Domain–Retargetable Reverse Engineering III: Layered Modeling

Scott R. Tilley

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890
stilley@sei.cmu.edu

Abstract

This paper describes ongoing work on a domain-retargetable reverse engineering environment which is used to aid the structural understanding of large information spaces. In particular, it presents a layered modeling approach to representing three classes of artifacts manipulated during the reverse engineering process. The approach provides a practical and extensible method of integrating existing tools and techniques into a reverse engineering environment by leveraging results from other areas, such as relational databases, hypertext, and conceptual modeling.

Keywords: domain-retargetable, modeling, reverse engineering.

1 Introduction

Reverse engineering technologies can be used to aid *hyperstructure understanding* (HSU): identifying artifacts and understanding their structural relationships in complex information webs. HSU is an objective rather than a well-defined process [1]. The prefix *hyper* is used to distinguish HSU from the in-the-small activity of understanding the internal structure of any single artifact; the focus is the analysis of overall system structure.

When any entity increases in size by several orders of magnitude, it changes in nature as well as in size [2]. When one attempts to understand a large body of information, the overall structure of the information space is just as important as the inner structure of any single artifact—if not more so. This is especially true when the number of artifacts in the domain is much larger than the size of each artifact.

Decomposition has long been recognized as a powerful tool for the analysis of large and complex systems. The technique of decomposing a system, studying the components, and then studying the interactions of those components has been used successfully in many areas of engineering and science [3]. For example, in the software engineering domain, modularization is a technique used to manage complexity by decomposing a large problem into several smaller ones. It can lead to simpler system structure, but it is not a panacea. It can lead to a proliferation of small parts; so much so that it is difficult to understand their inter-relationships [4]. Since good software engineering design suggests that modules be kept relatively small, the number of modules in a large system is significant [5]. For instance, in a system of 500,000 lines, with roughly 200 lines per module, there would be 2,500 modules—an order of magnitude more than there are lines of code in each module. At this scale, the understanding problem goes beyond the algorithms and data structures of computation. It moves into the realm of *architecture* and HSU: determining what modules comprise the system, how they are organized, and how they interact [6].

HSU involves inverse domain mapping. For example, in the program understanding domain, programmers make use of programming knowledge, domain knowledge, and comprehension strategies when attempting to understand a program. They extract syntactic knowledge from the source code and rely on programming knowledge to form semantic abstractions. To aid HSU, a reverse engineering environment must make this inverse mapping process easier by recovering lost information and making implicit information explicit.

The HSU process manipulates three types of "artifacts:"[1] (1) data: the factual information used

[1] The definitions used here are in accordance with Webster's online dictionary.

as a basis for reasoning, discussion, or calculation; (2) knowledge: the sum of what is known and represents the body of truth, information, and principles acquired; and (3) information: the communication or reception of knowledge obtained from investigation, study, or instruction. Based on these definitions, we can identify three canonical reverse engineering operations: (1) data gathering; (2) knowledge organization; and (3) information navigation, analysis, and presentation.

The Rigi[2] reverse engineering environment directly addresses each of these canonical activities. It does so by stratifying its support for each artifact category. It leverages existing technologies where applicable through a flexible tool integration mechanism. This approach enables the environment to build upon existing expertise and avoid "reinventing the wheel."

For example, the traditional approach to data gathering in a reverse engineering environment for program understanding is to parse the subject system's source code and extract complete abstract syntax trees with a large number of fine-grained syntactic objects and dependencies. To accomplish this, many researchers have spent an inordinate amount of time building parsers for various programming languages and dialects [8]. However, there already exists mature technology in the compiler arena to parse source code, perform syntactical analysis, and produce cross-reference and other information usable by other tools, such as debuggers. Rigi provides a simple mechanism to incorporate this data into the environment.

The next section describes the layered modeling paradigm. Section 3 outlines the support provided for layered modeling as part of the Rigi reverse engineering environment. Section 4 illustrates the use of layered modeling in two application domains: online documentation and program understanding. Section 5 summarizes the paper and briefly discusses future work.

2 Domain modeling

A *domain* is a problem area [9]. A *domain model* is a representation that captures the structure and composition of artifacts within a domain [10]. It may be constructed through *domain analysis*: the process of identifying, organizing, and representing the relevant information in a domain [11]. In fact, the construction of the domain model can precede reverse engineering (in which case it is used to guide the understanding process by supplying expected constructs), or it can be constructed during reverse engineering (if no previous knowledge about the domain was available). These two uses of domain models, as a guide to and as a product of reverse engineering, respectively, can also be combined in an iterative manner to support exploratory understanding.

A successful approach to reverse engineering must allow different domain models to be specified for different target domains. "Domains" in this sense is an over-burdened term. It includes different *application* domains, such as database systems, health information systems, and online documentation systems; *implementation* domains, including the application's implementation language; and the *reverse engineering* domain, in which the user applies reverse engineering to the problem of HSU.

