coDoc: A Tool for Managing Relationships between Codes and Documents

Dejun Qian

electronseu@gmail.com

Abstract

When writing and testing software, we need the documents in hand. This is extremely true for software that is document intensive, such as hardware driver and virtual device modules. This paper presents coDoc, an integrated development environment for managing relationships between codes and documents, as well as developing the codes. We first describe the architecture and the features of coDoc, We then discuss several design decisions that we faced in developing coDoc and the trade-offs in making these decisions. To provide perspective, we report our experiences in mapping two virtual network interface card (NIC) modules to the real NIC documents.

1 Introduction

Documents play an important role in software engineering process, from requirement analysis to architecture design, from code writting to software testing, from software maintainence to software upgrading. Software that relates closely to hardware devices is extremely document intensive. Examples of these software could be bootloader, Operating System (OS) Hardware Abstraction Layer (HAL), virtual devices of virtual machines. These software should conform to hardware documents to operate correctly. A typical embedded CPU usually have documents with thousands of pages. A computer system usually consistes of over ten hardware modules. To effectively maintain the relationships between the codes and the documents is really important for developing and verification of these software.

Much work has been done to deal with the relationships between Application Programming Interface (API) documents and their client codes [1]. API documents are usually well indexed by the functions or the classes. When we are interested in a piece of code in the client software, we can easily find the related API documents by the name of the functions that the piece of code calls. The relationships between the client codes and the API documents can be easily constructed by using the function names as the keywords. However, the codes related to hardware and their documents are usually related in different ways. The hardware documents usually talk about how to use the hardware, more specifically how to program the registers inside the hardware modules. The hardware documents differ from the API documents in the following ways.

- the hardware documents are usually published as pdf files, while the API documents are usually html files.
- the hardware documents are composed of texts, tables and diagrams, while the API documents are usually texts only.
- the hardware docuemnts are not orgnized by functions or classes, which are basic elements of codes.
- hard for the computer to relate the hardware document pieces to the code pieces automatically.

These differences make the constructing of the relationships between the hardware related codes and their documents really challenging. To our knowlege, there is no tools exist to address this problem. A simple workaround used is to include the positions of the related document pieces in the documents as comments the codes. However, this method can only handle simple situations. If there are thousands of these comments cluttered in the codes, it will be very hard to manage them. This paper presents coDoc, a tool to easily create, maintain, and display the relationships. The key features of coDoc are as follows:

- provide the ability to display codes and documents;

write and edit

- provide the ability to develop codes.
- select code pieces based on syntax tree of the code.
- code selections are stable to the changes of unrelated codes.
- create relationships between codes and documents.
- manage and display the relationships.

must be list of features (noun phrases) not actions (verbs phrases)

eh?

between

Using coDoc, we have created relationships for two virtual NIC devices in QEMU. These two devices are DIO and E100.

and ___?

The main contributions of this paper are listed as follows.

← end sentence here

- design a framework to display and manage codes, documents and relationships.
- remain integrity of the relationships when the codes evolve.
- present a standard format for the code piece identifier.
- a standard format for the document piece identifier.
- a standard format for the relationship between the code and document.
- implement coDoc which adopts these standard formats.

← Each contrib is then a separate sentence

are these contributions?

The rest of the paper is orgnized as follows. Section 2 introduces the background of poppler and CDT. Section 3 provides the overall architecture and key features of coDoc. Section 4 gives the design decisions of coDoc and the trade-offs of these decisions. Section 5 summarizes our experiences of creating and maitaining the relationships for two virtual devices. Section 6 presents the related work. Section 7 concludes our work and gives some thinking towards the future.

combine with your contributions

2 Background

← This needs to include other tools that make links between different media. Prof Delcambre worked on something like this for tracing requirements IIRC.

2.1 poppler

We adopt poppler, a open-source PDF render, to render PDF content in coDoc. This PDF render is written in C++, and is widely used in Linux

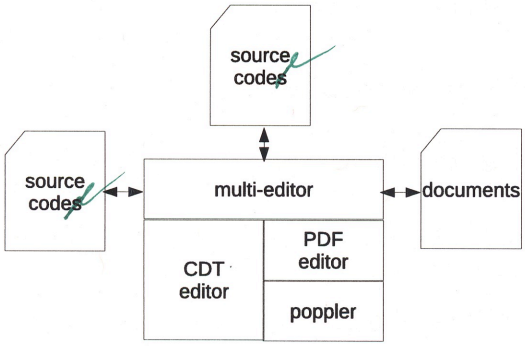

Figure 1: coDoc Architecture and Features

systems. It uses Cairo and GTK to draw the rendered content.

2.2 C/C++ Development Tool

C/C++ Development Tool (CDT) is an Eclipse plugin which supports C/C++ syntax parsing. It is written in Java and provides Eclipse with the ability of syntax highlighting and code completing when developing C/C++ codes.

