From
Proceedings of the[1]

The 17th Annual International Computer Software & Applications Conference

November 1-5, 1993
Phoenix, Arizona

Integrating Abstraction Flexibility with Diverse Program Perspectives

Javed I. Khan and Isao Miyamoto

Software Engineering Research Laboratory (SERL)
University of Hawaii at Manoa
Keller 200B, ICS, Honolulu, HI-96822
javed@wiliki.eng.hawaii.edu

Abstract

Technically, because of the limited dimensions of the presentation media, and psychologically, because of the limited human attention span, the graphical program visualization in computer assisted reverse engineering becomes increasingly difficult with **scale**. This problem becomes compounded when multi-perspective visualization is added into consideration. In this research we present a system called HVIEW, which attempts to overcome the problem of scale by logical **decomposition**. HVIEW can present a graphical **visual** of a computer program at various **abstraction levels** from diverse **perspectives** as interactively requested by the user. HVIEW logically formulates abstract function and data objects to construct compressed intermediate program representations and organizes them into independent data and function hierarchies. Whereas, these hierarchies provide the mechanism of flexible abstraction, a user specifiable query filter creates an assortment of perspectives on the rendered *visual* of the modelled code.

1 Introduction

Conventional computer programs represent information in a textual format. However, the underlying process model of the software system, which is based on the complex convolution of the application specifications on the computing architecture, is fairly complex. Various graphical techniques have been used informally to facilitate program representation and analysis both in forward and reverse engineering, such as *flow graph*, *petri-net* based process models, *inter-procedural call graph* [3] etc. Individually most of these visual languages suffer from the inadequacy of their expressiveness to encapture numerous facets of a software system. Each of these languages specializes on specific aspects of a computer program. However, multi-perspective visualization plays a key role in the human concept formation process. More recently some researchers have proposed comprehensive schemes based on semantic networks as a richer tool for program information representation to support multi-perspective. For example MERA, developed at SERL over last four years, has been successfully used to represent a dozens of perspectives [2,10] about a software process.

A common problem with most of these program representation techniques, and specially the ones which are based on semantic networks, are that their apparent size and complexity grow intractably from the human perception point of view with the size and complexity of the software system. In the forward engineering process, the target system is generally a machine therefore, the scale, affects only quantitatively. However, in reverse engineering, where the purpose is to produce effective process perception in human expert, the scenario is quite different. The size and complexity of the visual information can seriously effect the quality and response efficiency of human understanding. Even a control graph with various data dependencies, related to a slim sized program (<500 line) presented on a full sized 2-D workstation screen can be overwhelming for human perception. Program visualization therefore, in addition of multi *perspective* support, requires *decomposability* of the representation formalism and the *reducibility* of the information volume in the visualization schema. In an attempt to achieve reduced representation of program information Basili et. al [1] proposed a formal validation based approach to create concise specifications. On the other hand, Howden [5] used a formal comment based approach to generate logical specification of various program structure. However, most of these pioneering approaches attempts to compress program information at an pre-engineered abstraction level from a sterio-type perspective.

In this research, we propose a decomposable graphical representation technique for program representation. The system, called as HVIEW (Hierarchical View), captures the code knowledge in a

semantic network based hierarchical knowledge structure and is capable of displaying the program to the human expert at various abstraction levels. A combination of semantic network and frame has been used to store complex information necessary for multi-perspective visualization. The network has been designed is such a way that it can be modularized through hierarchical decomposition. Instead of emphasizing any single view, the system can retrieve various facets of a code at pliable dimensions through dynamic abstraction.

This paper presents a brief design overview of HVIEW and the motivation behind it. Interested readers should consult [6,7] for technical details. In the following section we first explain our notion of program information abstraction. Section 3 presents the hierarchical organization scheme. Section 4 and 5 presents the internal semantic network based representation and decomposition schemes.

2 Program Abstraction

Program abstraction *is a process by which a program is presented in a* **concise** *form which facilitates program understanding by* **emphasizing** *the* **significant** *information of the target program.*

Information compression and **information selection** form two basic means of abstraction. *Compression* process uses all the components to generate the synthetic concise output. On the other hand, *selection* process ignores components which are generally less desired to generate the concise output. Both of the above abstraction processes are generally irreversible.