During reverse engineering, artifacts must be put into a representation that facilitates efficient storage and retrieval, permits analysis of the artifacts and their relationships, and also reflects the users' perspective of the subject system's characteristics. These (sometimes conflicting) requirements suggest that a single technique for representing the different domains and the different categories of artifacts may not always be suitable. In its place, a layered approach to domain modeling may be used. For each type of artifact manipulated during each canonical reverse engineering activity, a different model may be used. The advantage of such an approach is that different technologies may be used to their strengths, while avoiding their weaknesses. For example, a relational model may be used for physical storage, a semantic network model for interactive exploration, and a conceptual model for representing domain-level information.

2.1 Data

Gathering data from the subject system is an essential step in reverse engineering. The raw data is used to identify a system's artifacts and relationships. Without it, higher-level abstractions cannot be constructed. For HSU, gathered data must be put into a representation that facilitates efficient storage and retrieval, permits analysis of the artifacts and relationships, and yet reflects the users' perspective of the subject system's structure. This requirement—the need to organize data in some well-defined and rigorous manner—led to the development of *data models* [12].

A data model captures the properties of an application needed to support the desired data-related processes. The result of data modeling is a representa-

[2] In this paper, "Rigi" refers to version V of the Rigi reverse engineering environment [7].

tion that has two components: (1) static properties that are defined in a *schema*; and (2) dynamic properties that are defined as specifications for transactions, queries, and reports. A schema consists of a definition of all application object types, including their attributes, relationships, and static constraints. Corresponding to the schema is a data repository called a *database*, an instance of the schema. A data model provides a formal basis for tools and techniques used to support data modeling.

The three best-known classical data models are the *hierarchical* data model, the *network* data model, and the *relational* data model [13]. The hierarchical data model is a direct extension of a primitive file-based data model; data is organized into simple tree structures. The network model is a superset of the hierarchical model; the objects need not be tree-structured. The relational model is quite different from the hierarchical or network model; it is based on the mathematical concept of a relation (a set of n-tuples), and organizes data as a collection of tables. All three classical data models are instances of the record-based logical data model.

2.2 Knowledge

Although well-suited to a computer environment, record-oriented data models are often semantically inadequate for modeling the application environment. They are highly machine-oriented and are organized for efficiency of storage and retrieval operations; ease of use for the non-programmer is of secondary importance. Typically, only two levels of abstraction are provided: the database schema, and the actual collection of records. There are no provisions to extend the levels to a more general hierarchy of types, meta-types, and instances, even though this extension would increase the model's expressive power and provide a mechanism which supports the reuse of common properties. The hierarchical and network models also do not support *semantic relativism*, which is the ability when modeling a system to view the elements and concepts representing it from different perspectives depending on the application. In particular, the concepts of entity, relationship, and attribute should be interchangeable. For these reasons, the classical data models are also known as *syntactic* data models.

The lack of abstraction mechanisms provided by the classical data models is particularly troublesome from an HSU point of view. Abstraction is a fundamental conceptual tool used for organizing information. It plays a key role in managing one of the fundamental problems with large-scale systems: coping with complexity [14]. When modeling such systems, the number of objects and relations in the knowledge base can grow very large. A large knowledge base—like a large software system—needs organizational principles to be understandable. Abstraction mechanisms such as classification (*instance-of*), aggregation (*part-of*), and generalization (*isa-a*) serve as *organization axes* for structuring the knowledge base [15]. Without them, a knowledge base can be as unmanageable as a program written in a language that has no abstraction facilities.

Conceptual modeling [16] is one approach to alleviating some of the deficiencies inherent in the classical data models. It is the activity of formally describing aspects of some information space for the purpose of understanding and communication. The fundamental characteristic of conceptual modeling is that is it closer to the human conceptualization of a problem domain than to a computer representation of the problem domain [17]. The emphasis is on knowledge organization (modeling entities and their semantic relationships) rather than on data organization. The descriptions that arise from conceptual modeling activities are intended to be used by humans—not machines. Concepts in a conceptual model are indexed by their semantic content. This differs from other data models, such as relational, where the indexing scheme is geared more towards optimal storage and information retrieval from the implementation perspective. This is one of the main reasons that conceptual modeling is eminently suited to HSU: the focus on the end user is paramount.

2.3 Information

Information processing represents the most important of the three canonical reverse engineering activities. While data gathering is required to begin the reverse engineering process, and knowledge organization is needed to structure the data into a conceptual model of the application domain, without the final step of information navigation, analysis, and presentation, there would be no benefit to HSU. The user navigates through the hyperspace that represents the information related to their application, analyzes this information with respect to domain-specific evaluation criteria, and uses various presentation mechanisms to clarify the resultant information. A general-purpose semantic network, represented as an attributed graph, is well suited to representing such structured sets of artifacts [18].

