Reverse Engineering Meets Data Analysis*

Periklis Andritsos
University of Toronto
Department of Computer Science
10 King's College Road
Toronto, Ontario
M5S 3G4, Canada
(416) 978 5182
periklis@cs.toronto.edu

Renée J. Miller
University of Toronto
Department of Computer Science
6 King's College Road
Toronto, Ontario
M5S 3H5, Canada
(416) 946 3621
miller@cs.toronto.edu

ABSTRACT
We demonstrate how the data management techniques known as *On–Line Analytical Processing*, or *OLAP*, can be used to enhance the sophistication and range of *software reverse engineering* tools. This is the first comprehensive examination of the similarities and differences in these tasks both in how OLAP techniques meet (or fail to meet) the needs of reverse engineering and in how reverse engineering can be recast using data analysis. To permit the seamless integration of these technologies, we extend a multidimensional data model to manage dynamically changing dimensions (over which data can be aggregated). We use a case study of the Apache web server to show how our solutions permit an integrated view of data ranging from low level program analysis information to abstract, aggregate information. These high–level abstractions may be provided either by humans (perhaps using a visualization tool) or directly from reverse engineering tools or data mining techniques.

1 INTRODUCTION

Reverse engineering involves the identification of the components of a software system and their interdependencies, along with the extraction of system abstractions and design information [42]. Intuitively, reverse engineering helps developers understand the architecture of large software systems. Several tools have been designed and built towards this goal. The majority of legacy systems are undocumented, and even if documentation exists, reverse engineering tools help engineers compare the as-implemented with the as-documented or the as-designed structure of the underlined system [19]. Numerous commercial reverse engineering tools and research prototypes provide sophisticated syntactic and semantic analysis of programs including (to name just a few): CIA and CIAO [12, 11], Dali [28], PBS [18], Sapid [17], Refine [29], MediaDoc [38], Rigi [35] and the Bridge toolkit [27].

At the heart of the reverse engineering activity, lies the analysis of software systems and related data (documentation, facts extracted by static or dynamic program analysis tools or parsers, annotations, *etc.*) This analysis is designed to support software engineers who need to understand the software and make decisions about its maintenance, evolution, extension, or role in an enterprise. *Parsing-based* reverse engineering tools, which will be the focus of our study, extract relevant data (these include parsers, code analyzers and documentation analyzers), visualize this data (using graphical or textual reports), summarize the data (using filters to select interesting subsets or using groupings to aggregate the data into higher level abstractions), and query (or browse) the data to find information of interest to a specific activity [19]. Detailed summaries and comparisons of some of these tools appear in the research literature [4, 5, 19].

The decision-support and data-analysis goals of reverse engineering are closely akin to the goals of *On-Line Analytic Processing (OLAP)* and *Data warehousing* [24]. An increasing number of organizations have realized the competitive advantage that can be gained from the efficient access to accurate information. Information is a key component in the decision-making process of a business or enterprise. *Data warehouses* came into existence to meet the changing demands of the enterprises as *On-Line Transaction Processing (OLTP)* systems could not cover the analytical and decision-support needs of the enterprises' competitive environment. Data warehouses support analytical capabilities by providing an infrastructure for integrated, comprehensive, historical data supporting detailed analysis [24]. A data warehouse integrates the data of an enterprise (which is drawn from many, often incompatible, legacy application systems and applications) and provides sophisticated analysis, visualization and summarization tools.

Despite this close similarity in goals, OLAP techniques have not been widely adopted in reverse engineering tools. In this work, we examine the reasons for this situation. We identify specific limitations in OLAP technology that have prevented or restricted their use in software reverse engineering. We go on to address these limitations. Our main contribution is to bring data ware-

*This work was supported by NSERC and a Premier's Research Excellence Award (PREA).

houses and reverse engineering together. In particular, we will examine how software engineers can benefit from a multidimensional view of large software systems and how software analysts can benefit from access to hidden structures in their data, obtained by reverse engineering. The hidden structures involve graphs and aggregations over graphs. We show how these structures can effectively be explored and browsed using OLAP techniques.

In Section 2, we begin by describing the goals and motivation for reverse engineering tools. We briefly present the state-of-the art in tool design and present a specific case study to ground our discussion. In Section 3, we describe the goals and motivation for OLAP tools. In Section 4, we identify a number of limitations in OLAP technology that have prevented their adoption in reverse engineering. We propose solutions to these problems specifically designed to extend OLAP to meet the needs of reverse engineering. We consider how data analysis can be recast as a reverse engineering activity. The reverse engineering paradigm permits enhanced data analysis by providing a foundation for integrating newly discovered knowledge into the analysis. In Section 5, we show how reverse engineering can be recast as data analysis. We present the benefits of such an approach which provides reverse engineering tools dramatic improvements in functionality and extensibility. Finally, we present related work and our conclusions.

2 SOFTWARE REVERSE ENGINEERING

Many systems, when they age, become difficult to understand and maintain. Sometimes, this task also becomes inefficient due to its high cost. "A reverse engineering environment can manage the complexities of program understanding by helping the software engineers extract high–level information from low–level artifacts" [41].

A major effort has been undertaken in the software engineering community to produce tools that help program analysts uncover the hidden structure of legacy code. These systems are focused on performing the central reverse engineering tasks including the following [8, 19, 41, 43].

