Toward a Generic Case-Based Reasoning Framework Using Adaptive Software Architectures

Erbil Yilmaz, Sara Stoecklin, and Daniel G. Schwartz
Department of Computer Science
Florida State University
Tallahassee, FL, U.S.A.

Abstract Nearly all currently existing case-based reasoners have been tailor made for some specific application domain. In this paper we report progress on abstracting the key elements of an arbitrary case-based reasoner and using these to develop a generic software framework that can be instantiated to create a case-based reasoner for any given application domain. This employs what is known as a “reflective,” or “adaptive” software architecture for defining the similarity metric used for case-retrieval. A metadata dictionary that associates case features with their “comparators” is used in conjunction with an XML data type definition for describing the cases in the case archive. The overall framework makes it easy to redefine the nature of a case, and to add any new feature comparators that may be required, with very minimal extra coding. This is still a work in progress, but the core functionality has been implemented.

Keywords: Case-based reasoning, Adaptive software architectures, Reflective software architectures, Adaptive frameworks

1 Introduction

The generic properties of the case-based reasoning (CBR) cycle have been studied by others with the aim of designing case-representation languages that can be used across different problem domains. Although these languages have been successful in dealing with the case-representation problem, there remain further properties of the CBR process that can be considered as common to all CBR applications. Our work has aimed to identify these properties and implement them in a generic framework that can then be instantiated to create a fully functioning case-based reasoner for any given problem domain. Our framework makes use of two recent developments in software engineering practice: (i) what are known as “reflective,” or “adaptive,” software architectures [1, 5], and (ii) the Java XML Binding (JAXB) facility created by Sun Microsystems [3].

2 Case-Based Reasoning

We assume that the reader is familiar with theasic CBR process. A description may be found in [4]. As discussed there, the CBR methodology can be distinguished from the technology used to implement it. In effect, a CBR system can utilize many different techniques in different phases of its process. As examples: (i) in retrieving cases from the case archive, the similarity metric used in the search process may benefit from statistical comparisons, fuzzy logic techniques, or pattern matching with neural nets, and (ii) when the task of adapting retrieved cases to the current problem situation might employ a rule-based expert system. This distinction between methodology and technology becomes important in the design of a CBR framework using adaptive software architectures. By means of this, the CBR method can be generalized to a level of abstraction that allows us to employ adaptive software architecture techniques. This generalization leads to a
flexibility not found in previous CBR systems.

3 Adaptive CBR Framework

A key idea is to use a metadata dictionary for run-time method selection. 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 case retrieval step. An adaptive program can be described as a partially defined, and not thoroughly implemented, class structure that sets some constraints over actual, full-functioned implementations [1]. Such a program is thus a framework that 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.

Our CBR framework is constrained by the generic notion of the CBR process and the desire to be able to adapt our system to any 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.

The instantiation (or customization) of the framework begins with defining a domain-specific case representation in an XML data type definition (DTD) that satisfies the case-representation constraints required by the framework. This is accomplished by analyzing the given problem domain, determining exactly what should be meant by a “case,” and encoding this in the DTD.

Given this DTD, one next creates an associated JAXB schema. To this one applies the JAXB schema compiler, which automatically creates the Java classes necessary to process XML documents satisfying the DTD.

At this point, 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. 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.

This framework has been used to create a case-based implementation of the well-known “Snort” network intrusion detection system [2].

References