In its most basic form, a semantic network represents knowledge in terms of a collection of objects (representing concepts) and binary associations (representing binary relations over these concepts). According to this view, a *knowledge base* is a collection of objects and relations defined over them [19]. The semantics of the model are a careful definition of the meaning and usage of the nodes and arcs. Modifications to the knowledge base occur through the insertion or deletion of objects and the manipulation of relations.

The use of a network model has at least three advantages related to navigating, structuring, and visualizing the knowledge base. The first advantage is that the network structures that encode information may themselves serve as a guide for information retrieval [20]. The association between artifacts defines implicit access paths. Using this model, the information space is indexed by neighborhoods, while artifacts are retrieved through navigation guided by spatial and visual proximity cues.

The second advantage is the use of the organizational principles described in Section 2.2 to structure the knowledge base. Such abstraction mechanisms capture the natural structure of the artifacts in the system, their properties, and the relationships among them. They can also be used recursively to construct abstraction hierarchies. These structuring aids can be represented in the semantic network by typing both the nodes and the arcs.

The third advantage is that network representation schemes lend themselves to a graphical notation that can be used to depict knowledge bases and increase their understandability. Most human beings visualize structure graphically. For examples, designers often describe system architecture using block diagrams of the major system components and labels that refer to their major functions. Modern interactive systems with graphical display capabilities facilitate the direct manipulation, processing, and presentation of information in graphical form. As discussed in the next section, this is the approach taken in the Rigi environment.

3 Supporting layered models

To support domain-retargetability, Rigi IV (the Rigi reverse engineering environment circa 1992) has been reengineered into Rigi V through a three-step process. The first step was to make the central component (rigiedit) programmable through the addition of a scripting language [21]. The second step was to make the user interface customizable [22]. The third step was to integrate the layered modeling paradigm discussed in Section 2 into the environment.

Rigi V is a prototype realization of the PHSE:[3] an architecture for a meta reverse engineering environment [23]. It provides a basis upon which users construct domain-specific reverse engineering environments. It is instantiated for a particular application domain by *specializing* its conceptual model, by *extending* its core functionality, and by *personalizing* its user interface.

Layered modeling is directly supported in the PHSE (and thus in the Rigi implementation). The relational data model is the foundation upon which the conceptual model is constructed. The conceptual model represents domain knowledge; it acquires its semantics when the abstract conceptual model is further refined to reflect a particular application domain. A semantic network information model is used to represent selected artifacts of the subject system; this layer forms the core of the graphical interface to the underlying information space. The mechanisms used for each layer are outlined below.

3.1 Data

The data model is implemented as a store of binary relations, similar to the mechanism used by Beynon-Davies *et al* [24]. The binary relations that represent the database are stored in one or more files as a series of RSF (*R*igi *S*tandard *F*ormat) triples. Each triple is of the form *type subject object*. The interpretation of this relation is straightforward: a directed relation of the type 'type' is asserted between the *subject* and the *object*. Using this simple binary mechanism, both intensional and extensional information are represented. Intensional information defines the structure of the information (the schema of the knowledge base, interpreted as *meta-data*), while the extensional information comprises the actual occurrences of relationships (the instance of the knowledge base; interpreted as data).

For example, in the program understanding domain, the RSF triple *call foo bar* might indicate the existence of a call relationship between the function *foo* and the function *bar*. Likewise, the RSF triple *in function foo* could indicate that *foo* is an instance of the *function* class. Other structuring mechanisms (discussed in Section 2.2) can also be represented as RSF triples (for example, aggregation using *member* and specialization using *isa*).

Since every n-ary relation can be expressed as a conjunction of $n+1$ binary relations [25], the RSF mech-

[3] The acronym PHSE, pronounced "fuzzy," stands for *P*rogrammable *H*yper*S*tructure *E*ditor.

anism is sufficient to store the data required by the reverse engineering environment. Its simplicity also permits the facile translation to and from other data formats (such as Prolog facts), the incorporation of retrieval improvement techniques (such as inverted lists or B-trees), and the integration of dedicated tuple engines (such as qddb [26]) into the reverse engineering toolset.

3.2 Knowledge

The language Telos [27] was chosen for the knowledge modeling layer. Telos was selected over other conceptual modeling languages because it is more expressive with respect to attributes, it is extensible through its treatment of metaclasses, and it has already proven successful in other application domains. For example, it has been used to provide a structural framework for an authoring-in-the-large hypertext system [28], and to perform requirements analysis in a software engineering environment [29]. Moreover, an RSF representation of the knowledge base can be used to represent all Telos propositions, from meta classes to tokens, in a uniform manner.