Our definition of program abstraction differs from Hartman's [4] *conservative abstraction* and Howden's [5] *formal abstraction* in the sense that we do not require abstraction process itself to be reversible. Rather we expect, an ideal program abstraction system to be capable of providing information with flexible abstraction level so that the user can retrieve full or part of the program with desired detail. Thus, the question of reversibility becomes irrelevant to our system.

3 Program Abstraction by Data and Function Hierarchy

To achieve the controlled *compression* and *selection* capability, we view a program as a composition of two symmetric hierarchies involving the two principal program entities; *instruction* and *data*. First we will consider the case of monolithic source code to explain the conceptual motivation behind the HVIEW design. Later we will show its natural extension to multi-moduler programs. A single monolithic source code provides a base model. The entities of the base model corresponds to the function and data entities and their relationships which are explicitly stated in the source code.

Hierarchy: HVIEW accepts the base model as input and gradually constructs abstract or composite data and function entities in a bottom up fashion and organizes them into two hierarchies named as (i) *Hierarchical Data Model* (HDM), and (ii) *Hierarchical Function Model* (HFM). Fig-1 shows the result of this process for an arbitrary computer program P which is made up of 5 statements and 4 data items. Let its statements are $F(P) = \{f_1(\delta_1), f_2(\delta_2), f_3(\delta_3) f_4(\delta_4), f_5(\delta_5)\}$, where, δ_i refers to the data set associated with each of the statements. Then $\bigcup_{i=1}^{5} \delta_i = D(P)$. And let $D(P) = \{d_i \mid 1 \le i \le 5\}$. In this figure the left and the right trees respectively denote configurations of the HDM and HFM of P. The leave nodes of the hierarchies denote the concrete (not abstract) entities of P and other nodes are generated abstract or composite entities.

The abstraction process can be viewed as the systematic compression of the sets F(P) and D(P) at each level by re-organizing or rediscovering implicit spatial relations among the function and data entities. This reduction process generates the opportunity of information compression at each abstraction level of program representation. Besides the systematic *compressibility*, further information *selectivity* are enforced by filtering the entities or concepts of immediate interest at the presentation level to achieve complete abstraction. However, both the *compression* and the *selectivity* are interactively determined by the human expert. HVIEW does not enforce any pre-engineered abstraction level on its user.

Rule Books: The creation of new entities during the process of hierarchy generation requires the computation of their individual specifications as well as the specifications of their relationships with the existing entities in the network. These computations are guided by domain knowledge inscribed in a set of 5 rule books, each of which specializes on the following five aspects.

i. Function Hierarchy Generation Rules (FHGR).
ii. Data Hierarchy Generation Rules (DHGR).
iii. Function Attribute Frame Specifier (FAFS).
iv. Data Attribute Frame Specifier (DAFS).
v. Linkage Attribute Frame Specifier (LAFS).

The first two rulebooks contain rules to generate the hierarchy. The other three rule books generate the frame specification of the new entities generated by the abstraction process. For example, as shown in Fig-2 the new relations (as shown by question marks) between the new data entities with existing function entities is derived using the Linkage Frame Specifier.

These rule books provide the isolation between the domain knowledge and the abstraction engine. The rule books may vary depending on the specific programming language, application domain or even due to the human engineers specific psychological and technical preferences.