- **Program Analysis**. The analysis of source code and extraction of relevant information.
- **Plan Recognition**. The identification of common patterns. The patterns may be behavioral or structural, depending on the desired usage scenario.
- **Concept Assignment**. The discovery of human–oriented patterns in the system. This includes the identification of concepts in the source code and their relationship to program components.
- **Re-documentation**. The construction of documentation for an undocumented or legacy systems. Such documentation describes the architecture and functionality of the system.

Reverse engineering tools aid in the extraction or discovery of an already existing, but unknown, structure of a software system. This involves the decomposition of the system either in system–oriented or human–oriented parts that represent natural groupings.

The system analysis and management is based on the use of graph structures built on features of the original code, such as function calls, or based on complex static or dynamic analysis of the code. The manipulation and visualization of graphs has long been a mainstay of reverse engineering. More recently attention has focused on the introduction of powerful filtering, grouping and summarization capabilities.

Recent studies of reverse engineering tools have identified the following criteria for evaluating and comparing their capabilities [5, 4].

- **Analytical capabilities** are the features of the parser or code analyzer that is used to extract facts from the source system. Such features include the programming languages and environments supported, the ability to do incremental parsing (on evolving system versions), and the fault tolerance of the analyzer. A salient feature of these capabilities is the need for extensibility. The set of extracted facts in any reverse engineering tool will naturally be expanding and evolving.
- **Representation capabilities** enhance the tools' user-friendliness and usability. These capabilities vary from simple textual reports to more advanced graphical ones. The latter, permit the representation of the subject software system in *layered hierarchical views*. Representation capabilities may also include functions for filtering and grouping the data.
- **Querying and browsing capabilities** allow the user to navigate through the numerous levels of abstraction of a software system. These facilities permit a user to find specific information using queries or by navigating through the complex information representing a software system.
- **User views** permit a user to tailor her view of the software system, concentrating on information of interest. To be most useful, views should be queriable and persistent to enable users to browse and view previously determined information.

Current reverse engineering tools have a diverse set of capabilities that can be classified by these criteria. Our goal is to enhance some of these capabilities using new data analysis strategies. Hence, we shall focus primar-

ily on the last three capabilities, *i.e.* the representation and manipulation of the source code and related data, including semantic abstractions of programs. We will, however, also be concerned with ensuring these capabilities accommodate extensibility in the data that is collected and represented. Specifically, our enhancements are designed to support the following.

- Integrated use and analysis of data produced possibly by different tools, using different modeling assumptions, and gathered at different levels of abstraction.
- Evolution of data and metadata modeling dynamic semantic and syntactic program information. Our solutions are specifically designed to support data independence so that users may focus on the reverse engineering task and not be effected by changes in physical or logical structure of the data.

To illustrate some of the data management and data analysis capabilities of reverse engineering tools, we present a very small portion of an analysis of the Apache web server, consisting of about 83K lines of code. Our example is not intended to be comprehensive, but only to give a flavor for the type of data manipulation facilities used in reverse engineering. Furthermore, we must, due to space limits, focus on a single tool. For this presentation, we use a tool with data management facilities that are representative of (though certainly far from identical to) the facilities of other prototype and commercial tools. Otherwise, this choice should not be viewed as especially significant.

Example from Apache Case Study
Figure 1 shows the overview of a small portion of Apache. The overview was created using the Rigi reverse engineering tool [35]. The different subsystems are broken down based on the directory structure of the source code and are depicted in a hierarchy of levels.

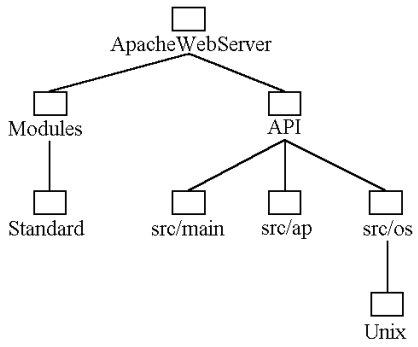

Figure 1: Overview of Apache architecture.

Rigi provides sophisticated facilities for refining and manipulating these views. A user may select restricted portions of the graph to view, select certain types of nodes or edges to view, collapse or aggregate subgraphs, search for nodes or edges based on their labels, and filter or refine a specific view of the system [35]. Selecting (clicking on) the *"src/main"* node gives all the artifacts of this subsystem. Filtering the nodes to include only *"functions"* and the edges to include only *"calls"*, we get a view of the function calls of this subsystem. The Rigi view for the *"src/main"* sub-system (Figure 2) is still too cluttered to be useful. So assume a user has refined the view further by selecting the function `ap_check_cmd_context()`. She might then select to see the view of Figure 3 which includes all functions that call `ap_check_cmd_context()`. For clarity, we have omitted the labels for many of the functions produced by Rigi.

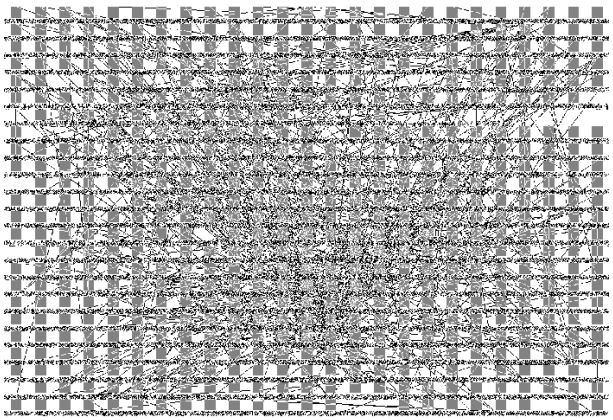

Figure 2: Function calls in *"src/main"* subsystem.