The primitive units of Telos, individuals and attributes, have a direct mapping to the primitive units of hypertext, namely nodes and links. Furthermore, attributes are treated as "first class citizens" when it comes to the built-in domain-independent structuring mechanisms for aggregating, classifying, and generalizing artifacts. This results in a uniform framework and provides solutions to some of the issues discussed above.

Telos' meta-modeling facilities for describing structures unique to a domain means that it can represent a wide variety of conceptual models. Although not used in the current implementation, its assertion language, integrity constraint mechanism, and deductive rules for refining the structural knowledge of Telos, and its facilities for representing and reasoning about temporal knowledge, can also provide additional benefits.

As illustrated in Figure 1, the abstract conceptual model provided is relatively minimal, due in part to the fact that Telos is used in its specification. Telos provides most of the modeling and knowledge organization facilities required; only a few extra artifacts and attributes are needed. Users specialize this model to represent domain-specific knowledge.

The central component in the abstract conceptual model is the PHSEObject; everything is derived from it. It is a SimpleClass object, an instantiation of the PHSEObjectClass meta class, with two attributes: a unique identifier (the primary search key) and a set of annotations. No other constraint is placed on a PHSEObject. The only other special component is the PHSEWeb, an unordered collection of PHSEObjects; its use is discussed in Section 3.3.

3.3 Information

The information model used in Rigi is a special-purpose semantic network, represented as an attributed graph. In the model, both artifacts (represented as nodes) and relations (represented as arcs) are specializations of the PHSEObject class. Modifications to the repository occur through the insertion or deletion of artifacts and the manipulation of relations.

The information model consists of four objects: webs, nodes, links, and attributes. A web is a subset of the entire knowledge base that is related is some fashion. It is composed of typed nodes representing artifacts and typed arcs representing relations. Each node has a set of incoming arcs and a set of outgoing arcs. A node represents an artifact in the target domain. Links between nodes represent relations between artifacts.

Nodes and links may exist simultaneously in one or more webs. For example, in the program understanding domain, a node representing a C++ function may be part of a web of functions that call one another, part of a web of member functions for a class, and a web of overloaded functions. This permits object sharing and facilitates multiple views of the data.

Object semantics are provided through user-defined attribute/value pairs, which can be attached to nodes or links. Attribute/value pairs permit the organization of nodes and links into subgraphs and webs. For example, subsets of objects may be extracted from large graphs using filtering mechanisms based on attribute predicates.

Interrelated webs of objects form the cornerstone of the information model. They are are manipulated using rigiedit, a graphical, hypertext-oriented, multi-window hyperstructure editor. Portions (or all) of a web are viewed by the user as a *neighborhood*. A neighborhood is simply a collection of artifacts that are immediately accessible from the current perspective. It is graphically represented in the editor as a single window containing the artifacts. Artifacts can exist in any number of neighborhoods simultaneously, since neighborhoods are simply dynamically computed perspectives of the underlying knowledge base. This permits multiple, co-existing views of the information space. An example of rigiedit's use at the information model layer are provided in the next section.

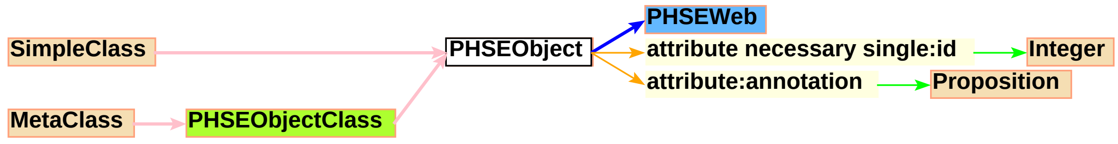

Figure 1: The PHSE abstract conceptual model

4 Using layered models

To illustrate the use of layered models within Rigi, two application domains will be used: online documentation and program understanding. To aid the user, default browsers are provided for each layer. The `qddb` tuple engine may be used as a data model browser to interrogate RSF streams. A graphical schema browser may be used to view the knowledge layer's conceptual schemas.[4] Rigiedit is used to manipulate the information layer. However, none of these tools are fixed; because of the programmable nature of the environment, users may replace these browsers (and other tools) with others of their own choosing [30].

4.1 Online documentation

Hypertext can be created from linearly organized online documents by retargeting the environment to support online information. The retargeting consists of three steps: organizing knowledge by specifying a domain model, gathering data via structural feature extraction, and navigating, analyzing, and presenting information by extending the editor. This section illustrates the use of layered modeling in this process using a representative LaTeX document. LaTeX was chosen as the text markup language since LaTeX documents are in plentiful supply, and thus the approach has immediate broad application.