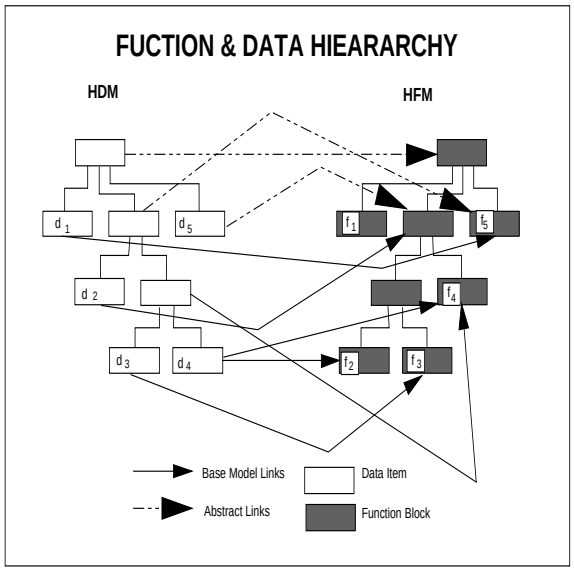

Fig-1

Flexible Abstraction: Now, the flexibility of abstraction will be illustrated. Any complete cut across the hierarchies forms a complete description of the program. The average height of the cut intuitively provides a measure of abstraction of that particular representation. Consider Fig-2 which repeats the same hierarchies of Fig-1, with two intuitive abstraction scales placed beside the two hierarchies, and two cuts across the hierarchies. In this example cut A is more abstract than cut B. The hierarchical representation of HVIEW makes it possible to transcend between different abstraction levels by gradually moving down and up the hierarchies. For example, when, more functional abstraction but detailed data dependency information is desirable, the system selects functional blocks from the upper level HFM and data items from the lower levels of HDM as shown. A symmetric inverse drift can generate a representation of P with more data abstraction and less function

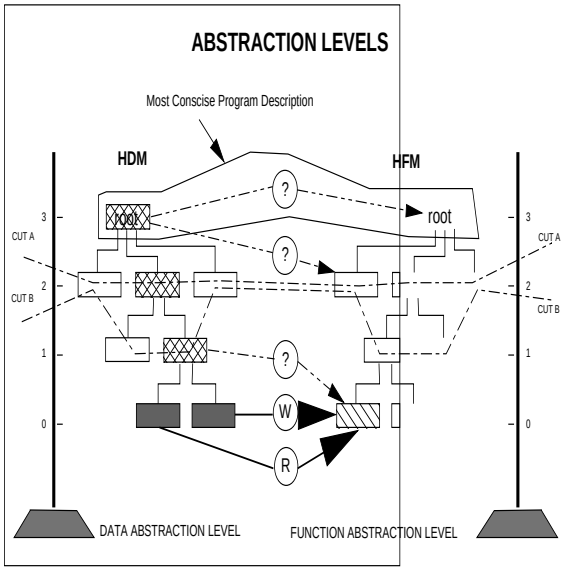

Fig-2

abstraction. At each level, the abstraction grammar provides the deduction ability of the inter-relations among all the entities in that level. At the highest level of the hierarchies, the abstraction is maximum. Thus, the description of the two root entities and their inter-relations, becomes the maximally abstract description of the program P.

Non-Monolithic Program: *Multi-module* (non monolithic) programs can be viewed as partially abstracted program, where, each of the sub-routine is a priori-abstracted module in HDM. Because of the difficulty of handling the complexity and size of monolithic program, the concept of modular programming has evolved over time as a natural extension towards abstraction. A programmer defined subroutine is a reflection of the abstraction process that has been performed in the mind of the human programmer for the sake of program organization during forward engineering. HVIEW design encaptures this pre-performed abstraction without any artificial distinction. Thus, multi-module programs only provides an advanced starting point where some of abstraction already has been done by the programmer. The generated base model includes not only base functional blocks but also some blocks in the upper level of abstraction. Therefore, HVIEW treats a program segment, a program or a collection of programs equivalently.

In a computer system a particular application program is a part of larger system program. In the abstraction process this means the derived hierarchical models are parts of their larger system models. On the other hand, the atomic statements and data items of higher level languages themselves are

abstract concepts relative to their machine code decomposition. Thus, any application program, subject to our understanding, is only a cross-section (cut) of the overall abstraction hierarchies which are both upward and downward expandable. The abstraction machine of HVIEW is intrinsically level independent in this hierarchy space. Depending on users requirement and only limited by the availability of resources, it constructs and manages the hierarchies and their interrelations.

4 Program Knowledge Representation

Now, we will describe the semantic net and frame based (Generic) Hierarchical Program Model (GHPM) formalism, that has been designed as a knowledge structure for HVIEW with decomposition support. It also can support concurrent process models. GHPM formalism has been defined using the meta language MERA [2]. Inquisitive readers may consult [8] for the theoretical aspects of GHPM in representing complete program knowledge that can ever be derived from a program code. Here we will provide only a brief sketch of the formalism.

