A MetaData Architecture for Case-Based Reasoning*

Sara Stoecklin, Daniel G. Schwartz, Erbil Yilmaz, Mahesh Patel
Department of Computer Science
Florida State University
Tallahassee, FL 32306, U.S.A.
{stoeckli, schwartz, yilmaz, mahpatel}@cs.fsu.edu

Abstract: ***Software design architectures created to meet generic requirements with specific requirements expressed in metadata are referred to as metadata architectures. In this paper we discuss a metadata architecture for a case-based reasoner using a reflective mechanism for implementation. The important aspect of this paper is that the it defines a reasoning system that is independent of the specific application domain, but which can be instantiated to create a reasoner for some given domain via specification of metadata appropriate for that domain. Thus the system can be used for many different types of domains. To date, it has been used to create case-based reasoners for (1) detecting possible intrusion packets in a network using past packet cases and their possible threats, (2) matching of a face in a facial case base using Eigenvalue features, and (3) solving network failure problems based on previous failure cases and their solutions.***

Keywords: Case-Based Reasoning, Metadata Architecture, Reflection

1 Introduction

The expectations for today's software make it increasingly difficult to design custom software for all applications. However, efforts to build generic systems to meet all user domains across various businesses are often not easily adaptable nor reusable. Some generic applications such as payroll, banking, and inventory have been successfully marketed. More recently some of the resource planning software has been successful. However, many applications do not yield themselves to generic solutions, so customized software is built. Many vendors do specialize in specific domains but their domain knowledge is the main component of the reusable software; little actual software components are reused.

With object-oriented programming concepts introduced in the 80's, polymorphism and inheritance began to be used to solve software problems. The introduction of new metadata programming that enables the user to change the behavior of the program without changing its code [1, 2, 5, 10] in the 90's allowed us to define systems using metadata. The use of reflection to implement these metadata programs using object-oriented languages allows major changes in software development practices. This article defines a system that exploits these changes.

2 MetaData Architectures

Metadata architectures are a recent innovation. They are defined as architectures that use metadata to adapt generalized systems at run-time. In our case the system is adapted to a new specific domain for reasoning. In the architecture, the application, in our case the reaonser, is developed in the form of underlying knowledge-level structural classes, constraints, and rules, rather than classes as in object-oriented programming [5]. This flexibility requires different levels of abstraction when developing the software. This level of abstraction makes the software more complex, but the flexibil-

*This work was supported by the US Army Research Office, grant number DAAD19-01-1-0502.

ity and reusability are especially high. Additionally, the amount of actual code required for the program decreases dramatically.

These architectures rely heavily on dynamic programming languages. Static languages, like C, require the programmer to establish the structure of the program in terms of data structures and data manipulation in the early stages of programming. However, in dynamic programming languages, a running program can add a new method to one of its classes without recompilation.

In metadata programs, the class structures are defined partially by setting a number of constraints over the class structure that must be satisfied. Examples of such constraints are method names and the types of the data structures that the methods will act upon. As a result, there are infinitely many class structures satisfying the constraints defined by a given metadata program. This is certainly the case in our MCBR. Further, behaviors of the methods are not implemented thoroughly in these programs. Rather the methods are implemented using metadata from the specific application domain.

3 Case Based Reasoning

Case based reasoning (CBR) systems store cases that represent past concrete experiences. Cases are stored as problem-solution pairs that describe complete episodes, or full experiences. Thus, in CBR, both knowledge elicitation and knowledge maintenance amount simply to identifying new cases and adding these to the library. For example, in a network intrusion detection problem, cases might represent full network packet information for the packets that are used in previous attacks. Therefore, such a case will have a feature for each field in the packet header and a feature for the packet payload.

The case-based problem solving process involves navigating through the solutions in a "solution space," guided by the similarity of a given problem to those represented by the cases stored in a case library. This is illustrated in Figure 1, adapted from [6]. As a new problem is encountered, the CBR system searches for those cases in the case library whose problem descriptions are similar, according to some similarity metric, to that of the given problem. The solution(s) of the most similar case(s) is (are) then used as a starting point for devising a solution to the new problem. The CBR system creates a solution to the new problem by adapting the solutions from the cases that were retrieved. This adaptation process is sometimes automatic, but typically requires human assistance. See [9] for a comprehensive overview of CBR including discussion of several applications.