The first step is the construction of the knowledge layer through the specification of a Telos conceptual model representing the LaTeX domain. A graphical representation of this schema is shown in Figure 2. The model does not describe all of LaTeX, just those features needed for illustration purposes. Nodes in the LaTeX domain model represent document artifacts, while links represent relations between these artifacts.

The second step is the construction of the data layer through the gathering of LaTeX artifacts from the subject document. This is done by using a text parsing system that extracts structure, relations, and actual text from LaTeX source and emits these artifacts and relations as an RSF stream. For example, LaTeX keywords such as `\cite` produce a *cite paragraph bibitem* tuple. The intention is not to duplicate the LaTeX parser in its entirety, but simply to extract features of relevance to HSU of online documents.

The third step is the manipulation (navigation, analysis, and presentation) of the information layer, which is a graphical representation of the gathered artifacts from the source document. The LaTeX artifacts are represented in the editor by their respective icons, as shown in Table 1. To illustrate this step, Figure 3 contains various views of the sample document. The window at the top contains the main control widget. The window at left represents the neighborhood of the 'root' of the document. The window on the right rep-

Artifact	Icon representation
`\document`	
`\part`	
`\chapter`	
`\section`	
`\subsection`	
`\subsubsection`	
`\par`	
`\bibliography`	
`\bibitem`	

Table 1: LaTeX artifacts and their icons

[4] The schemas shown in Figure 1, Figure 2, and Figure 4 were produced using this tool, which is based on a prototype from the University of Toronto.

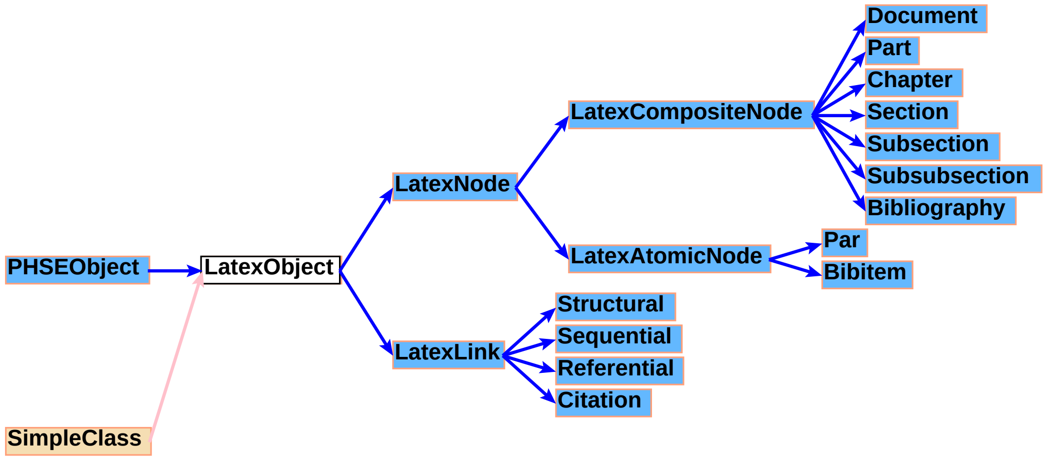

Figure 2: LaTeX conceptual model

resents the neighborhood near the artifact representing Chapter 2. A tree layout has been used to display the structural hierarchy below `chap2`, shown as the left-most icon in the figure. Although not discernible from the black and white image, the nodes and arcs are colorcoded (the colors are user-settable) to aid understanding.

The environment has been successfully used on several hypertexts. The repository representing the sample document used in the above example contains approximately 6,500 lines of RSF. Three other documents were placed online: a journal paper, a Ph.D. dissertation, and a textbook on software engineering. The three documents were chosen as illustrative examples because they represent documents of different sizes: the journal paper is roughly 40 pages, the dissertation 200 pages, and the textbook 400 pages. The journal paper is represented in 2,190 lines of RSF, the Ph.D. dissertation 7,542 lines, and the software engineering textbook 11,967 lines. In all cases, the environment's performance was acceptable for interactive use.

4.2 Program understanding

Program understanding, a more traditional application domain of reverse engineering, can also be supported by retargeting the environment appropriately. The same three steps are used as in the online documentation domain. This section illustrates the use of layered modeling in the process using the source code to SQL/DS as a reference system. SQL/DS is an ideal candidate for a layered modeling approach

Artifact	Icon representation
`system`	
`subsystem`	
`module`	
`proc`	
`data`	
`struct`	
`member`	

Table 2: PL/AS artifacts and their icons

since it represents a successful legacy software system of substantial size (over 3 million lines of code written in a proprietary language).