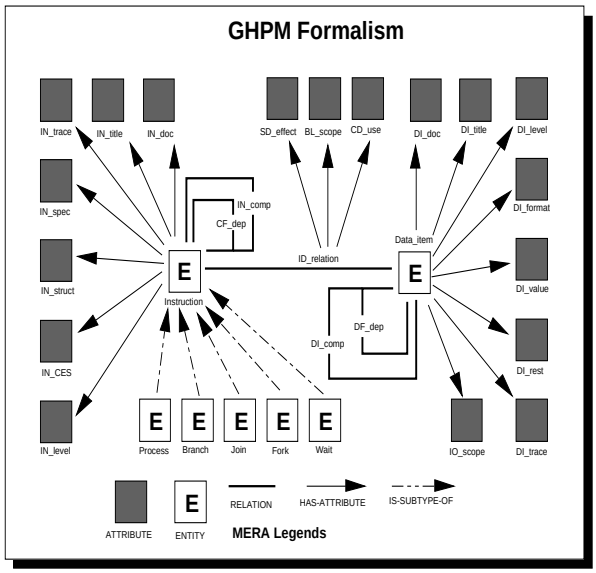

Fig-3

GHPM builds up a frame based semantic network representation of the input program. The information associated with the program concepts is defined and stored as *attribute: value* pair in the defining frames associated with the entity and relation objects. Fig-3 shows the definition of the formalism. Below we describe, the component concepts that is supported by the current definition of HVIEW.

The semantic network has two principal entities; i.INSTRUCTION and ii. DATA_ITEM. There are five sub-classes of INSTRUCTION type entities. HVIEW uses The CF_DEP along with (petri-net like) JOIN and FORK (null) instructions can represent and process parallel control flow. As shown in Fig-3, an instance of INSTRUCTION can be related with another instance of DATA_ITEM with an instance of relation ID_RELATION. Similarly, two instructions may be related by either or both CF_DEP and IN_COMP relations. Both of these are directed relations. Semantically CF_DEP refers to the control dependency among the INSTRUCTIONs and IN_COMP refers to the composition relation between a composite INSTRUCTION and its component INSTRUCTIONS in a hierarchy. Symmetrically, DATA items also have two analogous relations DF_DEP and DI_COMP.

The HAS_ATTRIBUTE relations in the meta graph show the components of the attribute Frames for these entities and relations. These attribute slots store the static properties of each of their corresponding entities. The value set for most of these attributes are generally part of the rulebook vocabulary. During, the base model generation, the atomic entities receives attribute values directly from the corresponds atomic statements present in the source code. The abstract entities generate their slot values during the abstraction process using the rulebooks.

5 HVIEW Architecture

The schematic architecture of HVIEW is shown in fig-4. Logically HVIEW has four principal assignments; (i) code parsing and generation of the base model, (ii) generation of hierarchies in the semantic net, (iii) controlled presentation of the semantic net, and (iv) interactive modification of the machine generated program model.

The data and instructions of the base model are processed through two independent processors (DATA and INST processors) for hierarchy generation. However, the activity of these processor units are coordinated through the first two rule books. Shortly, we will describe the processes. Once the hierarchical model is generated, the HVIEW Output Interface unit interactively controls the information presentation session through which suitably selected slice of the semantic network, and part of the attribute frame is graphically presented to the human expert. The hierarchical organization of the network (controlled by HDM and HFM controllers) , as well as the set of three filters jointly allow extremely flexible visualization of the network complexity at various abstraction levels. The interface controller

is supported by meraTalk [2] which is a Xwindow based graphical interfacing tool designed at SERL to support MERA based languages.

The machine generated automatic organization of the program knowledge in the hierarchical network can be interactively modified by human expert through the Interactive Hierarchy Modifier (IHM). IHM unit consults with the HiGen to maintain semantic network consistency.

