Architectural Element Matching Using Concept Analysis

Robert Waters, Spencer Rugaber, Gregory D. Abowd
College of Computing
Georgia Institute of Technology
{ watersr, spencer, abowd }@cc.gatech.edu

Abstract

A large portion of software development effort is focused on modification and evolution of existing software systems. To feed forward-engineering and design activities, analysts must first recover and synthesize a complete and consistent set of architectural representations. Architectural Synthesis is one method to build this representation. During the Architectural Synthesis of a software system, an analyst must combine information derived from a variety of sources (which we call perspectives). This combination process requires the analyst to make decisions about which elements in the perspectives denote the same underlying parts of the software system. We present an automated technique for matching these elements based upon a mathematical technique called concept analysis. This technique constructs a spectrum of matching relations using a lattice of concepts drawn from the perspectives and descriptive information about the system's application domain. The results show the promise of using concept analysis to match elements and aid in synthesizing a large number of perspectives.

1. Introduction

An analyst with a mission to evolve or modify a legacy system faces a daunting task. The architecture of the existing system must be recovered and understood so effective forward-engineering decisions can be made. Architectural recovery is performed using many techniques, each of which is concerned with some aspect of the existing system's architecture. Each technique therefore provides a perspective of what the architecture might actually look like. For a large, complex system there are many of these perspectives that can overwhelm the analyst with information. Our work focuses on the merging or synthesis of these perspectives towards a more complete and consistent set of representations. The specific contribution in this paper is a technique that provides automated support to the analyst for combining these perspectives and overcoming the information overload problem.

2. Architectural Synthesis

Architectural synthesis is the process we developed to manipulate and process architectural information relating to a legacy system. Synthesis is performed using a cycle of activities consisting of *extraction* (obtaining information about the architecture), *classification* (separating the information based upon the stakeholder concerns being addressed), *union* (combining all the information in the same category) and *fusion* (checking information across categories for consistency). Details of the synthesis process can be found in [9].

This paper will focus on the union phase of the synthesis process. We use three primary technologies to combine information during the union phase:

- Lexical: This matches names of elements similar to the approach used by Michail and Notkin for library reuse [7].
- Topological: This manipulates the architecture as a graph where components are nodes and connectors are edges. It is similar to the technique used by Kazman [5].
- Semantic approximation: This uses specific information from the problem domain to provide semantic information about the components and connectors in the architecture. We use the term *approximation* to differentiate our approach from more formal approaches to semantic definition of architectural elements using specification languages.

We now discuss our approach to semantic approximation of architectural elements. If one considers that an architecture is the highest abstraction of the software solution to a problem in a specific domain, then the semantics of the different elements can be expected to refer to common terms in that domain. If we obtain common domain terms and associate the terms with the elements they refer to, we might expect these sets of terms to give an approximation of the semantics of the element. We obtain the domain terms through a technique known as *dowsing* [2]. Dowsing searches domain documentation and source code to extract word sets based upon their frequency of occurrence. The remainder of this paper will discuss our approach to the partial automation of semantic approximation to aid an analyst in combining information from different architectural perspectives.

3. Concept Analysis

Formal Concept Analysis (FCA) is a relatively new approach that provides a conceptual tool for the analysis of data [1]. FCA uses lattice theory to provide a way to group and discuss objects based upon their common attributes.

The fundamental object of manipulation in concept analysis is the formal context (C) which is defined as the ordered triple (O, A, R) where O is the *extent* (or set of objects), A is the *intent* (or set of attributes) and R is a relation between the extent and intent and $R \subseteq O \times A$. The easiest way to visualize this relation is as a table where rows are indexed by objects and columns by attributes. If object *o* has attribute *a*, then the table entry (relation) is true (or $(o,a) \in R$); otherwise it is false.