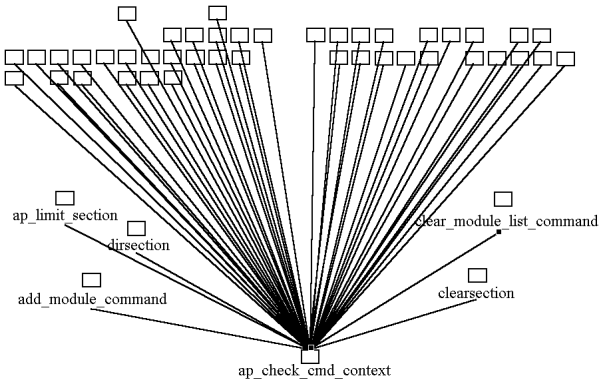

Figure 3: Function calls to `ap_check_cmd_context()`.

We can store this view, including both the view definition and its contents (the data objects). However, in Rigi, as in many tools and research prototypes [5], views are visual metaphors and are, in general, not first class objects in the system. They cannot always be composed and many of the actions that can be performed on the original database, cannot be performed on restricted views. As a simple example, suppose we have

stored the view of Figure 3. We then continued and created a second view by selecting `missing_endsection()` (Figure 4) and stored this view. If, now, we wish to ask the question *"what are the functions that call both* `ap_check_cmd_context()` *and* `missing_endsection()`*"* we cannot manipulate these views to see the response. These views can only be combined manually or using user programmed scripts.

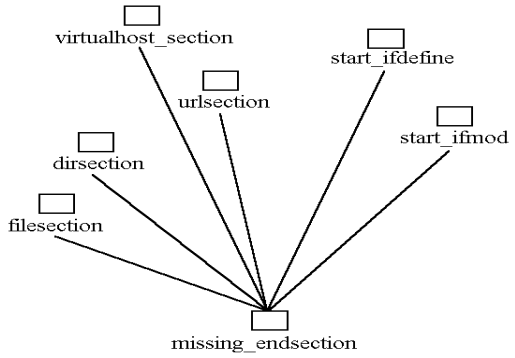

Figure 4: Function calls to `missing_endsection()`.

3 ON–LINE ANALYTICAL PROCESSING

In an OLAP system, data are presented to the user in a multidimensional model, which comprises one or more numerical measures and a collection of dimensions used to aggregate and summarize the measures [37]. A typical measure is the price or amount of a sale. Typical dimensions include location of the sale, product type, and time. An example data warehouse containing the dimensions: `location`, `time`, `product` and the fact table `sales` is depicted in Figure 5.

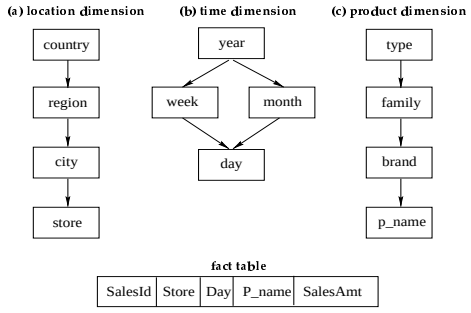

Figure 5: An example data warehouse.

Conceptually, the multidimensional model forms an n-dimensional data cube where each dimension is a separate axis of the cube [20]. OLAP operations, such as *roll-up* or *drill-down*, provide the means to navigate along the dimensions of a data cube.

While OLAP systems have the ability to answer "who?" and "what?" questions, it is their ability to answer "what if?" and "why?" that sets them apart from Database Management Systems [13]. OLAP enables decision-making about future actions. A typical OLAP calculation is more complex than simply summing data, for example: "What would be the effect on suit costs if fabric prices went down by 0.20/inch and transportation costs went up by 0.10/mile?"

OLAP tools provide managers with the information they need to make effective decisions about an organization's strategic directions. The key indicator of a successful OLAP application is its ability to provide information as needed, *i.e.*, its ability to provide "just-in-time" information for effective decision-making. Furthermore, due to the fact that data relationships may not be known in advance, the data model must be flexible. A truly flexible data model ensures that OLAP systems can respond to changing business requirements as needed for effective decision making.

Although OLAP applications are found in widely divergent functional areas, they all require the following key features [13, 2].

- **Multidimensional view of data**, which provides more than the ability to "slice and dice"; it gives the foundation for analytical processing through flexible access to information. Database design should not prejudice which operations can be performed on a dimension or how rapidly those operations are performed. Managers must be able to analyze data across any dimension, at any level of aggregation, with equal functionality and ease.
- **Calculation–intensive capabilities**. OLAP databases must be able to do more than simple aggregation. While aggregation along a hierarchy is important, there is more to analysis than simple data roll-ups. Examples of more complex calculations include share calculations (percentage of total) and allocations (which use hierarchies from a top-down perspective).
- **Time intelligence**. Time is an integral component of almost any analytical application. Time is a unique dimension because it is sequential in nature (January always comes before February). True OLAP systems understand the sequential nature of time. Business performance is almost always judged over time, for example, this month *vs.* last month, this month *vs.* the same month last year. In addition, OLAP systems are designed to reason about increments of time that lie in a partial, rather than total order. For example, years may be divided into both months and weeks where Week 5 is not in any order relationship ($\leq$, $=$ or $\geq$) with Month 1.