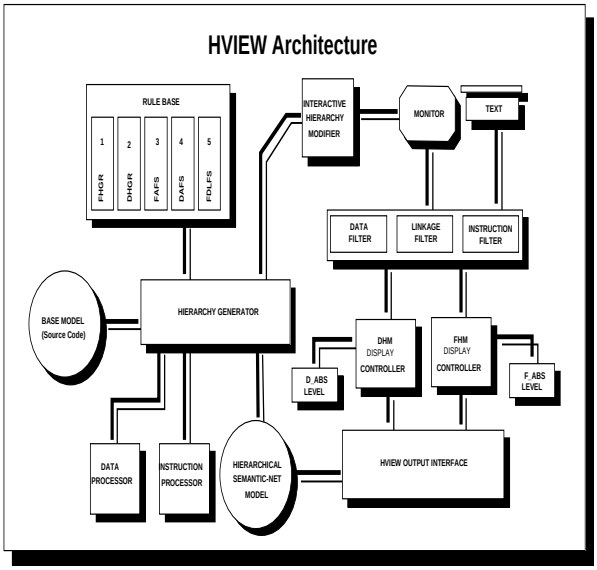

Fig-4

5.1 Generation of Hierarchy

Instruction Hierarchy: The instruction hierarchy is generated using a modified form of proper parsing of the control flow view of the base model. Proper parsing (a good description of the basic proper persing can be found in [4]) decomposes a proper (single entry, single exit) graph into sub-proper graphs. However, the basic proper persing is unable to decompose sequences. We have introduced subsequencing into proper parsing based on data coupling and concurrency analysis. This advanced parsing can handle system concurrency, parallel execution paths and can breakdown a long sequence of control flow into logically decomposed subsequences. In accordance with this modified technique, the following new set of prime propers has been used as HVIEW decomposition basis set in the Rule Book-1 (FHGR). The set has been found to be well tuned with the occurrence frequency in typical programs.

{SQ_N, TB_N, LP_3, LT_2, TL_2, ATYPE}

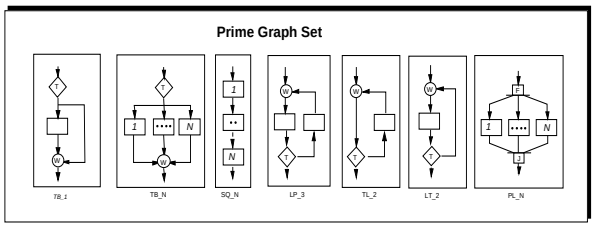

Fig-5

The structures of the first five of this set are shown in Fig-5. ATYPE refers to a class of primes which do not belong to the structured basis set. The decomposition process builds up the hierarchy of HFM, where the control flow view of the semantic network is recursively decomposed into sub-propers until all of them becomes prime.

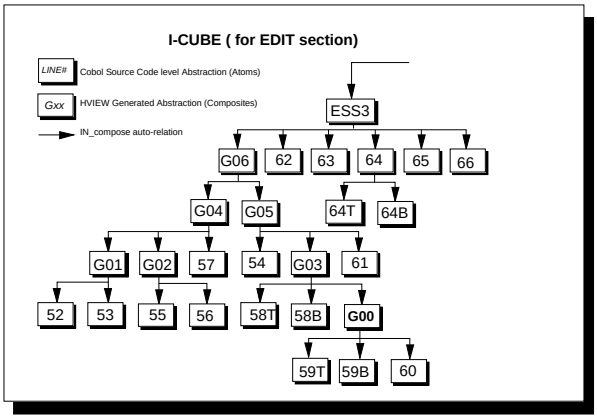

Fig-6(a)

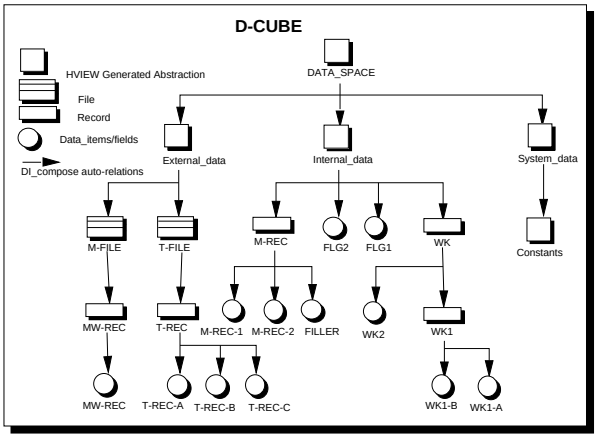

Fig-6(b)

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.      DBMS.
001100 DATA DIVISION.
001200 FILE SECTION.
001300 FD T-FILE.
001600 01   T-REC.
001700        03   T-REC-A          PIC 9(1).
001800        03   T-REC-B          PIC 9(4).
001900        03   T-REC-C          PIC 9(4).
002000 FD M-FILE.
002300 01   MW-REC                  PIC X(20).