4 Metadata CBR

Our system has made extensive use of the metadata software architecture techniques described in such works as [4, 5]. A key idea is to use a metadata dictionary for run-time method selection and run-time parameterization of methods from metadata. For instance, in our CBR framework, the metadata dictionary says which comparator methods are to be used for which case features during the similarity matching part of the case retrieval step. A metadata program is described as a partially defined, and not thoroughly implemented, class structure that sets some constraints over actual, full-functioned implementations [5]. Such a program is instantiated by providing the needed items for the metadata dictionary and the associated method definitions. As a result, there are infinitely many possible implementations, forming a family of programs that satisfy the constraints. The program uses two metadata dictionaries. The first, called the domain metadata, describes the domains usable in the system. For example, in the current domain metadata there are three systems, namely, network intrusion detection, facial recognition, and network fault detection. The data contained in these descriptions are the name of the system, the problem case base, and other metadata. The second, called the case metadata, defines the features of the particular type of system.

Our MCBR uses the generic notion of the CBR process and metadata of a given application domain. However, extracting the generic notion of CBR requires separating the characteristics of CBR that are common to all case-based reasoners from those that are specific to any particular problem domain. Two major common characteristics of CBR studied in our research are the case representation scheme and the mechanisms for assessing similarity among cases. Additional common characteristics might also be identified, e.g., the case-adaptation

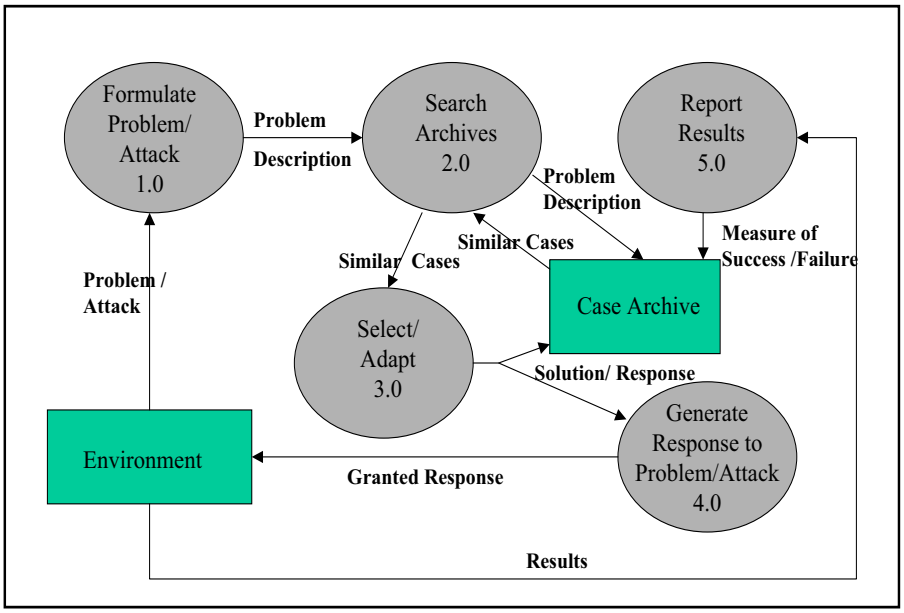

Figure 1: Case-based reasoning process

phase (Step 3.b), and it is planned to implement some of these in future work. Details of our present CBR framework and how it can be instantiated for a specific domain are depicted in Figure 2.

The instantiation of a specific domain CBR begins with defining the metadata for the domain, e.g., network intrusion detection. After the domain is defined, the domain-specific case representations are described in an XML schema. This is accomplished by analyzing the given problem domain, determining the representations needed for a "case," and encoding this in the XML descriptions. Cases in our system may have features of any user-defined type (e.g., IP address, eye color, license plate number, etc.). These steps provide the domain metadata shown on the left side of Figure 2.