3 Architecture and Features of coDoc

The architecture of coDoc is shown in Figure 1. It contains CDT editor, poppler, PDF editor and multi-editor. There are three kinds of data here, 1) the codes, 2) the documents, 3) the relationships. We only handle C/C++ code at this time, as most system code are written in this language. CDT editor reads and parses the codes, highlights syntax elements. We use poppler to read the PDF file and render the PDF pages to images. As poppler is developed in C++, and Eclipse RCP is in Java language, we need an adaptor to bridge the C++ code and the Java code, We use JNI to implement this adaptor. PDF editor implements an adaptor, and uses it to get the PDF page from poppler and displays the page. The multi-editor is responsible to read the relationship data, and highlight certain code piece and document piece respectively. To represent a relationship, we first need to define the strategy to identify the code piece and document piece.

3.1 Identifying the Code Piece

The codes are always evolving during its lifetime. We don't want our relationships created for one version stop working for other versions. This goal calls for a robust identifying strategy for the code piece. Our first method identifies the code piece as the offset of the selected code from the beginning of the code file and the selected length. This method is straight forward and is easy to understand and implement. However, this method definitely will violate the robust requirement. To achieve the goal, we need to find some properties of the code that are relatively stable when code evolves. These properties should also can serve as a key to uniquely identify the code piece. The intuitive thought is to use the semantic information, as it describes the intrinsical meaning of the code. However, there is no formal method to describe this information. We use syntax information to form the main element of our identifier. The code can be parsed and represented by a syntax tree. The change of the code in one branch will usually not affect the code in the other branches. If we can identify the code piece based on the position in the syntax tree, we can design a robust method to identify the code piece. To make the method more robust, we can introduce the information in the first method as the secondary key. When the syntax identifier fails to locate the code piece, we can use the offset information to assist the decision. We also include in the identifier the content of the code piece for the integrity check. The structure used to achieve this goal is shown bellow,

```
class CodeSelection {
  private String syntaxcodepath;
  private String selcodepath;
  private String syntaxcodetext;
  private String selcodetext;
}
```

Identifying the Documentation Segment?

4 Methodology and Design Decisions

4.1 Embedded vs Seperate for PDF Reader

The tool should display both codes and documents at the same time. We can implement the code viewer and the PDF viewer in seperate process, and use inner process communication (IPC) techniques to synchronize. The advantage of this method is that each process has its own window, can ocuppy the whole screen, and display more content which reduces the need to scroll the window horizontally. When the user uses a computer with two monitors, the tool implementing this method is extremely a good choice, as the user can choose to display the code and the document in seperate moniter. However, this implementation is heavy, and get performance issues. When using the tool, the user may need to launch the PDF reader, which is slow. Our earlier implementation use this method, and encountered a lot of synchronization problems oriatated from the IPC communication. In coDoc, we choose to implement the code viewer and the code viewer in the same process. This way, all the commucations are done in the same process, No IPC communication is needed, which is more efficient and makes the tool more user-friendly. With the new method, we can smoothly operate the tool without any uncomfortable feeling. The advantage of the first method become the disadvantage of the new method, as the code viwer and the document viwer should share the whole screen now. However, with a efficient and user-friendly screen split strategy and screen area assignment method, we can make the user feel less uncomfortable when viewing the content. Also, most programmer now not only are using two moniters, but also the moniters they are using are larger and larger. Using a larger moniter with high definition will make this disadvantage nothing. And even better, in this situation, this method will overcome the first method, as moving the eyes between two big moniter is energy consuming.

4.2 Relationship Store Model

In coDoc, we store the relationship data in XML file, instead of the SQL database. The reason that we didn't use an SQL database is that the relationships is semi-structured, as we decide to store the content along with the identifier. The content is used to further check the integrity. XML database is very suitable for our semi-structured data here.

5 Evaluation

Effectiveness. We have evaluated coDoc by mapping two virtual devices and their related documents. The number of relationships created and the time used to created these relationships are listed in Table 1.

Table 1: Evaluate Result

Code	# of Relationships	time(hr)
dio	108	2
e100	213	4

The fine granularity of the identifying method makes the relationship results very accurate, 80% of the code pieces are marked inside the statement. This result is hard for comment-based method to achieve. The same result happens for the document. In the whole relationships, 90% of the document pieces are marked on cells in the tables. The comments in the PDF file will be too crowd to be useful, as it will affect the normal reading of the document. Our approach can also make the development efficient, as we don't need the developer to change either the code or the document while only relationship is created. This makes the code and document neat and easily to read and understand.

