

CHIME: Customizable Hyperlink Insertion and Maintenance Engine for Software Engineering Environments

P. Devanbu
Dept. of Computer Science
University of California
Davis, CA, USA
+1 530 752 7324
devanbu@cs.ucdavis.edu

Y-F. Chen, E. Gansner
AT&T Labs–Research
190, Park Drive
Florham Park, NJ 95030
+1 973 360 8653/8646
chen,erg@research.att.com

H. Müller, J. Martin
Dept. Of Computer Science
University of Victoria
Victoria, BC V8W 3P6
+1 250 721 7630
hausi,jmartin@csc.uvic.ca

Abstract

Source code browsing is an important part of program comprehension. Browsers expose semantic and syntactic relationships (such as between object references and definitions) in GUI-accessible forms. These relationships are derived using tools which perform static analysis on the original software documents. Implementing such browsers is tricky. Program comprehension strategies vary, and it is necessary to provide the right browsing support. Analysis tools to derive the relevant cross-reference relationships are often difficult to build. Tools to browse distributed documents require extensive coding for the GUI, as well as for data communications. Therefore, there are powerful motivations for using *existing static analysis tools* in conjunction with *WWW technology* to implement browsers for distributed software projects. The CHIME framework provides a flexible, customizable platform for inserting HTML links into software documents using information generated by existing software analysis tools. Using the CHIME specification language, and a simple, retargetable database interface, it is possible to quickly incorporate a range of different link insertion tools for software documents, into an existing, legacy software development environment. This enables tool builders to offer customized browsing support with a well-known GUI. This paper describes the CHIME architecture, and describes our experience with several re-targeting efforts of this system.

1 Introduction

The World Wide Web (WWW) is an accessible, powerful, and ubiquitous medium for the delivery and access of widely distributed documents. The low cost, flexibility, and ease of access has led to rapid advance of browser technology. HTTP clients such as web browsers support sophisticated interactive capabilities: hot lists, browsing history, drag and drop, selection forms, etc.

In addition, they have the ability to handle different types of networking protocols. These clients are ubiquitous, and are constantly being enhanced. There is thus a strong incentive to leverage this increasingly powerful browsing technology in other contexts; in particular, CHIME is aimed at exploiting web-based browsing in software engineering environments. Software development projects are becoming increasingly more distributed, in response to personnel costs and market segmentation. Such projects will involve distributed software documents that have complex inter-relationships. Browsers that can handle such distributed documents, and expose their inter-relationships as traversable hyperlinks, will be helpful to developers in these projects.

Web-based source code browsing has been discussed by other authors [15, 14, 23]; our specific goal is to facilitate the addition of web-based browsing into existing software development environments (SDEs). The central focus of CHIME is the task of inserting HTML links into source code, using information stored in repositories associated with existing SDEs. Specifically, CHIME assumes that the online documents are stored in some sort of (possibly distributed) repository, and that syntactic and semantic inter-relationships between documents are derived by some analysis tools and stored in a repository in some format (not pre-determined). A user of CHIME can then specify a set of links, the relationship of the links to the contents of this database, as well as the action to be taken when the links are activated. From this specification, CHIME generates a link insertion engine that reads documents, queries the database and interprets the results, inserts the appropriate HTML links, and outputs the resulting text. Another key goal in the CHIME project is to provide flexibility in link insertion. The variability that CHIME tries to accommodate includes database implementation, interpretation of database contents, number of links, and the meaning of links.

The rest of this paper is organized as follows. In section 2, we discuss the importance of browsing functions in software development environments, and the motivations of CHIME. Then, in Section 3, we discuss the

details of how CHIME tries to achieve the goals outlined above. In Section 4, we briefly describe some applications of CHIME. In section 5, we discuss relevant work on automatic link insertion, including related projects, and the relationship of that work to the goals of CHIME; this is followed by the conclusion.

2 HTML browsing for Software Documents

A key functionality in software development environments is browsing. It has been reported [7] that programmers can spend 50% or more of their time trying to understand the system, especially in large projects. Good browsing support can be helpful during this process. Students of program understanding have described two strategies used by programmers to comprehend programs: top-down and bottom-up. Empirical studies [19, 17] indicate that programmers use both styles of exploration and comprehension. Thus, it would be desirable for a browser to support both kinds of exploration. For example, a C++ programmer might start first examining the class structure of a system; she might then focus on a few specific methods and their calling structure; then she might scrutinize the precise use of class member data of a specific class within the member functions. This might again lead to examine the type definitions of some class data members, which leads to further exploration of the class structure, etc. Different users' styles and different comprehension tasks require different links. This has led us to emphasize flexible link configuration as a vital, key goal of the CHIME system. A secondary goal in the CHIME system is *retargetability*. Large legacy projects have a lot of inertia. Part of this is due to personnel and training: it can be time-consuming, expensive and difficult to introduce an entirely new development environment into an ongoing project just because part of the development effort has been delegated a geographically far removed site. So it would be desirable to introduce tools to support distributed development into the project in a minimally disruptive manner.