OLAP systems offer the data analyst tools to view, navigate and analyze data at different levels of abstrac-

tion. Abstraction is done using dimensions which provide ways of grouping data to hide unnecessary detail. Aggregation of numerical measures permits information from the more detailed data to be effectively summarized. A data analyst may, for example, sum–up sales values of all days of 1995 to see the yearly sales and, furthermore, compare them with those of 1994. If abnormalities are present in the totals, she may drill–down again, and view monthly, even individual sales again in order to figure out the cause of the problem.

4 RECASTING OLAP

From this discussion, a number of clear similarities and differences between OLAP and reverse engineering can be identified. Both aim to provide powerful visualization and summarization techniques for analysis and decision support. However, reverse engineering has focussed on graphical data while OLAP has focused on (flat) business data. In this section and the next, we address the following dual questions.

- Can reverse engineering benefit from OLAP techniques and perspective?
- Can OLAP benefit from reverse engineering techniques and perspective?

We first turn our attention to the task of enhancing the reverse engineering process using OLAP techniques. We will show in Section 5 that such a marriage will address some of the well-known limitations of reverse engineering tools that have been pointed out in the literature [5] and provide new opportunities for extending reverse engineering tools. However, before such a union is possible, we must address some important limitations in OLAP.

Multidimensional View of Graphs

OLAP techniques are designed to work over a flat base table of facts. The relationships between facts, that is the potential groupings, are captured in dimensions. These dimensions represent abstractions over the base data. This is in contrast to software data which has (often numerous) inherent graph structures. There may be a graph structure in both the base facts (for example, information about function calls) and in the dimensions themselves (for example, directory inclusions).

To illustrate this, consider a data warehouse of module dependency graphs [23]. The graphs may be modeled by a table of nodes representing modules together with their descriptive attributes (perhaps version number, owner, parameters, *etc.*) and a table or set of tables representing edges, that is dependencies between modules. Call information may be modeled by a table with the identifiers for the calling and called modules along with descriptive information about the call (perhaps its location, parameter bindings, *etc.*) A portion of this information is depicted in Figure 6. This schema is just a portion of a rich model of program semantics used to structure information gleaned by reverse engineering tools.

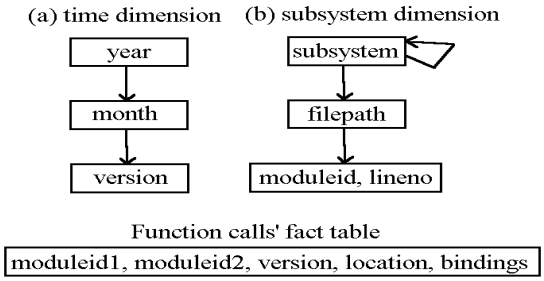

Figure 6: Multidimensional Model for Function Calls.

In addition to standard OLAP dimensions, the graph, which is encoded in the edge relationships of a fact table, plays an important role in aggregating and grouping information. For example, an analyst may request all modules reachable from a specific Module M, then drill-down into this information following additional call links. A key benefit of using a multidimensional model is that the resulting data can be grouped by any combination of dimensions.

Example from Case Study
Suppose that a user has drilled-down in the Apache data and has created the view of Figure 3. Rather than sorting through all this data visually, using OLAP the user can request to see this data aggregated by the subsystem in which each of the called functions resides or by the function's owner or by any dimension of the data. Furthermore, arbitrary filters can be applied to the data or dimensions to select sub-cubes of the data.

Dynamic Dimensions

A number of dimensions may be provided over software data. The dimensions may include tables encoding the total or partial order among versions, the hierarchy of file inclusions (encoding the directory structure), or architectural abstractions encoding the actual or discovered system architecture. Many of these dimensions may be encoded in fixed sets of tables as done in standard multidimensional models. However, some dimensions may not have a fixed depth. For example, the version information may not have a fixed (or bounded) number of levels. As a result, it may not be possible to encode this information in a fixed set of tables where each table represents one level of the dimension. However, such a representation is required in standard multidimensional models [37]. These models assume that while facts may change dynamically, the structure of dimensions are relatively static (both in dimension values and their relative orderings). Clearly, this is not the

case in software data where new dimension values and new levels within dimensions may be discovered as part of the reverse engineering process.

Current multidimensional models view dimensions as first class objects in the query language [25]. This permits flexible, data independent querying of dimensions. However, they require that dimensions be modeled as separate tables with each level in a dimension represented by a single table. To permit the use of arbitrary graphs as dimensions and to permit the use of graph structures within fact tables, we have made two extensions to the multidimensional model [1].

- We permit a single dimension table to represent multiple levels of a dimension. For example, in Figure 6, the subsystem table can represent an arbitrary number of levels of a subsystem hierarchy (or general graph). The dimension is encoded as a set of edges.
- We permit references between or within fact tables (in object-oriented terms, object identifiers and in relational terms, foreign keys) to be used to model dimensions. For example, the edges of the `calls` fact table can used as a dimension for grouping data.

Note that these changes represent a pivotal change in the way data is viewed in OLAP, a change required by and indeed inspired by reverse engineering. In OLAP, the goal of analysis is the understanding of business data represented in the fact table(s) and dimensions are relatively static tools for arriving at this understanding. In reverse engineering, the goal is to discover (or help the user create) new abstractions for organizing the data. Hence, these abstractions by their very nature must be dynamic and evolving.