Usability. We implement coDoc as an Elipse[1] Rich Client Program (RCP), which can make our tool share the use style with Elipse that is acustomed by many programmer. Figure 2 is the screenshot of the tool. Three persons with no exprience with coDoc before have successfully use this tool to create the relationships for their codes and documents. They didn't find any difficulties when using this tool. All the feedbacks are positive regarding to

[1] http://www.eclipse.org

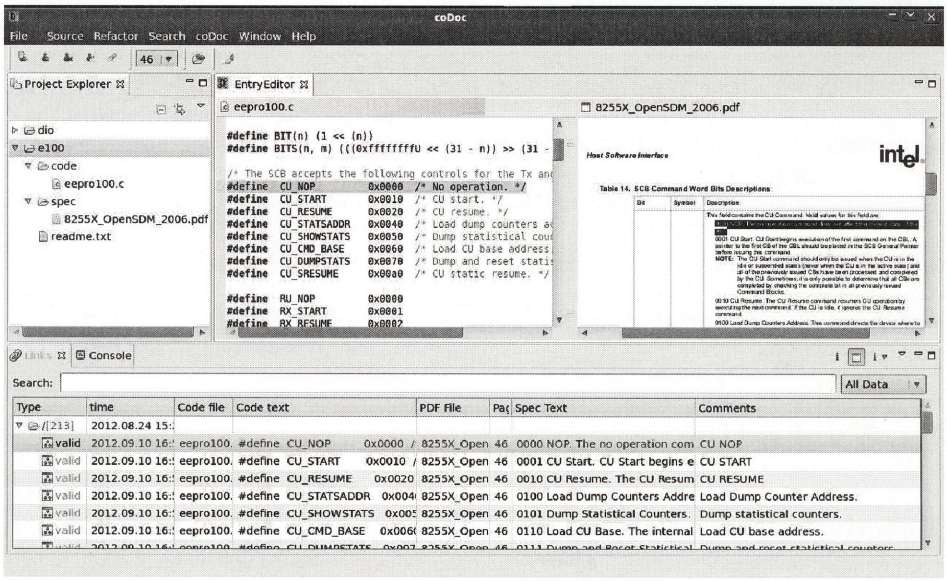

Figure 2: coDoc Platform View

the use of the tool. Now they can easily create their relationships by selecting the code and document piece using mouse, input the comment using keyboard. They can also manage the relationships by sorting by attributes, or by search relationships by keywords. They also find the feature that allow them to categorize the relationships helpful. This feature makes the relationships more when the number of relationships is large.

In our own experience, we found that the ability to see the codes, the documents and the relationships on the same screen saves a lot of time, as we don't need to switch between different tools. The visualized interface also saved us time to learn the detailed syntax about the identifier formats, and to write the identifier manually. We believe this tool will lead to increased productivity.

6 Related Work

Current tools can manage codes. These kinds of tools include revision control software like git, svn. Revision control tools can give the evolving history of the software. These tools usually adopt text comparison tools to figure out the differences between different versions of the software. The comparation tools can easily give the result about the matching points

and mismatching points in the codes, which means they can manage the relationships between different versions of codes. However, these tools are not designed to manage documents. Though some people also use them to manage documents, they only use it as a persistent data warehouse. They can not take the advantages of the comparison tools, as these tool usually can only deal with text file. People are using code comments to record the related document piece. However, no consistent format exists, which makes these information not managable, especially when these comments are made by many different developers.

Full text indexing system is developed to processing documents. However, they can only relate keywords to certain documents. They do not deal with both codes and documents. Like full text indexing system, search engines also work with documents in similar way. Most of the documents that are handled by search engines are web pages. Many PDF readers can support highlighting and commenting. With these features, we can relate a document piece to a code piece. However, these links created are not stable, and PDF readers can not check the integrity of these information.

Generally speaking, these are tools can either do code management or do document management. However, they are not designed to do both, and managing the relationships between codes and documents is definitely a mission impossible task for them.

7 Conclusions and Future Work

We implement coDoc, a tool to managing the relationships between the hardware related codes and their documents. The main contributions in this paper are the method to identify the code piece. Our experiences of using coDoc to manage two virtual device codes and their documents show the tool is robust and efficent, and is helpful when developing and maitaining the virtual devices. The relationships remain stable respect to the evolution of the codes.

There are still opportunities to improve coDoc,

1. combine it with revision control tools to provide a more user-friendly tool suit for developing and maitaining the codes while managing the documents at the same time.

than what?

2. more robust method to identify the document pieces in the PDF documents to make the tool invariant to the change of the documents.

3. use machine learning method (use the data created as seed to train the model) to construct and check the relationships between hardware related codes and their documents.

References

[1] Pandita, R., Xiao, X., Zhong, H., Xie, T., Oney, S., and Paradkar, A. Inferring method specifications from natural language API descriptions. In *Proceedings of the 2012 International Conference on Software Engineering* (Piscataway, NJ, USA, 2012), ICSE 2012, IEEE Press, p. 815825.

You need to add citations throughout. Describe how related work is related, and how it differs from yours