In addition, because of the complexity of the configuration and build procedures [31] in large projects, it can be very expensive to bring new tools and processes into the environment, specially if these tools need to be run over the entire source base. If at all possible, it is advantageous to introduce new capabilities into the development environment in a minimalist, incremental fashion that avoids global analysis with new tools. A goal of the CHIME system is to introduce web-based source code browsing into an existing development environment without forcing changes to the configuration and build procedures. This has two advantages. First existing processes are not disrupted. Second, the developers don't have to learn a new browsing tool: their current familiarity with the WWW browsing tools can

be leveraged.

A browsing tool [28, 4] includes a graphical interface (GUI), and it accesses the document repository and a cross-referencing relational database (XRDB). The relational database has the relations corresponding to browsing steps (e.g., from a reference to a function to its definition). The GUI includes document viewers and other devices such as buttons, menus and scroll bars for manipulating the viewers. For example the GNU emacs editor [28] can use the GNU tags database to support browsing of C code: a simple mouse action can move the view from a function reference to a function definition. C++ development environments such as those available from Symantec [25] also support various browsing actions. With distributed environments, browsers need to access remote documents transparently in response to browsing requests and queries. It is important to note here that the information in the relational database used by the browser is typically built by analyzing source documents. For example, the databases used by the Ciao [4] browser are built by running analysis tools over C, C++, or Java programs. These analysis tools are quite difficult to build, particularly for complex languages with sophisticated pre-processors. Tools such as CIA[3], CIA++ [13], and Acacia[5], which analyze C and C++, take many person-years to construct. To the extent possible, it would be highly desirable to use the databases built by existing analysis tools.

Some key issues to be addressed in browser implementation are GUI design, extraction of browsing relations from source code, and distribution. The goals of CHIME are to leverage existing tools and technologies in all of these areas. HTML, the source language of WWW, provides a powerful and simple GUI interface paradigm. Public domain (or low cost) browsers such as Netscape Communicator support HTML, and are well known to many users. These browsers come with many desirable features such as "click-to-browse", browsing trails, hot lists, etc. Distribution of documents over different machines that communicate using TCP/IP and associated protocols is handled in a transparent manner. Before HTML browsers can be used to browse source code, one must insert software browsing relations into source code as HTML links. We now describe how CHIME addresses this link insertion task.

3 The chime Architecture

CHIME is a meta tool; it generates tools that insert HTML links into raw text. The overall framework in which CHIME-generated tools function is shown in Figure 1.

Example and Background

An example fragment of C source code is shown in Figure 2 before and after link insertion. In this frag-

Input Source:

```
if (transmutable(a,b))
  j = interSolve(a, b);
else j = -1;
```

Output HTML:

```
if (<a href="http://mach/cgi-bin/find?name=transmutable&view=plaincodeview&type=fundlink">
  transmutable </a> (a,b))
  j = <a href="http://mach/cgi-bin/find?name=interSolve&view=plaincodeview&type=fundlink">
    interSolve </a> (a, b);
else j = -1;
```

Figure 2: Input Source and Output HTML

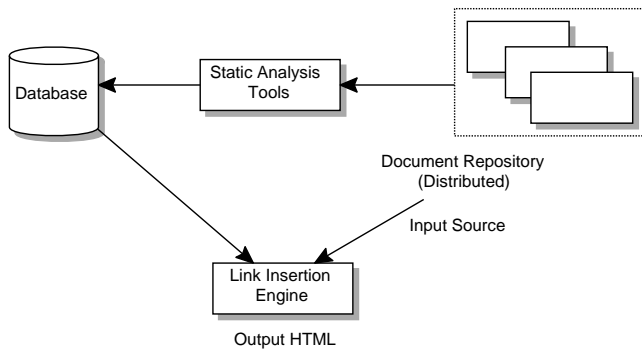


Figure 1: Architecture of a link insertion device

ment, links are inserted into the occurrences of function names; these would be highlighted in browsers. The developer can then click on the names, and the corresponding gateway tool `mach/cgi-bin/find` would be executed. The effect of this would be to retrieve the source text constituting the definition of the relevant function, into which HTML links would be inserted; the resulting HTML would be returned to the browser. This can then be a basis for further browsing. It is important to note that links are inserted *on demand*. Storing “with links” and “without links” versions of all source would be undesirable in many cases, given the volume of software documents, the frequency of change, and the importance of presenting accurate information to maintainers. The advantages of dynamic link insertion have been explored at length elsewhere [2, 11].

As discussed earlier, different sorts of links may be required for different types of program understanding and maintenance tasks. CHIME can be used to generate link insertion engines to insert different sorts of links. A single CHIME-based link insertion engine can expose different types of links depending on the current maintenance task. We now describe the elements of an HTML link

that are relevant to CHIME, and the range of variabilities in links that can be handled.