002400 WORKING-STORAGE SECTION.
002500 01   M-REC.
002600        03   M-REC-1          PIC X(5).
002700        03   M-REC-2          PIC X(4).
002800        03   FILLER           PIC X(11).
002900 01   WK.
003000        03   WK1.
003100             05   WK1-A      PIC X(1).
003200             05   WK1-B      PIC X(4).
003300        03   WK2              PIC 9(4).
003400 77   FLG1                    PIC X.
003500 77   FLG2                    PIC X.

003600 PROCEDURE DIVISION.
003700 MAIN-SEC SECTION.
003800    PERFORM INIT-SEC.
003900    PERFORM EDIT-SEC UNTIL FLG1='1'.
004100    STOP RUN.

004200 INIT-SEC SECTION.
004300    MOVE '0' TO FLG1.
004500    OPEN INPUT T-FILE.
004600    OPEN OUTPUT M-FILE.
004700    READ T-FILE
004800       AT END MOVE '1' TO FLG1
004900              CLOSE T-FILE M-FILE.
005000     EXIT.

005100 EDIT-SEC SECTION.
005200    MOVE T-REC-A TO WK1-A.
005300    MOVE T-REC-B TO WK1-B.
005400    MOVE T-REC-C TO WK2.
005500    IF WK1-A = '3'
005600       THEN MOVE '1' TO FLG2.
005700    MOVE WK1 TO M-REC-1.
005800    IF WK2 > 100 ADD 100 TO WK2
005900             ELSE IF WK2 < 50 ADD 200 TO WK2
006000                      ELSE ADD 150 TO WK2
006100    MOVE WK2 TO M-REC-2.
006200    WRITE MW-REC FROM M-REC.
006300    READ T-FILE
006400       AT END MOVE '1' TO FLG1
006500              CLOSE T-FILE M-FILE.
006600    EXIT.
```

Fig-6(c)

Data Hierarchy: HVIEW places special emphasis in modelling and organizing the data space of a program. At the top level data hierarchy or HDM is generated using static decomposition based on the scope of data items. In the lower levels it performs rule based analysis to dynamically discover logical groups of data items with respect to their coupling and associated function decomposition. However, the static hierarchy explicitly stated thorough the data declaration segment always overrides the dynamic analysis. The well defined data declaration of COBOL generally produces satisfactory result. Fig-6(a) and Fig-6(b) presents a HDM and HFM of the example source code of Fig-6(c) generated by HVIEW.

5.2 Logical Specification

Both, the HDM and HFM generate new abstract entities. The specification of these entities are generated using other three rule books. The specification frames of the new abstract entities are generated from the composition type and pre-defined tree-tables, which specifies the generic connection between the reserve words of the code language.

The most critical is the abstract description of the link relation between the function and data items. The conventional relations, such as, Read/Write or Use/def etc. are insufficient to describe the abstract relations that evolves between abstract (composite) data items and abstract (composite) instructions. For, example, if a conditional composite block (like TB_2 in Fig-5) is reading a data item in one branch but writing it in the other branch, then neither "read" nor "write" can be specified as the relation between the composite block and the data item, without apriory knowledge about the result of the test condition. An extended and novel relation set has been developed for HVIEW to resolve the limitation of the classical dependency set in specifying abstract relations. This combines four primitive relations namely i. Read, ii. Write, iii. Read then Write, and iv. No Operation, using OR operators. These generate a complete set of 15 possible relation types. This novel set allows all the usual dependency analysis frequently performed in software engineering at higher abstraction level in an "optimistic" manner. Description of this extended set can be found in [7].

5.3 Visual Controls

The abstraction level and the perspective of the visuals is guided by the HDM and HFM controllers working in tandem with the filters. In a typical session of HVIEW, the pop-up EDIT window supports the graph *expansion* and *collapse* options which allows the easy traversal between the program (or program slice) representations with various abstraction levels. A Dialog Shell allows interactive attribute frame filtering. In general the attribute frame filters in conjunction with the hierarchies, provide flexible abstraction.