Summarizing Data Without Measures

The computation and visualization of aggregations over numerical values or measures lies at the heart of OLAP systems. Queries are posed to find out totals, averages or numerical trends in data. Browsing is a means of arriving at the numerical summaries of interest. In contrast, in reverse engineering aggregations are used to help in browsing, locating information, and in understanding the data and underlying software system [22]. Measures may be missing or simple counts (lines of codes or number of qualifying functions) over which OLAP style statistical computations are not meaningful. Measures may also be textual and aggregation operations may be simple set operators. For example, instead of finding the average number of data types used by a set of functions, the measure may be the union (as a set) of all data types used. In either case, the groupings themselves, not the aggregates, are the focus. This difference was also noted in Web-based applications and in multimedia applications where aggregations are used as the "navigational vehicle" for finding and organizing relevant data [34]. Although the OLAP operators (drill-down, roll-up, slicing and dicing) can be used for both styles of applications, the use of these operators for interactive data browsing requires new query rewriting and evaluation techniques. Our previous work on IBM's Data-Web project has addressed these issues [34] and the query rewriting strategies have made their way into the marketplace [10]. The key insight in this work is that data abstractions (for example architectural abstractions of a software system) are made interactive. The interactive hierarchical visualizations of Data-Web are similar in spirit to the point and click interface of reverse engineering tools. However, the former permit the expression of arbitrary data cube style selections and operations [34].

Example from Case Study
Suppose a user is overwhelmed by the number of objects in a selected view, for example the view of Figure 3. She selects a dimension over which to perform the grouping. The system automatically writes a query to aggregate this data over the selected dimension. Suppose the subsystem dimension is selected, the query result, containing a set of subsystems, is displayed. By clicking on one of these subsystems, say S, the user can see all the functions in S that call `ap_check_cmd_context()`*. So the interactive style of reverse engineering tools is retained. The benefit is that this query rewriting is done automatically by the system. Any OLAP query can be specified in this way.*

Furthermore, the warehouse uses its knowledge of the query answer to choose the best rewriting. In this example, if dozens of functions that call `ap_check_cmd_context()` *are contained in S, the warehouse will not choose to return a cluttered view of all of these functions. Rather, the rewritten query will include an aggregation on the next level of the chosen dimension (in this case subsystem). So the response will have a list of subsystems of S each of which contains one or more functions that call* `ap_check_cmd_context()`*. Before drilling-down further, the user can view any aggregates available on these collections (perhaps number of functions, data types used, etc.)*

From Hierarchies to Graphs

Note that we permit general graphs as dimensions. Hence, the OLAP operators may be applied to the graph structures (such as dependency graphs) of software data. In standard OLAP, dimensions must be hierarchies (or in some proposals lattices). To ensure the OLAP operators retain a meaningful (and intuitive) semantics, we do not permit the application of some numerical aggregates over dimensions that contain cycles. However, many aggregates of interest in reverse engi-

neering (such as taking the union of textual attributes) remain meaningful over such dimensions. The full semantics of our extended model are presented in [1].

5 RECASTING REVERSE ENGINEERING

We now examine an integrated scenario where reverse engineering tools are coupled with a data warehouse. The data extracted and managed by the reverse engineering tool is modeled using the multidimensional model of Section 4 and the warehouse's OLAP facilities are used to support the data analysis required within reverse engineering tasks. We have built a prototype warehouse that manages data extracted by reverse analysis tools [1]. Our prototype uses IBM's solutions based on DB2, but it relies only on warehouse functionality that is common to most major vendors [7]. The model extensions proposed in Section 4 can be implemented by query and dimension rewrite facilities on top of the warehouse. This architecture addresses the following major shortcomings of current reverse engineering technology that have been cited in the literature [5].

View Management

Views are used in reverse engineering to permit a user to select a subset of data of interest to her and a specific organization for that information. Views may help in understanding and conceptualizing the system. Views also provide for flexible data independence permitting different users to customize their own view of the data and hiding physical and logical changes in the data structure.

Example from Case Study

Returning to Figure 6, a user may select modules that call a specific function `ap_check_cmd_context()` *and request that only the function id's and their file be returned (that is, other attributes should not be displayed as they are not of interest).*

Such a query can be formed using the Query-by-Example forms interface of the warehouse so the user need not know SQL or the underlying query language of the warehouse. This query, and optionally its results, can be saved as a view. Other views, including for example a second view containing all modules that call the function `missing_endsection()` can also be stored in the system. There are clear advantages to using the view facilities of the warehouse rather than a scripting language such as Tcl (used in Rigi) or a pattern matching language like Grok (used in PBS). Warehouse view languages are relationally complete permitting the expression of any subset or reorganization of the data. Furthermore, these languages have been extended to support the pattern matching (regular expression) style predicates that are often required in software engineering. This addresses a major shortcoming of current reverse engineering tools that support only a fixed set of commonly used views. Using declarative views, rather than procedural scripts, the underlying DBMS can optimize and efficiently evaluate views and compositions of views. This is particularly important for large complex views. Declarative views are composable and reusable. Any view can be queried and used in future queries in a transparent way. Furthermore, the results of a view can be used to more efficiently evaluate subsequent queries using the materialized view facilities of the warehouse [37].

Classifications

Classifications are groupings of low-level information into more abstract architectural objects [4]. Numerous proposals exist for developing classifications including (semi-)automated techniques based on concept analysis [32, 39, 40], clustering [3], or graph algorithms [26]. An important common trait of all these approaches is that the groupings they produced, whether flat groupings of functions into modules or hierarchically related groupings, can all be represented as dimensions in our multidimensional model [1]. These dimensions can then be compared and browsed using OLAP operators.