Links, and their insertion

There are two important parameters of an HTML link: the *position*: where the link gets inserted (this is often called the link *anchor*) and the *semantics*: what happens when the user of an HTTP client activates that link by clicking on it.

In most software browsers, the position of the link is determined by an XRDB, which is derived by mechanical static analysis of the source code. For example, the positions in the source code (*viz.*, line numbers, column numbers) of all function calls, variable references, etc., are stored in an XRDB. The link insertion engine uses this information to introduce links in the input software document. If the original XRDB can generate information on link positions in the order of their occurrence in the source code, the insertion can be performed very efficiently.

An HTML link can either be a simple URL that points directly to another HTML page, or an invocation of a CGI engine to generate HTML from a data source. Links can also contain additional information that is used by the CGI engine for its processing. For example in Figure 2, the first link, `http://mach/cgi-bin/find?name=transmutable&type=fundlink`, contains two additional parameters, `name` and `type`, along with values. We assume that HTML documents are created “on-the-fly” from source code; all CHIME links are of the CGI type. For example, if the first link shown in Figure 2 is clicked, the definition of the function `transmutable` will be transformed to HTML with embedded links and then displayed. This is done using the `find` (or other) CGI executable, in several steps.

Step 1 `find` will query the database to find the URL of the target document that actually contains the definition of the function `transmutable`. We refer

to this query as the *target query* below, since it finds the target document. Then, the document itself will be retrieved.

Step 2 After retrieving the document, the database will be queried again to locate cross-referencing information about where function calls occur within the body of the definition of `transmutable`. This query is called the *anchor query* below. Each tuple in the *anchor relation* resulting from this query gives a position or (*anchor*) for a link to be inserted.

Step 3 Next, the links will be inserted to invoke a specific CGI such as `find`; additional information in the form of parameters (such as `name` and `type` in Figure 2) that could be used by `find` could also be placed in the link. This information can be drawn from the corresponding tuple in the anchor relation.

CHIME allows each of the steps above to be customized, so that both the position and meaning of links can be tailored. It is also designed to be database-retargetable; the CHIME user can specify how to get the relevant information out of a range of different XRDBs. In this way, CHIME can accommodate the browsing needs for different types of tasks (via different links) and can provide a way to add WWW browsing into legacy projects (via XRDB-retargetability).

In the rest of this section, we discuss the different customizable aspects of CHIME.

4 Chime Language

The CHIME language is used to specify the particulars of the three steps described in the previous section. We first describe the main concepts of the language, and then illustrate the details of the specification language with an example.

CHIME language concepts

The language deals with several basic concepts of browsing. First, there are *documents*, which can be of various *document types*: source code documents, test documents, configuration management documents, formal specification documents, etc. In the following examples, we just deal with source code documents. Each *document type* is associated with one or more *views*, which are collections of *links*. Link specifications are central to CHIME. They identify: first, for Step 1, the target query that is used to locate the target document. Link specifications also determine the type of the target document (function definition, class definition, etc.). They also specify the anchor query (which is also specified separately) for Step 2, and the details on what parameters are to be inserted into the links (in Step 3). A view represents a specific HTML rendering of a document type, exposing a certain set of links relevant for a particular task, such as design, implementation, maintenance, etc.

There may be several views applicable to a given document type. For example, in a function declaration, one view may expose links from classes to their declarations, another from a class to a list of its members or ancestors, etc. Whenever a document is viewed with CHIME, alternate views (if available) will be presented as a series of links at the top of the page. A default view is used for each document type; selecting an alternative view makes that the default view for that document type.

A CHIME application is generated from a set of *specifications* (see examples in Figures 3 and 5) each describing different elements of the 3 steps of link insertion described above. Each specification names an element, specifies what type it is (view, link, target query, link query, etc.), and gives *values* for several *attributes* of the element (e.g., for a link, the specification would say what its target query is, what its anchor query is, etc.).

CHIME is a domain specific language [26]. As an aside, we note here that the CHIME language differs from languages for other domains like source code analysis [8] or web applications [16]. All CHIME-based applications have the same basic 3-step algorithm for link insertion. So unlike [8, 16], the CHIME language has no procedural constructs, but a series of table-like structures that describe the elements of each step of the (fixed) 3-step application algorithm. In this sense, CHIME resembles parser-generator or lexer-generator systems, which use a fixed table-driven algorithm; the languages for specifying lexers and parsers have no procedural elements either.

Implementation Details

Now we consider in detail how a browsing action (*i.e.*, a click on a link) is processed in CHIME, along with the applicable CHIME specifications and their interpretation.

Each URL in the HTML generated by a CHIME insertion tool names the link that it is associated with. For example in Figure 1, the name of the link is `fundlink`¹ in the first URL. This link is described in a CHIME specification, from which the invoked link insertion engine is generated. The engine recognizes `fundlink` as a link from a function to its definition; its specification is shown in Figure 3. The attribute `resulttype` gives the document type of the target document that will be displayed when this link is exercised: in this case, it has the value `functioncode`, which indicates that it is the body of a function definition. This document type is used to identify a default view that would be used to display the document.