From the relation R, a lattice is computed which allows examination of concepts shared among objects. A *concept* can be thought of as a maximal set of shared attributes. There are two mappings of importance, the set of all common attributes of a set of objects (σ) and the set of all objects which are common to a set of attributes (τ). More formally: $\sigma(X) = \{a \in A \mid (o,a) \in R, \forall o \in X\}$ and $\tau(X) = \{o \in O \mid (o,a) \in R, \forall a \in X\}$. Further details on the underlying theory of concept lattices can be found in Ganter and Wille [4].

4. Applying Concept Analysis

To apply concept analysis to the task of automating the comparison of the semantic approximation information across multiple perspectives we must first determine the sets that will form the formal concept. We chose to use the architectural *elements* (where an element is a component or connector) as the objects (O) and the domain terms as the attributes (A). The relation R is true if an architectural element is semantically related to the domain term.

The difficult task is not in determining how the formal context will be defined, but what the results of the mathematical manipulation mean. Since we are trying to use the computed concept lattice to match elements in different perspectives we first create a context out of the elements and attributes of two different perspectives. We then generate a concept lattice using the standard mathematical algorithm. Finally, we traverse the lattice to find the elements in the two perspectives that are the same and those that are different. This allows integration of multiple perspectives in a semiautomatic manner.

We now define several *matching relations* that can be analyzed as the lattice is traversed. Consider the case of two perspectives P_1 and P_2, each containing a set of architectural elements $\{e_i\}$. The matching relations are presented from highest to lowest confidence.

- EXACT(e_1,e_2). This relation is true if $\sigma(e_1)= \sigma(e_2)$ (that is the attributes, in this case domain terms, of the elements are equal.)
- SUBSUME(e_1,e_2). This relation is true if $\sigma(e_2) \subset \sigma(e_1)$ (that is the attributes of element e_2 are a proper subset of e_1.) This situation occurs when one perspective contains more information about a specific element that another.
- CONTAIN(e_1,e_2). Any component or connector within a specific representation may be decomposed into another representation made up of another set of elements. We refer to this set of elements as a subsystem of the component or connector which was decomposed. The CONTAIN relation is true if e_2 is part of the subsystem of e_1. This occurs when we can match e2 using EXACT, SUBSUME or OVERLAP to an element in the subsystem of e_1.
- OVERLAP(e_1,e_2). This relation is true if $(\sigma(e1) \cap \sigma(e2) \neq \emptyset) \wedge (\sigma(e1) - \sigma(e2) \neq \emptyset \wedge \sigma(e2) - \sigma(e2) \neq \emptyset)$ (that is e_1 has attribute values in common with e_2, but has other attribute values which are different.) Like SUBSUME, this situation occurs when the techniques producing the perspectives elicit different kinds of information.
- NOREL(e1,e2). This relation is true if $\sigma(e1) \cap \sigma(e2) = \emptyset$ (that is e1 and e2 have no apparent commonality).

A concrete example will help to illustrate more clearly the relationship between these matching relations and the computed concept lattice.

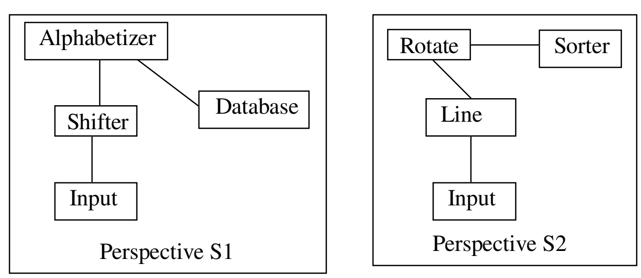

Figure 1: Simple Perspectives to Combine

5. Example

Consider a simple example where the two perspectives in Figure 1 have been extracted for the Key Word in Context (KWIC) problem. Each perspective contains four components and four connectors. Without loss of generality, we will limit the discussion to component matching

Table 1: Simple Architecture Formal Context

Domain Terms / Components	Change	IO	Data	Persistent	Lexeme	Reorder	Order	File	Lexical	Access
S1.Alphabetizer							X		X	
S1.Shifter	X				X					
S1.Input		X						X		X
S1.Database				X						
S2.Rotate					X	X				
S2.Line			X							
S2.Sorter							X		X	
S2.Input		X								