The first step is the construction of the knowledge layer through the specification of a Telos conceptual model representing the PL/AS domain. A graphical representation of this schema is shown in Figure 4. The model captures only those artifacts and relations required of a PL/AS program to support HSU. Nodes in the PL/AS domain model represent document artifacts, while links represent relations between these

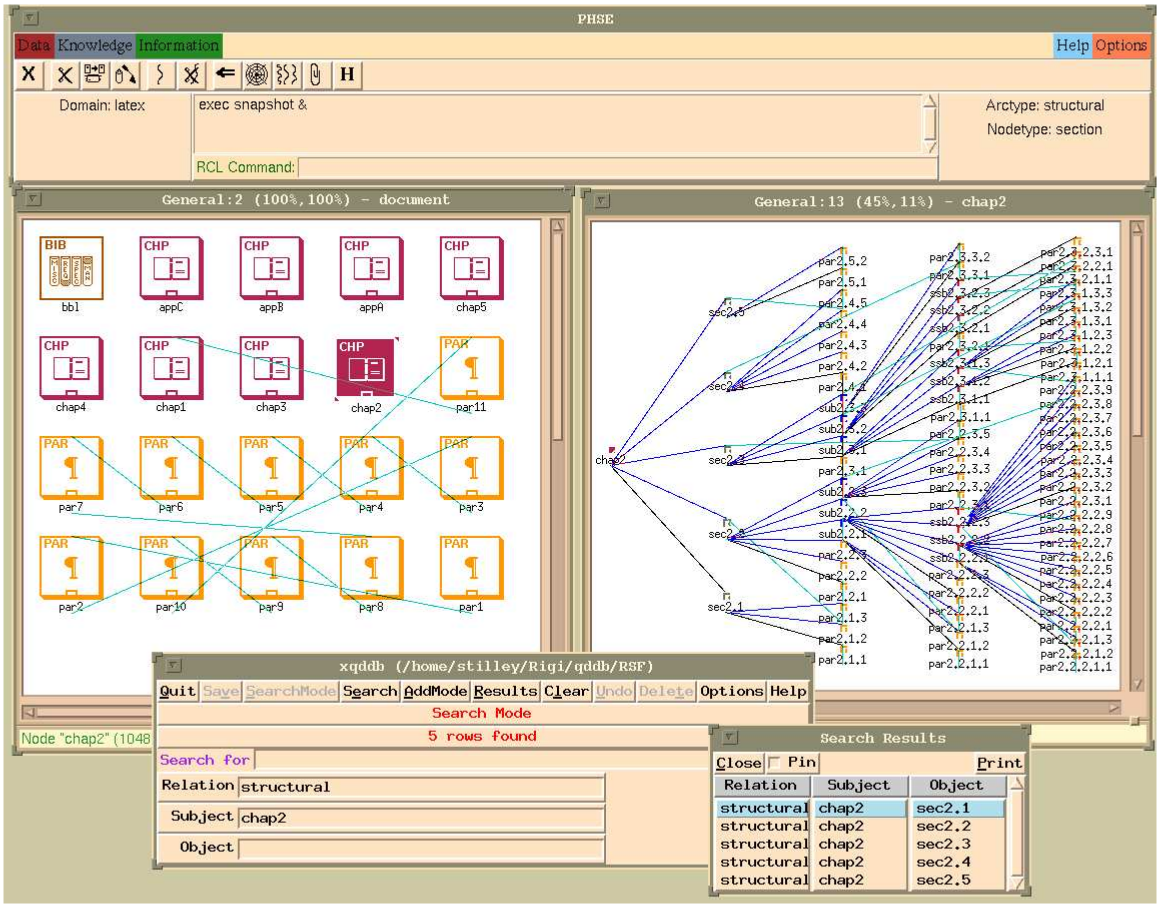

Figure 3: Different views of a LaTeX document

artifacts.

The second step is the construction of the data layer through the gathering of PL/AS artifacts from the subject document. This is done by loading an RSF stream representing the desired artifacts and relations of SQL/DS. This data was produced by a commercial reverse engineering tool and used to populate the data layer. A typical RSF tuple in this domain might represent the nesting relationship between structured variables and their components, represented as *member struct-name member-name*.

The third step is the manipulation of the information layer. The PL/AS artifacts are represented by their respective icons, as shown in Table 2. The facilities provided by the editor for navigating, analyzing, and presenting the information in this layer were used to redocument some of the structural aspects of the system [31].

The entire SQL/DS system is composed of over 100,000 RSF tuples at the data layer. For a repository of this size, some of the limitations of the prototype implementation of the environment become apparent. For example, some of the graphical routines for window management become sluggish when so many artifacts are simultaneously manipulated. However, many of these limitations are inherent in the X window library routines used. Moreover, it is not common for a user to reverse engineering the entire system at once; a more likely scenario involves partial reverse engineering of selected components.

5 Summary

Layering techniques have been used in many ways to improve understanding of large information spaces. For example, Rajlich *et al* describe the use of layered annotations to improve program comprehension [32].