The specification also names a target query to be eval-

¹As we shall see below, the names of links, views, etc. are encoded into HTML links. Thus, in practice, to minimize resource usage, it is important to keep these names much smaller than the ones we use in our illustrative examples.

uated against the XRDB. All queries are separately specified in the CHIME specification. The target query

```
fundlink = {LINK
  resulttype=function-code
  focus = function
  anchor-query=funrefs
  target-query=fdlq
  mapparms = {(name,name2)}
}
```

Figure 3: Example of a link specification

named in the specification of `fundlink` is `fdlq`; this query specification is shown in Figure 4. This specification includes a query string in which to substitute certain parameters specified in the link before executing the query. For example, the first link in Figure 2 includes a parameter `name` whose value names a function `transmutable`; this value would be substituted into the query string. The resulting query “`rigirep Function type.db | fgrep transmutable`” would then be evaluated. The precise mechanics of evaluating the query is determined by the database interface, which is described in the following section. The target query specification also describes how to find the document locator in the result. The specification of `fdlq` says that the file locator is given by the attribute `file1` of the resulting tuple, and the starting line number is given by `refline`. Let’s assume here that the document location is `file://mach2/src/transmutable.c`. Using the document location, the document is then retrieved. The precise mechanics can vary with the particular SDE; the retargeting mechanism is described in the following section. This concludes Step 1 of the link insertion process.

Associated with each document type, there is a default view. In this case (Figure 3) the resulting document type is `functioncode`. The default view associated with document type is `plaincodeview`,² shown in Figure 5. This default view specifies two links, `vardlink` and `fundlink`. We now turn back to `fundlink` (Figure 3), and examine how Step 2 of link insertion proceeds. The `fundlink` specification names an anchor query, `funrefs` which is specified separately (Figure 4).

The `funrefs` query specifies a query string into which the document locator is substituted. For example, the locator `file://mach2/src/transmutable.c` is substituted into the query string given for `funrefs`, resulting in the actual query “`rigirep "file://mach2/src/transmutable.c" call.db`” Evaluating this query results in a list of tuples, each of which corresponds to the location of a link (an anchor position) in the document.

²Details of associating document types with default views are fairly straightforward, and are omitted here.

```
fdlq = {TARGET_QUERY
  string="rigirep Function type.db \
    | fgrep \"\$name\$\"
  focus=function
  resultdocument={ATTRIBUTE
    start = "refline"
    locator = "file1"
  }
}
```

```
funrefs = {ANCHOR_QUERY
  string = ‘‘rigirep \"\$loc\$\" call.db’’
  position = {ATTRIBUTE
    linenummer = "refline"
    matchstring = "name2"
  }
}
```

Figure 4: Example specifications of a target query and an anchor query

```
plaincodeview = {VIEW
  fortypes = <function-code>
  links = <fundlink,vardlink>
}
```

Figure 5: Example of a view specification

The anchor query also specifies the attributes of the returned anchor tuples that contain location information. For example, the `funrefs` query corresponding to `fundlink` would return tuples containing line numbers, (in the attribute `refline`) and strings to be matched in each line (in the attribute `name2`). For example, a tuple might have a value 200 for the attribute `refline`, and the value “`isFixed`” for the attribute `name2`, indicating that an HTML link of the type `fundlink` should be inserted at the position where the string “`isFixed`” is matched on line 200 of the target document. Thus, this anchor tuple indicates the position of a call to the function `isFixed` occurring in the body of the target document containing the definition of the function `transmutable`.

Once the locations where the HTML links to be inserted are known, we can begin Step 3, the actual link insertion. Each link always contains the URL of the link insertion engine; in addition the link always includes a parameter-value pair specifying the type of the link (`fundlink` in this case). Additional parameter-value pairs, such as the name of the item that is linked (in our example, the function name `isFixed`) can also be inserted. The values are drawn from the anchor tuple that

is associated with each specific link insertion. If such additional link parameters are required, they are specified in the link specification, as a `mapparm` attribute. For example, the `fundlink` specification indicates that the value of the `name2` attribute of each anchor tuple is to be entered into the corresponding link as the value of the parameter `name`. Link insertion is the final step; the resulting HTML is returned as the result of the activation of the original link from the call to the function `transmutable`.

5 Retargeting to different software engineering environments

Two of CHIME’s key objectives are *compatibility* and *leverage*. First, we want to introduce WWW browsing into legacy software development environments, to reduce disruption to the ongoing projects; second, we would like to reuse the results of existing analysis tools whenever possible. However, this presents an implementation difficulty: the XRDB and the document repository may use a variety of different data models, schemas, and storage strategies. It is impossible to determine *a priori* the storage format and access methods that could be used. CHIME makes some assumptions and tradeoffs. First, we assume a relational model for the XRDB, and a “bag of flat streams” model for the document repository. Within the relational model, the CHIME language and retargeting machinery can accommodate different schemas for relations, attributes, etc.

