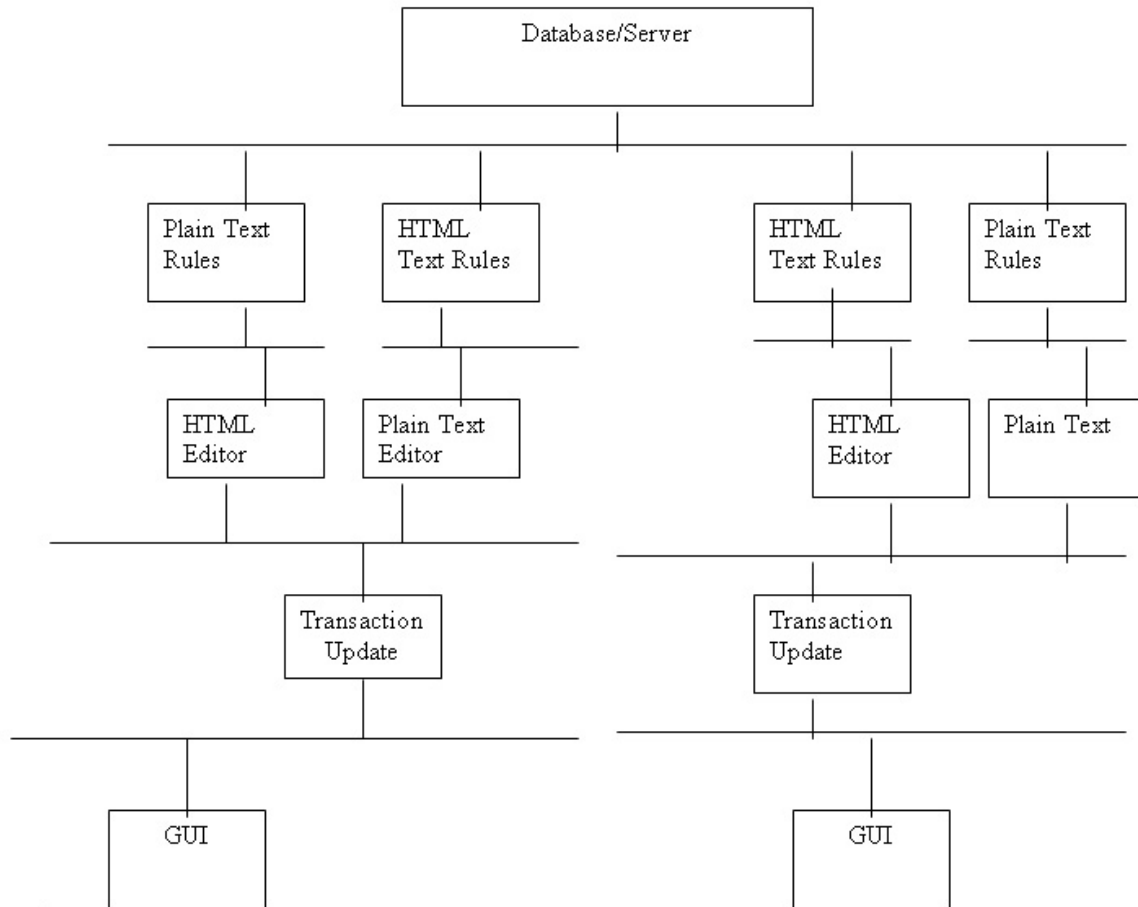
c) **Update Component Interactions:** We updated the proposed C2 to include the database/server needed for concurrent editing; also we updated the C2 to support 2 users (which could be extended to support more users). We tested the interactions by implementing the key use cases and generating sequence diagram for the same. Fig.7 shows these sequence diagram generated for one of the key cases.



SCENARIO VIEW PLAN

Fig 7

- d) ***Generate Evolved Architecture:*** All the above changes to the original architecture were integrated into the new C2 architecture for the editor. Fig 8 shows the new C2 architecture.



New Architecture

Fig 8

Disadvantages and Future Research:

1. Domain Translation is interpreting the requests and message in each component to execute the desired function. This unavoidably adds overhead to the message passing process but it a tradeoff between Component reuse and overhead of message passing.
2. There has to message replication at each intermediate component, which increases the complexity of interaction between components at different levels.

Future Research:

1. Recover all the other components of the Mozilla 1.0 and implement it using the C2 architecture
2. Provide additional functionality like a chat client to the existing Concurrent Editor.

References:

1. Focus: A Light-Weight, Incremental Approach to Software Architecture Recovery and Evolution by Lei Ding and Nenad Medvidovic
<http://sunset.usc.edu/publications/TECHRPTS/2000/usccse2000-522/usccse2000-522.pdf>
2. Extending the Mozilla Editor by Brian King
http://www.oreillynet.com/pub/a/network/2000/06/30/magazine/mozilla_editor.html
3. Introduction to Unified Modeling Language (UML)
http://sunset.usc.edu/classes/cs577a_2001/papers/Introduction_to_UML.pdf
4. Call Center Customer Care System
<http://sunset.usc.edu/~nenod/teaching/s99/casestudy.pdf>
5. Secrets from the Monster Extracting Mozilla's Software Architecture
<http://plg.uwaterloo.ca/~migod/papers/coset00.pdf>

Appendix

List of files submitted.

1. Java source files (located in directory "proj_test\proj_test\proj1\c2\Mozilla")
 - TransactionRegistry.java
 - Server_DB.java
 - PlainTextEditorRules.java
 - PlainTextEditor.java
 - MozillaEditor.java
 - HTMLERules.java
 - HTMLERules.java
 - ClockComponent1.java
2. Rational rose .mdl file
Contains class diagrams recovered using Rational Rose.
 - editor src1.mdl
 - editor src_back1.mdl
 - editor src_back12.mdl