Example from Case Study

Concept analysis has been used in reverse engineering to group modules that are maximally similar based on a selected set of attributes [32, 39, 40]. More precisely, a concept is a set of objects O that share a specific set of attributes A, such that there are no other objects possessing all the attributes of A. This analysis produces a hierarchical grouping of modules. Applying concept analysis using attributes that describe a function's use of user-defined data types, we might discover a concept that includes all functions that use the data type `hlink_front`*. This concept may contain hundreds of functions. Selecting this concept, then drilling-down, the user would see a set of sub-concepts. For example, there may be three sub-concepts: the first representing all functions that use both the* `hlink_front` *and* `hlink_end` *data types; the second representing all functions that use* `hlink_front` *along with* `slink_front`*; and the third representing all functions that use* `hlink_front`*,* `file_front`*,* `file_end`*,* `dir_front` *and* `dir_end`*.*

The result of this analysis can be stored as a new dimension over modules. Hence, our proposed extended multidimensional model provides a consistent way to integrate the results of different analyses and classifications, including classifications developed manually by software engineers. Furthermore, users can explore and compare these classifications using powerful OLAP operators. All of these automated techniques are making use of syntactic information to deduce classifications. As a result, they all require semantic validation by a software engineer. The interactive modeling framework

we have presented provides a natural way for introducing and validating semantics and introducing domain knowledge.

In current tools, it is difficult to relate automatically generated classifications with those provided manually [5]. As a result, these tools do not offer the ability to produce a comprehensive picture of the whole system. Views that aggregate (group) previously discovered aggregations or classifications have to be generated manually in most cases [4]. In our proposal, all classifications are modeled and queried in an integrated way permitting the seamless integration of new classifications and abstractions.

Version Management
In their survey of reverse engineering tools, Bellay and Gall criticize all the tools for not providing the ability to reason about different versions of a system [5]. A software engineer is unable to view and compare different versions of the same program using these tools. Using our proposed architecture, version information is modeled as a natural dimension of the data. This permits any analysis to be done on a single version or across different versions of a system.

Example from Case Study
The original or intended structure of a system may be revealed by analyzing how classifications change through different versions of a system. Simple examples of time analysis include asking which functions call `missing_endsection()` *in version x of Apache but not in version y. More complex examples include aggregating data across multiple dimensions, including the version dimension. Continuing our example using concept analysis, suppose we have performed concept analysis on a version of the system. The concept lattice forms a classification or abstraction of the system. To judge the validity of the concepts, software metrics can be applied to the sets of functions within each group to measure the cohesion or complexity of the resulting group [32]. For example, suppose we have discovered there is an interesting concept representing all functions that access the data structures* `hlink_front` *and* `hlink_end`*. To further test the validity of this concept, we can use the concept dimension together with the version dimension and apply these same metrics to the different versions of these functions.*

From "What?" to "What If?"
So far we have been considering using filters and views to answer specific questions about the properties of the data. However, OLAP systems also have the ability to answer speculative questions. In particular, a user can perform an analysis on arbitrary subsets of the data or export these subsets for analysis by an external reverse engineering or mining tool.

Example from Case Study
Researchers have reported some success in using concept analysis to reverse engineer a modular (object-oriented) class structure from legacy code. However, studies on large systems have shown that this technique, by itself, will generally not reveal a tidy architectural abstraction for a large, legacy system [30]. Nevertheless, concept analysis can reveal structure where it exists and identify tangled piece of legacy code that require more manual analysis [32]. An example of a tangled concept is one containing hundreds of functions which cannot be decomposed into sub-concepts. Upon identifying such a concept, a user can use the OLAP operators to view the functions along different dimensions. This analysis may reveal that by ignoring one data type (perhaps a ubiquitous data type or a data type that has been used by different developers for different purposes), there may be a more natural classification or abstraction of the system. To confirm this hypothesis, the user can request all data about these functions (or the system as a whole) excluding the suspect data type. Concept analysis (or any reverse engineering analysis) can be performed on the resulting data. The new concept hierarchy can be imported as a new dimension and compared to other classifications.

Although our examples have focused on concept analysis, similar results hold for classifications deduced by other automated techniques including hierarchical clustering [3].

Additional Features
There are a number of other properties of our integrated architecture that, we outline briefly.

- **Extensibility and Data Exchange** We have already discussed the extensibility of the view architecture. In addition, our proposal supports the easy exporting or exchange of data between tools. This issue has received a great deal of attention recently including an ICSE 2000 Workshop on Standard Exchange Format. Warehouses typically support a number of features to permit the exchange of both relational and XML data. In particular, export facilities of our warehouse permit the software analysis data to be exported in XML. To support true tool integration however, this is not sufficient. Each tool requires data in a specified format or schema. The warehouse schema and data must be mapped and transformed into the required tool format. We have experimented with the use IBM's Clio schema integration tool to perform this integration [33]. The promising results of this study are reported elsewhere [33].
- **Scalability** Reverse Engineering tools use plain files or database management systems (DBMS) to store the artifacts and data generated by the parser

[5, 4]. Even tools that store low level component data within a DBMS, often represent views and view definitions in data structures or files and scripting languages (respectively) external to the DBMS. These structures are large and cumbersome to manipulate and compose. By storing views as first class objects within the DBMS, we are able to store and manipulate these views efficiently.