The XML schema is used in conjunction with the Sun Microsystems Java XML Binding (JAXB) package to generate a binding schema, which in turn is used to generate the Java classes necessary to parse XML documents that conform to the schema. The Generic CBR Source code is the recompiled so as to inherit these cases. This produces the core of a case-based reasoner capable of receiving an input problem described in XML and searching the case archive for those cases that are most similar to it according to a given similarity metric.

At this point, however, the resulting system still doesn't know how to assess the similarity of cases in the archive with a given problem situation. This is accomplished by applying comparator methods to the individual case/problem features. As mentioned, the determination of which comparator will be used for which case feature is encoded in a metadata dictionary. When building a CBR system from scratch, all the needed comparator methods must be written. In subsequent systems, however, many such comparators may be reused. These components are depicted in the lower right part of Fig. 2. Once the comparators have been written or selected, and the corresponding entries have been made in the metadata dictionary, the instantiation of the framework is complete. One now has a fully functioning CBR system.

During case-retrieval, the correct comparator method needed for each case feature is determined dynamically by a look up in the metadata dictionary. Each entry in this dictionary contains the name and type of a case feature, the name of the comparator to be used for that feature, and other information relevant to that feature, such as the

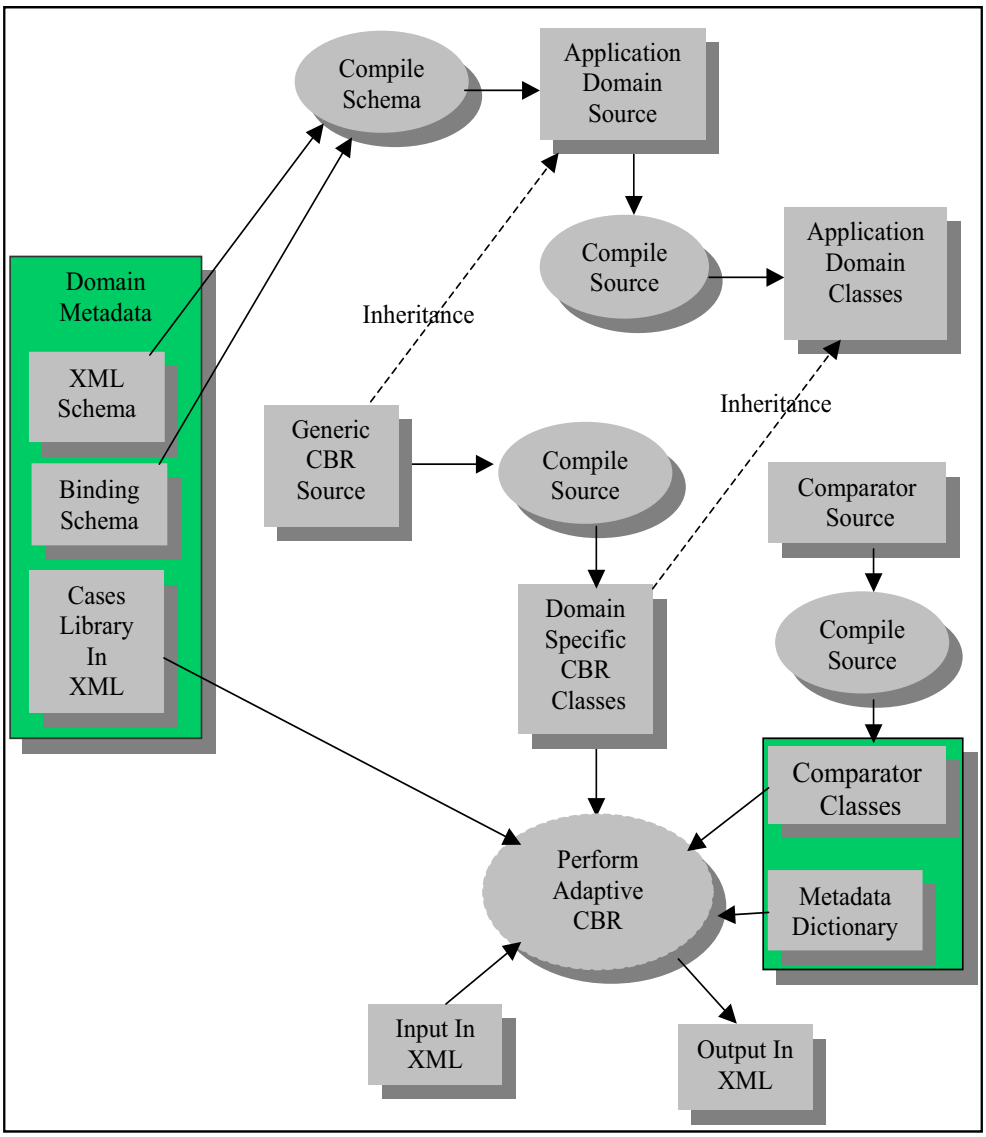

Figure 2: Metadata case-based reasoner

name of any method used to generate the associated part of the user interface. When a particular case in the archive is being assessed regarding its similarity with the given problem situation, each case feature is compared with the corresponding problem feature. This is accomplished by looking up the feature in the metadata dictionary, extracting the name of the associated comparator, and then dynamically creating an instance of that comparator by reflection on the comparator's class definition. Any domain specific comparator data is encoded in the metadata. Last the comparator is applied to the case-feature problem-feature pair, returning a measure of the degree of match. This use of a meta-data dictionary is a well-known practice of metadata software programming. After all the necessary comparators have been applied for a particular case, their results are combined using combination algorithms specified in the domain metadata.

5 Conclusion