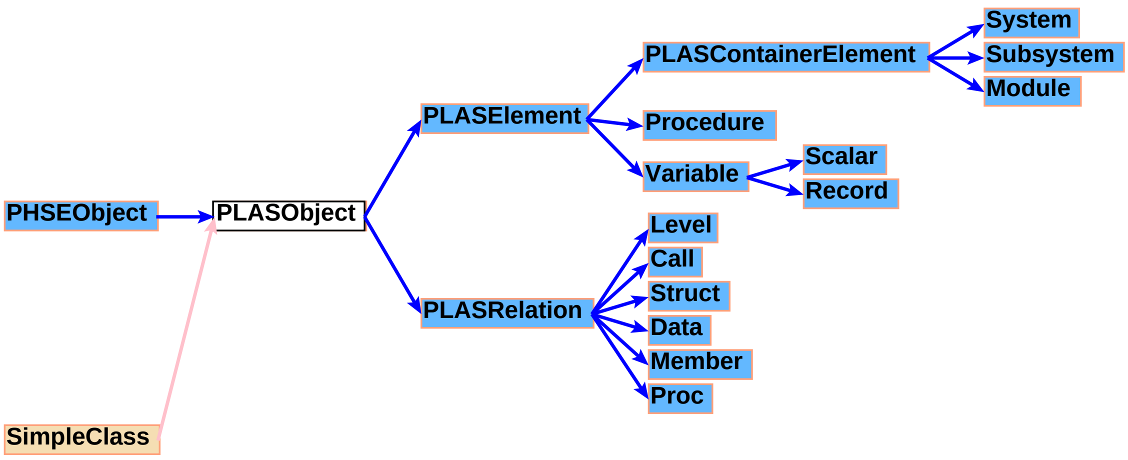

Figure 4: PL/AS schema

A prototype realization of the layered modeling approach proposed in this paper is found in the Rigi environment.

A relational model is used to represent the results of data gathering. The simple yet efficient format of RSF provides significant integration benefits to the reverse engineering environment. A scripting language based on Tcl [33] is used to enable the straight forward inclusion of externally produced data. A number of tools are available to browse the data in its relational format.

A conceptual modeling language is used to organize knowledge concerning the information space. The structuring facilities provided by the Telos language reduces the cognitive overhead associated with very large knowledge bases. An interactive schema browser may be used to visualize the conceptual model (and instances of it).

A semantic network is used to model information artifacts. A graphical representation of a semantic network facilitates the interactive navigation, analysis, and presentation of the information space. Thus, the network paradigm is well-suited to the exploratory nature of HSU.

In summary, the information space is constructed out of data artifacts, structured through the construction of conceptual models, and interactively explored in a hypertextual manner by representing neighborhoods as (parts of) semantic networks. The approach encourages the integration of additional tools into the reverse engineering environment, building on other research work in an evolutionary manner.

Future work will include the further investigation of the integration of other tools into the reverse engineering environment, the retargeting of the environment to new application domains, and the construction of the associated domains models. Preliminary results indicate that an extensible but integrated toolkit is required to support the multi-faceted analysis necessary for HSU.

Acknowledgments

Part of this research was performed while the author was at the University of Victoria. The guidance of Hausi Müller, the discussions with Kenny Wong, and the implementation efforts of Michael Whitney and Brian Corrie are gratefully acknowledged. This work was supported in part by the IBM Software Solutions Toronto Laboratory, the Science Council of British Columbia, the University of Victoria, and the U.S. Department of Defense.

References

[1] A. O'Hare and E. Troan. RE-Analyzer: From source code to structured analysis. *IBM Systems Journal*, 33(1):110–130, 1994.

[2] J. H. Walker. Authoring tools for complex document sets. In E. Barrett, editor, *The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information*. The MIT Press, 1989.

[3] P.-J. Courtois. On time and space decomposition of complex structures. *Communications of the ACM*, 28(6):590–603, June 1985.

[4] D. L. Parnas. Designing software for ease of extension and contraction. *IEEE Transactions on Software Engineering*, SE-5(2):128–137, March 1979.

[5] Z. L. Lichtman. Generation and consistency checking of design and program structures. *IEEE Transactions on Software Engineering*, SE-12(1):172–181, January 1986.

[6] D. Schefström and G. van den Broek, editors. *Tool Integration: Environments and Frameworks.* John Wiley & Sons, 1993.

[7] K. Wong, B. D. Corrie, H. A. Müller, M.-A. D. Storey, S. R. Tilley, and M. Whitney. Rigi V user's manual, 1994. Part of the Rigi distribution package.

[8] T. Cahil. Practical difficulties in developing tools for analysis of large application systems. In *3rd Reverse Engineering Forum (REF '92),* (Burlingon, MA; September 15-17, 1992), September 1992.