Filters on relation, instruction and data item provide multi-perspective viewing modes to the modelled code. For example, by selecting only one data item, and a "ID_relation" path, a typical *program slice* [9] can be viewed. The depth of the slice can be controlled by specifying k-closure on

the "ID_relation". A selection of only the INSTRUCTION and CF_DEP entities of GHMP generates a *flow graph* visual. Similarly, a selection on INSTRUCTION and IN_COMP can provide a visual super set equivalent to *inter-procedural call graph*. In fact a much more variety of perspectives on the visuals can obtained using the sophisticates path specification language which can support filter specification operators including *closure* and *negation*. The scope of the specification can be set on both entity type as well as on attributes. This provides a powerful tool for various sophisticated query options to fit user's perspective requirement.

6 Conclusion

The intended purpose of HVIEW is to assist human expert in the reverse engineering of fairly large but trivially complex computer codes. The critical algorithms of HVIEW have linear time complexity. The hierarchical organization of the internal semantic network based model allows HVIEW to produce program visuals at various abstraction levels to human experts according to user specification. In the conclusion we would like to summarize the following aspects of this work:

- ❏ HVIEW provides a decomposable representation of computer programs based on function and data hierarchy.
- ❏ Programs can be visualized at various depth and details due to the flexible abstraction capability of the hierarchical organization.
- ❏ Usual dependency analysis can be carried out at higher abstraction levels of HVIEW without backtracking to lower levels.
- ❏ HVIEW supports a wide range of program views or perspectives on the above abstraction levels. The internal semantic network contains all the relational concepts related to a program. The filters at the output interface provides the mechanism to generate clean visuals from the specified perspective.
- ❏ HVIEW can be used for a broad class of computer languages, such as multi-program COBOL, JCL, machine code. It can also naturally process parallel languages.

HVIEW is expected to be integrated with the *Software Maintainers Assistant* (SMA) system developed at SERL in recent years. At the end, the authors would like to thank members of SERL and specially Takeshi Nishimura, Bo Ma, Xiaobo Wang, Hai Huang, Koji Takeda and Taro Adachi of SERL for their support at various capacity in this work.

7 References

[1] V. R. Basili, S. K. Abd-El-Hafiz, G. Caldiera, Towards Automated Support for Extraction of Reusable Components, Proceeding of the IEEE Software Maintenance Conference, Sorento, 1991.

[2] k. Takeda, D. C. Chin and I. Miyamoto, MERA: Meta language for Software Engineering, Proc. Of the 4th Intl. Conf. on Software Engineering & Knowledge Engineering, June, 1992, Capri, Italy, pp495-502.

[3] M. J. Harrold, B. Malloy, A Unified Interprocedural Program Representation for Maintenance Environment, Proc. Conference on Software Maintenance, Sorrento Italy, Oct. 1991, pp138-147.

[4] J. E. Hartman, Automatic Control Understanding for Natural Programs, Ph.D. Dissertation, Univ. Of Texas at Austin, May 1991.

[5] W. E. Howdens & S. Pak, Abstraction and Problem Dependency in Reverse Engineering, Fujitsu Workshop on Program Maintenance, Tokyo, Japan, Sept, 92.

[6] Javed I. Khan, Program Abstraction Using Hierarchical Function and Data Model, Technical Report 92-21, Software Engineering Research Lab, Univ. Of Hawaii, July, 92.

[7] Javed I. Khan, Hierarchical Program Abstraction System Design Documents, Technical Report 92-30/92-31, Software Engineering Research Lab, Univ. Of Hawaii, Sept, 92.

[8] Javed I. Khan & Isao Miyamoto, Formalism for Hierarchical Organization and Flexible Abstraction of Program Knowledge, Proceedings of the 5th International Conf. on Software Engineering and Knowledge Engineering SEKE'93, San-Fransisco, p301-303, June 1993.

[9] M. Weiser, Program Slicing, IEEE Tran. On Software Engineering, v. SE-10, no.4, July 1984, pp352-357.

[10] A Case Study In Reverse Engineering, Technical Report 91-44, Software Engineering Research Lab, Univ. Of Hawaii, Sept, 92.