This reasoner has been implemented as a proof of concept to create a case-based implementation of the well-known "Snort" network intrusion detection system [8]. Snort is rule-based, where each rule has features describing characteristics of network packets (protocol type, source and destination IP addresses and ports, payload contents, etc.) together with a prescribed action (e.g., raising an alert). The rule features are easily described in XML, and the Snort rule set (currently around 1300 rules) is easily converted into an XML archive (each Snort rule becoming its own case). The comparators look for such things as exact string matches on protocol names, and whether a destination IP address or port falls within a certain range. Our system effectively replicates the functionality of Snort. Additionally, we have also applied this approach to provide an intrusion detection system for ad hoc wireless networks [3]. Other domains, including facial recognition and network fault management, have been specified and are currently under implementation.

References

[1] Foote, B., and J. W. Yoder. "Metadata and Active Object-Models." Technical Report, WUSC-98-25, Department of Computer Science, Washington University, 1998.

[2] Fowler, M. Analysis Patterns: Reusable Object Models. Addison Wesley, 1997.

[3] Guha, R., O. Kachirski, D. G. Schwartz, S. Stoecklin, and E. Yilmaz. "Case-based agents for packet-level intrusion detection in ad hoc networks." Seventeenth International Symposium On Computer and Information Sciences, ISCIS'02, Orlando, FL, October 28-30, 2002, pp. 315-320.

[4] Hayes, C. and P. Cunningham. "Shaping a CBR view with XML." Case-Based Reasoning Research and Development, Proceeding of the Third International Conference on Case-Based Reasoning, ICCBR-99, Lecture Notes in Computer Science, LNAI v 1650, Springer Verlag, 1999.

[5] Lieberherr, K. J. Adaptive Object-Oriented Software: The Demeter Method withPropagation Patterns. PWS Publishing Company, 1996.

[6] Schwartz, D. G., S. Stoecklin, and E. Yilmaz. "A case-based approach to network intrusion detection." Proceedings of the Fifth International Conference on Information Fusion, IF'02, Annapolis, MD, 2002, pp. 1084-1089.

[7] Shimazu, H. "A textual cased-based reasoning system using XML on the world-wide web." Advances in Case-Based Reasoning, Proceedings of 4th European Workshop, EWCBR-98, Lecture Notes in Computer Science, LNAI v 1488, Springer Verlag, 1998, pp. 274-285.

[8] Snort, The Open Source Network Intrusion Detection System, available at http://www.snort.org

[9] Watson, I. "CBR is a methodology not a technology." The Knowledge Based Systems Journal, v. 12, no.5-6 (1999) 303-8.

[10] Yoder, J., Balaguer , F., and Johnson, R. "Architecture and Design of Adaptive Object Models" Intriguing Technology Presentation at the 2001 Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '01), ACM SIGPLAN Notices, ACM Press, December 2001.