[9] J.-M. DeBaud, B. M. Moopen, and S. Rugaber. Domain analysis and reverse engineering. In *Proceedings of the International Conference on Software Maintenance* (Victoria, BC, Canada; September 19-23, 1994), pages 326–335, 1994.

[10] W. Tracz. Domain-specific software architecture (DSSA) frequently asked questions (FAQ). *ACM SIGSOFT Software Engineering Notes*, 19(2):52–56, April 1994.

[11] W. A. Rolling. A preliminary annotated bibliography on domain engineering. *ACM SIGSOFT Software Engineering Notes*, 19(3):82–84, July 1994.

[12] S. A. Borkin. *Data Models: A Semantic Approach for Database Systems.* The MIT Press, 1980.

[13] J. D. Ullman. *Principles of Database Systems.* Computer Science Press, Inc., 1980.

[14] F. P. Brooks Jr. No silver bullet: Essence and accidents of software engineering. *IEEE Computer*, 20(4):10–19, April 1987.

[15] J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine.* Addison-Wesley, 1988.

[16] M. L. Brodie, J. Mylopoulos, and J. W. Schmidt, editors. *On Conceptual Modelling: Perspectives from Artifical Intelligence, Databases, and Programming Languages.* Springer-Verlag, 1984.

[17] B. B. Kristensen and K. Østerbye. Conceptual modeling and programming languages. *ACM SIGPLAN Notices*, 29(9), September 1994.

[18] J. Rohrich. Graph attribution with multiple attribute grammars. *ACM SIGPLAN Notices*, 22(11):55–70, November 1987.

[19] J. Mylopoulos and H. J. Levesque. An overview of knowledge representation. In M. L. Brodie, J. Mylopoulos, and J. W. Schmidt, editors, *On Conceptual Modelling: Perspectives from Artifical Intelligence, Databases, and Programming Languages*, pages 3–17. Springer-Verlag, 1984.

[20] G. G. Hendrix. Encoding knowledge in partitioned networks. In N. V. Findler, editor, *Associative Networks (Representation and Use of Knowledge by Computers)*, pages 51–92. Academic Press, 1979.

[21] S. R. Tilley, M. J. Whitney, H. A. Müller, and M.-A. D. Storey. Personalized information structures. In *Proceedings of the 11th Annual International Conference on Systems Documentation (SIGDOC '93),* (Waterloo, Ontario; October 5-8, 1993), pages 325–337. ACM (Order Number 6139330), October 1993.

[22] S. R. Tilley. Domain-retargetable reverse engineering II: Personalized user interfaces. In *International Conference on Software Maintenance (ICSM '94),* (Victoria, BC; September 19-23, 1994), pages 336–342. IEEE Computer Society Press (Order Number 6330-02), September 1994.

[23] S. R. Tilley. *Domain-Retargetable Reverse Engineering.* PhD thesis, Department of Computer Science, University of Victoria, January 1995. Available as technical report DCS-234-IR.

[24] P. Beynon-Davies, D. Tudhope, C. Taylor, and C. Jones. A semantic database approach to knowledge-based hypermedia systems. *Information and Software Technology*, 36(6):323–329, 1994.

[25] R. Kowalski. *Logic for Problem Solving.* North-Holland, 1979.

[26] E. H. Herrin and R. A. Finkel. QDDB. University of Kentucky, 1994.

[27] J. Mylopoulos. Conceptual modelling and Telos. Technical Report DKBS-TR-91-3, Department of Computer Science, University of Toronto, November 1991.

[28] R. Sobiesiak. A hypertext authoring framework based on ceonceptual modelling. Master's thesis, University of Toronto, 1991.

[29] M. Jarke, J. Mylopoulos, J. W. Schmidt, and Y. Vassiliou. DAIDA: An environment for evolving software systems. Technical Report DKBS-TR-91-1, Department of Computer Science, University of Toronto, October 1991.

[30] S. R. Tilley, K. Wong, M.-A. D. Storey, and H. A. Müller. Programmable reverse engineering. *International Journal of Software Engineering and Knowledge Engineering*, 4(4):501–520, December 1994.

[31] K. Wong, S. R. Tilley, H. A. Müller, and M.-A. D. Storey. Structural redocumentation: A case study. *IEEE Software*, 12(1):46–54, January 1995.

[32] V. Rajlich, J. Doran, and R. T. Gudla. Layered explanations of software: A methodology for program comprehension. In *Proceedings of the Third Workshop on Program Comprehension (WPC '94),* (Washington, DC; November 14-15, 1994), pages 46–52, November 1994.

[33] J. K. Ousterhout. *An Introduction to Tcl and Tk.* Addison-Wesley, 1994.