The retargeting machinery in CHIME is based on a couple of design patterns [12]: the ADAPTER pattern and the FACTORY method. The basic CHIME link insertion engine is implemented in C++. It defines and uses some virtual base classes (interfaces) corresponding to a simple relational database access. To retarget CHIME to a specific XRDB, it is necessary to define an implementation class that inherits publicly (interface inheritance) from each of these base classes, and implements the virtual member functions. In the classic ADAPTER pattern (See [12], page 139), these implementation classes would inherit privately (implementation inheritance) from the actual implementation classes of the particular XRDB.

A synopsis of the public base classes for the XRDB and document interfaces, along with the details of some of the classes, are shown in Figure 5. There are several classes: `Document`, `DB` (for database), `Relation`, `Tuple`, and `Query`. Some of the classes are shown in more detail: thus, `class DB` has methods for opening, and closing databases, and evaluating a query. These are virtual methods to be implemented by a specific database interface. The result of evaluating a query (via method `DB::evaluate` is an instance of `class Relation` (details not shown), which provides ways of iterating through the relation, a tuple at a time, and for applying selections. The `class Tuple` has a method

```
class DB;
class Query;
class Relation;
class Tuple;
class Document;
class Repository;
class attrValue;
class Document {
public:
    virtual int
        getNextLine(String &) {};
    virtual String getId() {};
    virtual int linePos() {};
    virtual int docSize() {};
};

/* Factory Method examples */
Repository &makeRep(Map <String,
                    String> &parms);
Query &makeQuery(String &qstr);
DB &makeDB(Map <String,String> &parms);

class DB {
public:
    virtual void
        DBopen(Map <String,
                String> &parms)

    virtual void
        DBclose() {};
    virtual Relation*
        evaluate(const Query &) {};
};

class Tuple {
public:
    virtual attrValue
        getField(const String & attr ) {};
```

Figure 6: Some of the interface classes in the CHIME database and document interface

for accessing values of individual attributes. The class `attrValue` provides virtual methods for accessing values that are not in first normal form. Details are not shown here, but the abstraction used here corresponds to the COMPOSITE pattern (see [12], page 163), which provides a uniform way to access compositions of objects and individual objects. Likewise `class Document` has member functions for processing the individual lines in the document. The `Repository` class (not shown) has methods for opening and closing documents.

There are also several FACTORY METHODS (See [12], page 107), to create the top-level objects such as queries, the database and the document repository. These methods are to be implemented to return instances of the implementation classes that derive from `DB`, `repository` and `Query` interface classes discussed above. The use of the FACTORY METHOD allows the CHIME implementation to invoke the creational operations to create instances at the right times, while deferring the actual implementation to the specific retargeting.

6 Experience

CHIME has been retargeted to four different environments. Our first experience was using a simple C++ cross reference analyzer based on GEN++ [9], a C++ analyzer generator. Our second experiment was with an old version of CIA [6], a C cross referencing tool. Our third experiment was with Rigi [29], a reverse engineering environment. Finally, our most recent retargeting was to Acacia [5]. In this section, we describe our experience with these retargetings.

We first describe our experience with GEN++ and CIA++. Analyzing C and C++ programs is notoriously difficult because of syntactic and semantic irregularities caused by macros, typedefs, etc [8]. Given this, the popularity of the languages, and especially the CIA and Acacia tools, implementing a WWW browsing ca-

pability for C and C++ using existing analysis tools was an attractive proposition. Our experience with GEN++ was relatively easy: we built an analyzer that produced a simple cross referencing relation for functions and global variables into a flat file database. The document repository was simply the UnixTM file system. Our second effort, with CIA, exposed several problems: most notably, the relations were not in first normal form. The CIA family of tools [13, 5, 4] uses a highly compacted database format to store potentially quadratic sizes of cross-reference relations. This format sometimes stores a list of values (for example, a list of line numbers where a name is referenced) in an attribute. Our database interface, and the CHIME language were extended to accommodate this sort of non-first normal form attribute values in the XRDB. We implemented several links, including links from functions, variables and macros to their definitions, and views that presented different combinations of these links.

Our third and fourth efforts followed a major restructuring of the CHIME database interface: originally, we had used templates, but we changed that to use design patterns. This was motivated by the desire to release the CHIME software in binary form rather in the source form. The third retargeting, to Rigi, was relatively easy and straightforward. The fourth retargeting, to a newest member of the CIA family, Acacia, was easy as well.

7 Evaluation

The CHIME system is a *retargetable* domain specific framework intended to support the introduction of *customizable* web-based browsing into software development environments. The arguments for customizability and retargetability, have been presented earlier in this paper. As a result of our retargeting work with CHIME, we believe it is now a mature, reusable framework that is both retargetable and customizable. Our secondary goal in undertaking the retargeting trials described in the previous section was to address another hypothesis—that a retargetable framework such as CHIME is better than implementing an HTML link insertion engine from scratch for each software development environment. We now discuss our findings relating to this issue.