- **Data Visualization** Software reverse engineering tools provide sophisticated graph visualization facilities. However they have been criticized for the paucity of other styles of reports supported. Our integrated architecture permits the combination of graph-based visualizations with the spreadsheet-like visualizations of the warehouse. Our experience comparing the often cluttered graphs produced by tools (for example Figure 2), is that for views with large numbers of objects, flat textual representations may, for some tasks, be easier to manipulate than their richer graph-based counterparts. On the other hand, graph representations are indispensable for abstract views of the data and for views of smaller, more manageable components.

6 RELATED WORK

Reverse engineering and program comprehension tasks are often accomplished using direct manipulations of graph structures and graphical representations of source code [15, 9]. Properties of these graphs are used to break the source code into modules and subsystems leading to natural groupings of the initial system. The benefits of using data management systems in support of these tasks are well-known [6, 31, 36, 27]. The work done on the BRIDGE Project [27] demonstrated how database management systems can help in the consolidation of diverse metadata information related to a legacy system in order to support the migration of such a system. However, apart from a visual, Excel-like, interface there is no definition of a full query language that helps maintainers and developers to navigate through the meta-data. Furthermore, the BRIDGE work did not present a model for dynamically evolving the semantic model as new knowledge and program abstractions are created. We have provided such a model and have shown how an evolving semantic model of the program can efficiently and effectively be queried and managed. Our own work on the Assay Project [21] also uses data warehousing but this work did not explore the use of OLAP specifically. The use of abstraction and aggregation in reverse engineering have been proposed based on the Tarski Relational Algebra [22]. The operators of this algebra can be expressed by the OLAP operators we proposed. Our work and others have explored both the value and in some cases limitations of incorporating data mining techniques in reverse engineering [32, 14, 39, 30, 16]. However, this is the first systematic study of where data warehousing and OLAP techniques meet the challenging data management requirements of reverse engineering and where they fail.

7 CONCLUSIONS

We have identified limitations in the data modeling tools of OLAP that must be addressed to meet the data management requirements of reverse engineering. Specifically, multidimensional models assume that while facts may change dynamically, the structure of dimensions are relatively static. We showed both why this is required in current OLAP solutions and provided new solutions that effectively manage dynamic dimensions, including dimensions that involve general graph structures. We demonstrated how OLAP techniques can benefit from viewing these abstractions of data as dynamic, important components of the analysis process.

We also identified limitations in reverse engineering paradigms that can be addressed by using the flexible data modeling techniques of OLAP. We showed how these limitations could be overcome and the ensuing benefits from building reverse engineering tools on a solid data analysis foundation. Our approach complements reverse engineering tools by providing a powerful mechanism for integrating, manipulated and managing their results.

ACKNOWLEDGEMENTS We thank Johannes Martin for his help and for providing the Apache data used in our case study and prototype.

REFERENCES

[1] P. Andritsos. Recasting Program Reverse Engineering Through On-Line Analytical Processing. Master's thesis, University of Toronto, 2000.

[2] P. Andritsos and A. Papagianni. *On the development of a tool that supports OLAP queries.* Diploma thesis, Dept. of Electrical and Computer Engineering National Technical Univeristy of Athens, 1998.

[3] N. Anquetil, C. Fourrier, and T. C. Lethbridge. Experiments with hierarchical clustering algorithms as software remodularization methods. In *Working Conf. on Reverse Eng.*, 1999.

[4] M. N. Armstrong and C. Trudeau. Evaluating architectural extraction tools. In *Working Conf. on Reverse Eng.*, pages 30–39, Oct 1998.

[5] B. Bellay and H. Gall. A comparison of four reverse engineering tools. In I. Baxter, A. Quilici, and C. Verhoef, editors, *Proc. of the 4th Working Conf. on Reverse Eng.*, pages 2–11, Oct. 1997.

[6] M. Blaha, D. LaPlant, and E. Marvak. Requirements for repository software. In *Working Conf. on Reverse Eng.*, pages 164–173, Oct 1998.

[7] C. Bontempo and G. Zagelow. IBM Data Warehouse Architecture. *CACM*, 41(9):38–48, 1998.

[8] A. W. Brown and K. C. Wallnau. A Framework for Evaluating Software Technology. *IEEE Software*, 13(5):39–49, Sept. 1996.

[9] G. Canfora, A. D. Lucia, G. A. D. Lucca, and A. R. Fasolino. Recovering the architectural design for software comprehension. In *Proc. of the 3rd IEEE Workshop on Program Comprehension*, pages 30–38. IEEE Computer Society Press, 1994.

[10] D. F. Carr. Intranet Portal Helps Companies Keep Tabs on Customers in Real Time. In *Internet World*, March 22 1999.

[11] Y. Chen, M. Nishimoto, and C. Ramamoorthy. The C information abstraction system. *IEEE Trans. on SW. Eng.*, 16(3):325–334, March 1990.

[12] Y.-F. Chen, G. S. Fowler, E. Koutsofios, and R. S. Wallach. Ciao: A Graphical Navigator for Software and Document Repositories. In *IEEE Proc. of the Int'l Conf. on Software Maintenance*, pages 66–75, Nice, France, Oct. 1995.

[13] O. Council. OLAP Council's White Paper. In *http://www.olapcouncil.org/*.

[14] H. Dayani-Fard and I. Jurisica. Reverse engineering by mining dynamic repositories. In *Working Conf. on Reverse Eng.*, pages 174–182, Oct 1998.

[15] U. De Carlini, A. De Lucia, G. A. Di Lucca, and G. Tortora. An integrated and interactive reverse engineering environment for existing software comprehension. In B. Fadini and V. Rajlich, editors, *Proc. of the IEEE Second Workshop on Program Comprehension*, pages 128–137, 1993.