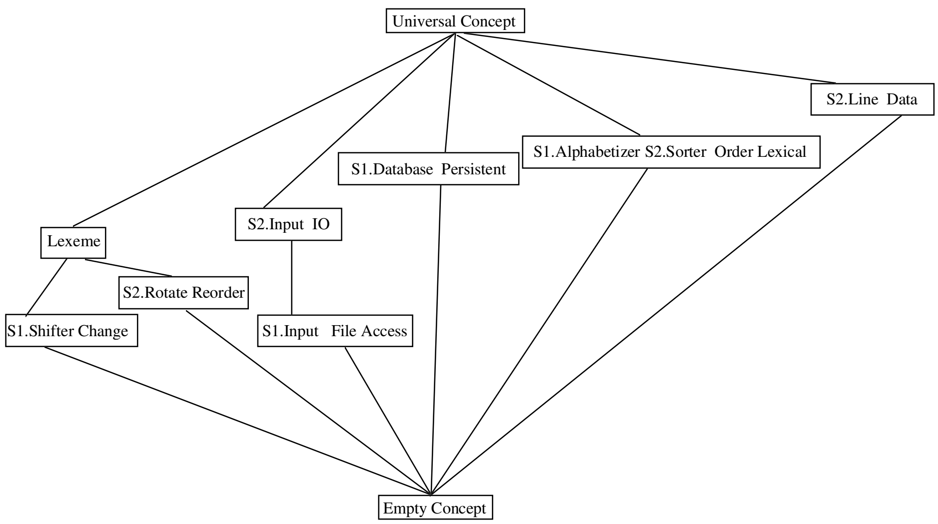

Figure 2: Simple Example Concept Lattice

only. This makes the relations easier to detect visually in the computed lattice.

Table 1 relates the domain terms dowsed from a set of KWIC problem descriptions to the elements in the two perspectives. We then compute the concept lattice using Christian Lindig's *concepts* package [6]. The computed lattice is presented in Figure 2. The top of the lattice is the universal concept and represents the concept where $\tau(X)=\{$all the objects$\}$. Likewise, the bottom of the lattice is the empty concept or the concept where $\sigma(X)=\{$all the attributes$\}$. A concept higher than another in the lattice is called a *superconcept*. We can now look at the lattice and visually see examples of the primary matching relations we previously defined.

If we examine the lattice node containing the Alphabetizer component from perspective S1 and the Sorter component from perspective S2 we see an example of the EXACT relation. Since the σ maps of each component are equal, they will both be found at the same concept in the lattice.

Likewise, the NOREL relation implies the intersection of the σ maps is empty. In the lattice, this means the least upper bound of a node is the universal concept (top) and the greatest lower bound is the empty concept (bottom). In Figure 2, there are two NOREL concepts, S1.Database and S2.Line.

Since the SUBSUME match relation means one σ map is a subset of another, we look for a concept which is a superconcept of another. We find just such a relationship in Figure 2 between S2.Input and S1.Input.

Finally, we can look for the OVERLAP match relation—admittedly the most difficult to find visually (and automatically, for that matter). Two concepts with an OVERLAP relation will have a superconcept in common. This relation can be seen in Figure 2 between S1.Shifter and S2.Rotate which both share the *Lexeme* superconcept.

This simple example did not demonstrate the CONTAIN match relation. The technique to compute this is the same since we are looking for an EXACT, SUBSUME, or OVERLAP where one of the elements matched is a subcomponent of another. The real difficulty in CONTAIN is determining when an element appears to be NOREL, but is in reality in a CONTAIN relation with some other element.