Our experience with the four retargeting efforts indicate that the bulk of the retargeting effort is implementing the database interface. With both the prior template-based interface, and the current ADAPTER-based interface, it takes around 200-250 lines of C++ code (LOC) to implement the interface—about 10-15 hours of coding, for some one knowledgeable about specific XRDB and document repository. Once this is complete, one can write specifications for several different links, and views that group these links in different ways. The database interface code can be leveraged across the different types of links and views, and even different link insertion en-

gines. The CHIME specifications are compiled into tables that are used by a core engine that inserts links. This engine is about 1300 LOC (the CHIME language compiler is not included in this count). The small size is due to heavy use of templates (both custom and the C++ standard template library[21]). Roughly a fifth of this code is for interpreting the tables—the rest does the string matching and manipulation, HTML encoding/decoding, database interaction and document access (via the ADAPTER interfaces), attribute/parameter mapping during link-insertion, etc. This core link-insertion engine code is leveraged for every retargeting of CHIME.

One alternative would be to implement a link insertion engine for each software development environment. Such an engine (for reasons described earlier) should be customizable, and allow ready addition of new links and views. This indicates the use of a layered architectural style [27, 24], with the bottom layers implementing the database access, and HTML encoding/decoding primitives, the middle layer providing primitives to support basic link insertion functionality, and the top layer for implementing different links. Without such a structure, it would be difficult to provide different sets of links for different browsing needs. Building a robust, mature, reusable infrastructure of this type (with the right abstractions) is difficult, and nearly impossible to engineer correctly the first time. The core CHIME engine provides precisely such an infrastructure. We find that we can leverage about 1K LOC, by writing the (about) 250 LOC ADAPTER interfaces to an existing XRDB/repository. The lines of code comparison, while providing a baseline for comparison, is not definitive. There are several other favourable (+) and unfavourable (-) factors, such as the effort to learn the CHIME language (-), the effort to design, develop and debug a reusable link-insertion framework similar to CHIME's from scratch (+), the difficulty of understanding and implementing the CHIME ADAPTER interfaces (-), the convenience of using the CHIME language (+), etc. Our experience, however, indicates that the CHIME framework offers a viable alternative to “from scratch” implementation of WWW browsing in a legacy software environment. In addition, once the retargeting interface is implemented, it's fairly simple and quick to produce as many customized links and views as desired.

8 Related Work

WWW is a research hotbed—there are too many projects and efforts to discuss here, so we provide comparisons to what we believe to be a representative sample. Storing and managing links separately from content is a key issue, and several systems have addressed this issue. The Microcosm [2] project at University of Leeds is specifically concerned with this problem. Hyperlinks

are stored in a separate *link base*, and are dynamically inserted into documents at the time they are dispensed by an HTTP server. This system has sophisticated features for handling link insertion into “cooked” documents. For example, Microcosm can read the hypertext content from a Microsoft WordTM document, perform the appropriate text formatting, and then insert the links. The delayed, dynamic link insertion policy allows the browsing context to determine which links are actually inserted. Microcosm has special link definition features that allow links to be parametrized; this allows links to be dependent on textual context in a semantic way rather than on specific locations in a file. This reduces the “tight binding” between content and links, allowing content to evolve somewhat more freely of the linkage. Their custom authoring facility has specialized features for creating links.

However, Microcosm uses a *specialized database* for storing the link information. While CHIME’s link insertion facilities are more limited, it can accept link information from much more varied sources; indeed, it is predicated on the assumption that this information is readily available for re-use from various existing sources. In the case of software documents, source code analyzers that generate such link information are hard to build and/or modify. The key contribution of CHIME in this regard is the retargetable database interface, as well as the CHIME specification language, which is used to specify the precise mapping of the link insertion information in a given database.

Constellation [30] is a broad conception of a distributed software development environment based on WWW technology. Typical HTTP clients, perhaps augmented with significant client side GUI extensions can be used to handle variable functionalities for editing, browsing, debugging, etc. On the server side, enhanced services are offered for source code control, debugging, and other development oriented activities. Integrated facilities for tele-conferencing are also envisioned. CHIME could help leverage existing WWW and software engineering assets to implement the functionality envisioned in [30].

Hyper-G [11] has some similarities to Microcosm, but it has also many advanced features for handling multilingual documents, as well as links into and out of multimedia content such as video. Hyper-G is also concerned with maintaining consistency of link information across distributed documents. Sophisticated, distributed, probabilistic algorithms have been developed to distribute the state of the link information. CHIME has a specific focus on link insertion for software documents; however, the link insertion capabilities are more flexible. Using the specification language and the retargetable database interface, it is possible to insert links based on information stored in a wide range of databases

derived using existing analysis tools.