[16] C. M. de Oca and D. L. Carver. A visual representation model for software subsystem decomposition. In *Working Conf. on Reverse Eng.*, pages 231–240, Oct 1998.

[17] Dept. of Information Engineering, Nagoya Univ., Japan. Sapid Tool. In *http://www.sapid.org/*.

[18] P. J. Finnigan, R. C. Holt, I. Kalas, S. Kerr, K. Kontogiannis, H. A. Müller, J. Mylopoulos, S. G. Perelgut, M. Stanley, and K. Wong. The Software Bookshelf. *IBM Systems Journal*, 36(4): 564–593, 1997.

[19] G. C. Gannod and B. H. Cheng. A framework for classifying and comparing software reverse engineering and design recovery techniques. In *Working Conf. on Reverse Eng.*, 1999.

[20] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group By, Cross-Tab and Sub-Totals. In *Int'l Conf. on Data Eng.*, 1996.

[21] M. J. Harrold, R. J. Miller, A. Porter, and G. Rothermel. A Collaborative Investigation of Program-Analysis-Based Testing and Maintenance. In *Int'l Workshop on Experimental Studies of Software Maintenance*, pages 51–56, Oct. 1997.

[22] R. Holt. Structural manipulations of software architecture using tarski relational algebra. In *Working Conf. on Reverse Eng.*, pages 210–219, 1998.

[23] S. Horwitz and T. Reps. The use of program dependence graphs in software engineering. In *Int'l Conf. on Software Eng.*, pages 392–411, May 1992.

[24] W. H. Inmon and R. D. Hackathorn. *Using the Data Warehouse*. John Wiley & Sons, Inc., 1994.

[25] H. V. Jagadish, L. V. S. Lakshmanan, and D. Srivastava. What can Hierarchies do for Data Warehouses? In *Proc. of the 25th Int'l Conf. on Very Large Data Bases, (VLDB)*, pages 530–541, Edinburgh, Scotland, UK, 7–10 Sept. 1999.

[26] C. Jermaine. Computing program modularizations using the k-cut method. In *Working Conf. on Reverse Eng.*, 1999.

[27] Juanita Walton Billings and Lynda Hodgson and Peter Aiken. The BRIDGE: A Toolkit Approach to Reverse Engineering System Metadata in Support of Migration to Enterprise Software. In *Advances in Conceptual Modeling ER'99 Workshop on Reverse Engineering in Information Systems*, pages 134–148, Paris, France, 15–17 Nov. 1999.

[28] R. Kazman and S. J. Carrière. Playing Detective: Reconstructing Software Architecture from Available Evidence. Technical Report, CMU/SEI-97-TR-010, Software Engineering Institute–Carnegie Mellon University, Pittsburg, PA 15213, Oct. 1997.

[29] Kestrel Institute. Refine Tool. In *http://www.kestrel.edu/*.

[30] C. Lindig and G. Snelting. Assessing modular structure of legacy code based on mathematical concept analysis. In *Proc. of the Int'l Conf. on Software Engineering*, Boston, MA, 17–23 May 1997. IEEE Computer Society Press.

[31] A. Mendelzon and J. Sametinger. Reverse engineering by visualizing and querying. *Software—Concepts and Tools*, 16(4):170–182, 1995.

[32] R. J. Miller and A. Gujarathi. Mining for Program Structure. *Int'l Journal on Software Eng. and Knowledge Eng.*, 9(5):499–517, 1999.

[33] R. J. Miller, L. M. Haas, and M. Hernández. Schema Mapping as Query Discovery. In *Int'l Conf. on Very Large Data Bases*, Sept. 2000.

[34] R. J. Miller, O. G. Tsatalos, and J. H. Williams. DataWeb: Customizable Database Publishing for the Web. *IEEE Multimedia*, 4(4):14–21, Oct 1997.

[35] H. A. Müller, M. A. Orgun, S. R. Tilley, and J. S. Uhl. A reverse engineering approach to subsystem structure identification. *Software Maintenance: Research and Practice*, 5(4):181–204, Dec. 1993.

[36] V. Narat. Using a relational database for software maintenance: a case study. In *Int'l Conf. on Software Maintenance*, pages 244–251, Sept. 1993.

[37] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill, 2nd edition, 2000.

[38] Rogelio Adobbati and W. Lewis Johnson and Stacy Marsella. PESCE: A visual generator for software understanding. In *Proc. of the 1999 International Conf. on Intelligent User Interfaces*, Posters/Demonstrations, page 195, 1999.

[39] M. Siff and T. Reps. Identifying Modules via Concept Analysis. In *Proc. of. the Int'l Conf. on Software Maintenance*, pages 170–179, Bari, Italy, Sept. 1997.

[40] G. Snelting. Concept Analysis—A New Framework for Program Understanding. In *Proc. of the ACM SIGPLAN/SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, pages 1–10, Montreal, Canada, 16 June 1998.

[41] S. Tilley. A Reverse–Engineering Environment Framework. Technical Report, CMU/SEI-98-TR-005, Software Engineering Institute–Carnegie Mellon University, Pittsburg, PA 15213, Apr. 1998.

[42] S. R. Tilley. Management Decision Support Through Reverse Engineering Technology. In *Proc. of CASCON'92*, pages 319–328, 9–11 Nov. 1992.

[43] J. B. Tran and R. C. Holt. Forward and Reverse Repair of Software Architecture. In *CASCON*, Toronto, Nov. 1999.