If only real-world applications produced simple lattices like that of Figure 2, there would be no need to have any further automated support. Unfortunately, the lattices for a real system may have a large number of concepts and the match relations (especially for SUBSUME and OVERLAP) are almost impossible to detect visually. For this reason we developed a lattice traversal algorithm to allow us to find these match relations automatically. The algorithm is quite simple and involves a depth-first search of the lattice and an analysis of the concept labels to determine the proper relationship.

The algorithm also computes an overlap measure, which is simply the percentage of domain term commonality. Currently this measure treats all domain terms as having equal weight. The software uses this measure to allow an analyst to specify a *match threshold*, above which all matches are automatically accepted.

6. Future Work and Conclusions

We are extending the technique described in this paper in several ways. First, as previously discussed, a complex concept lattice can overwhelm an analyst. We are researching methods to reduce the concept lattice to manageable levels. One technique we are examining uses subdirect and horizontal decompositions, as described by Funk, Lewien and Snelting [3].

We previously discussed a problem we have with the CONTAIN relation. We hope to apply the work in identification of modules by Siff and Reps [8] to the identification of new components in subsystems.

We also are examining techniques to automate the initial assignment of attributes to elements. Our preliminary solution is to augment the dowser to perform word proximity analysis. This improvement is necessary to make our element matching solution truly scalable.

Finally, we are integrating the automated algorithm for element matching into our REMORA toolkit for architectural synthesis. This tool provides an interactive workbench that an analyst can use to perform the entire architectural synthesis task. The tool combines perspectives in an automatic mode where new perspectives are built using just the results of our matching relations, or it may be used in interactive mode where the analyst accepts or rejects suggestions made by the algorithm.

Some form of automated assistance is necessary to help an analyst make sense of the many different perspectives that are recovered during a reverse engineering project. One method of providing this assistance is through the use of concept analysis. By observing the relationship of concepts to the elements in different perspectives, decisions can be made as to whether elements in two different perspectives are the same or different. Our matching relations and algorithm for traversing a lattice to look for these relations provides this assistance to an analyst in making architectural recovery decisions.

7. Acknowledgements

This work supports MORALE (Mission-Oriented Architectural Legacy Systems Evolution) a part of the DARPA EDCS project and was sponsored by the Defense Advanced Research Projects Agency, and Air Force Research Laboratory, Air Force Material Command, USAF, under agreement number F30602-96-2-0229.

8. References

[1] P. Burmeister, "Formal Concept Analysis with ConImp: Introduction to the Basic Features," available electronically from http://www.mathematik.tu-darmstadt.de/~burmeister/ConImpIntro.ps, 1998.

[2] R. Clayton, S. Rugaber, and L. Wills, "Dowsing: A Tool Framework for Domain-Oriented Browsing of Software Artifacts," *13th International Conference on Automated Software Engineering*, Honolulu, Hawaii, 1998.

[3] P. Funk, A. Lewien, and G. Snelting, "Algorithms for Concept Lattice Decomposition and their Application," : Informatik-Bericht Nr. 95-09, 1995.

[4] B. Ganter and R. Wille, *Formal Concept Analysis : Mathematical Foundations*. Berlin: Springer Verlag, 1999.

[5] R. Kazman and M. Burth, "Assessing Architectural Complexity," *2nd Euromicro Working Conference on Software Maintenance and Reengineering (CSMR '98)*, 1998.

[6] C. Lindig, "Concepts," available electronically at http://www.cs.tu-be.de/softech/people/lindig/software/index.html , 1999.

[7] A. Michail and D. Notkin, "Accessing Software Libraries by Browsing Similar Classes, Functions, and Relationships," *21st International Conference on Software Engineering*, Los Angeles, 1999, pp 463-472.

[8] M. Siff and T. Reps, "Identifying Modules Via Concept Analysis," *ICSM' 97: IEEE Conference on Software Maintenance*, Bari, Italy, 1997, pp. 170-179.

[9] R. Waters and G. Abowd, "Architectural Synthesis: Integrating Multiple Architectural Perspectives," in proceedings *Working Conference on Reverse Engineering (WCRE 99)*, Atlanta,Georgia,1999.