Another noteworthy related system is Genera [18] (which is also a specification-driven environment like CHIME). Using Genera, one can specify detailed ways of formatting sophisticated object-oriented database content. Examples of Genera usage in the Human Genome Database are shown in the link from the citation [18]. Since much summary information about software objects might be available in cross-reference databases or software repositories, some of the features of Genera could be used in conjunction with CHIME for information display. We are exploring this possibility.

Complementary WWW tools

The WWW is a very fertile environment for innovation; numerous tools to supplement the basic web technology are available, and new ones emerge frequently. Browsing source code using the web can leverage these technologies. Any type of tool which can work with dynamically generated web pages can be useful in this context. For example, tools such as `htmldiff` [10] could be adapted and used to find and view in a web browser the precise differences in source files, including differences in inter-source file relationships that are exposed as links. Alerting systems such as NETMIND [22] are available to monitor and report changes. Convenient graphical tools for visualizing and using browsing histories can be helpful: tools such as [1] provide graphical visualization of browsing histories in distributed, shared context that can be used by teams co-operating over an intranet.

A novel application of the web is in distributed code inspections. Empirical studies [23] suggest that distributed inspections based on the WWW can save costs and reduce intervals while preserving effectiveness. New technologies such as GROUPWALK[20] allow a distributed group of browsers to follow a leader through a series of links. This is achieved by “slaving” a group of browsers to a master; browsing events are distributed via notification system. This has a clear application to distributed inspections. In addition, it could be used by an experienced developer to lead a distributed group of novices through a large body of source code, and explain the design and function.

A major project in the area of WWW-applications to software development is the Hypercode [15] effort at Columbia University. Hypercode is an architecture for distributed collaborative software development which leverages the web infra-structure. Hypercode has an open architecture that accommodates new tools and information sources [14]. On the one hand, CHIME could fit in nicely as a link insertion tool within the Hypercode framework. On the other hand, there is a difference in motivations. CHIME can introduce the incremental benefit of WWW-browsing into a legacy software develop-

ment environment, while leaving the rest of the environment unchanged. Hypercode is a completely novel approach, based on WWW-based integration of a distributed set of environments under a uniform WWW-based interface. The persistence of legacy environments offers a continuing and viable opportunity for incremental tools such as CHIME.

9 Conclusion, Limitations and Future Work

CHIME is a generator of HTML link-insertion engines. CHIME is *customizable* and *retargetable*. Different software engineering tasks require different strategies for exploring code; the ability to customize links allows the creation of browsing support to suit the specific needs of different tasks. The retargetability of CHIME allows us to introduce web-browsing into legacy software development environment with minimal effort and disruption. This has several benefits. First, the analysis tools and build procedures (which build the cross-referencing database) in the legacy environment can be leveraged. Second, programmers don't require re-training; most are familiar with the WWW infrastructure used in CHIME, and the rest of their environment is undisturbed. Finally, the constant stream of new WWW technologies for collaboration, notification, etc., can be leveraged in context with CHIME in legacy environments.

CHIME has several limitations. On the language and link-insertion side, it is a "pure browser". No facilities have been added as yet to integrate it with other elements of the environment, such as editors and debuggers. We are examining this issue carefully; such an integration would introduce a major new paradigm into legacy environments, which is not our original intention. On the database interface side, CHIME is strongly biased towards a relational model. Other models could be accommodated by appropriate implementation of the ADAPTER interface; however, this will be harder for non-relational databases. CHIME-based link insertion engines currently maintain all context information (e.g., the current default view) in the links. This wastes bandwidth. It would not be difficult to change the engines to use client-side persistent data (cookies) to keep this information. Another problem endemic to languages like C which have a pre-processor is that occasionally semantic cross-referencing information can be masked by a macro call. In this case, difficulties may occur during link insertion. Some XRDB's may provide enough information that can be used to handle the situation—but CHIME currently passes over such difficulties and keeps proceeding with link insertion. Finally, the current implementation of CHIME assumes a single location (URL) for the link insertion engine. This may not be well-suited for situations where the XRDB and the document repository are distributed. We are working on

approaches to address this problem.

For the immediate future, we are developing an HTML-based front end to the CHIME specification language that can hide some of the syntax from a user, and provide a more friendly user interface. Further on, we are interested in extending CHIME beyond software documents. The main difficulty is inserting links into "cooked" documents, based on information derived by the analysis of "raw" documents. We are exploring approaches based on the plug-in interfaces published for popular HTTP clients.

Acknowledgements: We would like to thank Tom Ball, Naser Barghouti, Alex Borgida, Henry Kautz, DeWayne Perry, and David Rosenblum for their helpful comments and suggestions.

REFERENCES

- [1] E. Z. Ayers and J. T. Stasko. Using graphic history in browsing the world wide web, fourth international world wide web. *World Wide Web*, 1995. <http://www.w3.org/pub/WWW/Journal/1/ayers.270/paper/270.html>.
- [2] Leslie Carr, David De Roure, Wendy Hall, and Gary Hill. The distributed link service: A tool for publishers, authors, and readers. In *Proc. Fourth International World Wide Web Conference*. O'Reilly Associates, 1995.
- [3] Yih-Farn Chen. Reverse engineering. In Balachander Krishnamurthy, editor, *Practical Reusable UNIX Software*, chapter 6. John Wiley & Sons, 1995.
- [4] Yih-Farn Chen, Glenn S. Fowler, Eleftherios Koutsosios, and Ryan S. Wallach. Ciao: A Graphical Navigator for Software and Document Repositories. In *International Conference on Software Maintenance*, 1995.
- [5] Yih-Farn Chen, Emden Gansner, and Eleftherios Koutsosios. A C++ Data Model Supporting Reachability Analysis and Dead Code Detection. In *Proc. Sixth European Software Engineering Conference and Fifth ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 414–431, 1997.
- [6] Yih-Farn Chen, Michael Nishimoto, and C. V. Ramamoorthy. The C Information Abstraction System. *IEEE Transactions on Software Engineering*, 16(3):325–334, March 1990.
- [7] T. A. Corbi. Program understanding: A challenge for the 1990's. *IBM Systems Journal*, 28(2), 1989.

- [8] P. Devanbu. Genoa—a language and front-end independent source code analyzer generator. In *Proc. Fourteenth International Conference on Software Engineering*, 1992.
- [9] P. Devanbu. The GEN++ page. <http://seclab.cs.-ucdavis.edu/~devanbu/genp>, 1998.
- [10] Fred Douglass, Thomas Ball, Yih-Farn Chen, and Eleftherios Koutsofios. The AT&T internet difference engine: Tracking and viewing changes on the web. *World Wide Web*, January 1998.
- [11] Frank Kappe *et al.* The hyper-g system. <http://www.iicm.tu-graz.ac.at/chyperg>, 1995.
- [12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1994.
- [13] Judith Grass and Y. F. Chen. The C++ Information Abstractor. In *The Second USENIX C++ Conference*, April 1990.
- [14] Gail E. Kaiser and Stephen E. Dossick. Xanth: An architecture for effective utilization of distributed heterogeneous information resources. Technical Report CUCS-003-98, Columbia University Department of Computer Science, March 1998.
- [15] Gail E. Kaiser, Stephen E. Dossick, Wenyu Jiang, and Jack Jingshuang Yang. An architecture for www-based hypercode environments. In *Proc. 1997 International Conference on Software Engineering*, 1997.
- [16] D. A. Ladd and J. C. Ramming. Programming the web: An application-oriented language for hypermedia. In *Proc. 4th Intl. World Wide Web Conference*, June 1995.
- [17] S. Letovsky. Cognitive processes in program comprehension. In *Proc. Second Workshop on Empirical Studies of Programmers*, Washington, DC, 1986. Ablex Publishers, Norwood, NJ.
- [18] S. Letovsky. Genera: A specification driven web/database gateway tool. <http://gdbdoc.-gdb.org/letovsky/wgen.html>, 1995.
- [19] S. Letovsky, J. Pinto, R. Lampert, and E. Soloway. A cognitive analysis of a code inspection. In *Proc. Second Workshop on Empirical Studies of Programmers*, Washington, DC, 1986. Ablex Publishers, Norwood, NJ.
- [20] W. S. Meeks, C. Brooks, and F. J. Hirsch. Staying in the loop: Multicast asynchronous notification for intranet webs. Technical report, The Open Group, 1997. <http://www.osf.org/www/waiba/-papers/aw3tc/notif.html>.
- [21] D. R. Musser and A. Saini. *STL Tutorial and Reference Guide*. Addison Wesley, 1996.
- [22] NETMIND, INC. Url minder service. <http://www.netmind.com>.
- [23] J. Perpich, D.E. Perry, A. Porter, L.G. Votta, and M.W. Wade. Anywhere, anytime code inspections: Using the web to remove inspection bottlenecks in large-scale software development. In *19th International Conference on Software Engineering*, 1997.
- [24] Dewayne E. Perry and Alexander L. Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, October 1992.
- [25] Symantec Personnel. <http://www.symantec.com>, 1995.
- [26] J. Christopher Ramming. *Proc. First Usenix Conference on Domain-Specific Languages*. The Usenix Association, October 1997. (Edited).
- [27] Mary Shaw and David Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall, 1996.
- [28] R. Stallman. Gnu emacs manual. <http://www.cs.-utah.edu/csinfo/texinfo/emacs19>, 1994.
- [29] Margaret-Anne Storey, Kenny Wong, and Hausi A. Muller. Rigi: A visualization environment for reverse engineering. In *Proc. 1997 International Conference on Software Engineering*, 1997.
- [30] Nino Vidovic and Dado Vrsalovic. Constellation: A web-based design framework for developing network applications. In *Proc. Fourth International World Wide Web Conference*. O'Reilly Associates, 1995.
- [31] S. Zeigler. Comparing development costs of c and ada. http://sw-eng-falls-church.va.us/AdaIC-/docs/reports/cada/cada_art.html.