DAREonline: A Web-Based Support Tool For Domain Engineering

Raimundo F. Dos Santos and William B. Frakes
Virginia Polytechnic Institute and State University
Computer Science Department
7054 Haycock Rd, Falls Church, VA 22043 USA
rdossant,wfrakes@vt.edu

Abstract: DAREonline is a web-based tool for domain engineering. It supports the DARE framework in a centralized platform-independent environment. Our approach leverages concepts of Service-Oriented Architecture (SOA) to aggregate data and functionality from diverse sources that can be helpful in domain engineering. In this demo, we describe DAREonline’s architecture and implementation.

1. Introduction

DARE – Domain Analysis and Reuse Environment – is a comprehensive framework that assists analysts of different levels in understanding the reuse potential of software systems. For this purpose, DARE provides facilities where domain information can be collected from technical files, architectural blueprints, and source code, among other documents. Along with that, human experts complete an essential list of resources targeted by DARE. DAREonline avoids some of the disadvantages of previous versions, such as being tied to a specific platform or development language. In fact, as web technologies allow, it can support a wide array of software solutions from competing vendors, mashup functions from independent contributors, server and client-side processing, and service-oriented architectures for access to external systems that are SOA compliant. One of the goals of DAREonline is to leverage open-source tools that can accomplish tasks around domain analysis. Of particular interest is the utilization of web services and its reuse potential. Moreover, DAREonline serves as a repository where analysts can store information during a domain analysis project. In this demo, we describe the architecture of DAREonline and provide some details of the underlying implementation.

2. Design and Implementation

2.1. Object Model

DAREonline employs common concepts of domain engineering as a foundation for its components. It uses the analogy of a domain book where each chapter of the book is filled in as the analysis process evolves. The parent component is a domain analysis object, which serves as no more than a reference around which other components can aggregate. A domain analysis allows many children and grandchildren components depending on how extensively the analysis process is done. Thus, if the analysis is about stemming algorithms, it may include several systems addressing different stemmers, such as Porter, Paice, and Lovins. Figure 1 provides a snapshot of the DAREonline object model. The arrows denote a “has” relationship. For instance, a domain analysis has a scope, which has a statement in addition to one or more equations.

A domain analysis uses the scope module (Figure 3) to establish coverage of what can and cannot be done in a particular instance of the domain engineering process [7]. The statement is a textual definition that briefly explains the purpose and limits of the software systems in question, and what it attempts to accomplish. Equations are derived as a means of relating variables, functions, and sets to establish a logical representation of the domain in question. For example, the goal of Information Retrieval (IR) is to search for and find documents according to some search criteria[2]. Given the set of all documents D in a corpus, the IR system uses a translation function M to find D’ documents according to query Q. The equivalent equation can be represented as  \[ D' = M(D,Q) \]

\(^1\)Our implementation is available at http://208.29.54.207:8080/dareonline.
DAREonline allows the analyst to store the variables, functions, and sets above in order to establish equations for a particular domain bounded by its scope. There is no limit to the number of equations that can be represented. In addition, more can be added or deleted as necessary, or edited on the go.

Systems represent the next component of the domain book implemented by DAREonline. Apart from scope, systems should be defined early in the process, possibly before other analysis tasks go too far. Most of the other components (e.g., documents and experts) must have a direct relation to at least one of the systems. DAREonline does not restrict the analyst in the number of systems that must be included. In domain engineering, it is generally accepted that at least three exemplar systems be specified for the domain analysis to proceed. At the object level, a system is simply composed of a name, a free-text description, and an identifying number. Other of its aspects are given deeper in the object hierarchy by the other components, such as document sources.

DAREonline has two form-based components. The first one, system characteristics (Figure 4), describes aspects such as architectural style, QA methods, and type of users they target, and is given by Frakes [1]. Forms are collected on a one-per-system basis, allowing the analyst to quickly toggle among different ones for a quick snapshot of what each system entails. The second form-based component is related to experts (Figure 5). System experts represent an essential source of information in the domain engineering field, and therefore must be identified according to their qualifications and relevance to the system in question. For each named professional, the expert object tracks a name, position, familiarity, and years of experience with the system. System and experts are directly linked to one another. The final component of the DAREonline object hierarchy is the document. Throughout the domain analysis process, analysts are bound to find a wide variety of files describing various systems, such as feature tables, activity logs, and source code. Along with knowledge conveyed by system experts, documentation presents in detail the inner workings of software systems. For each system, DAREonline allows the analyst to select among predefined types of documents and create a compilation of the sources of information that have been found during the domain analysis process.

2.2 Tools

DAREonline provides various tools for vocabulary analysis, feature tables, facet tables, and diagramming. A brief description of these tools is available at
During the design phase, our vision was to utilize as much open source functionality as possible that could be implemented locally or leveraged remotely via a service-oriented approach. The data repository uses MySQL 5.5, and stores procedures. The front-end interface is partially pictured in Figure 2. It suggests a top-down flow that can be followed during a domain analysis project: bound the domain’s scope, document the systems in question, collect information from sources and experts, and perform data analysis with the various tools. This flow is not programatically enforced by DAREonline, as it may be restrictive in many situations. The tools currently available in DAREonline can be categorized in three groups: javascript functions executed at the user’s browser; java modules running on the server; and web service calls. Each of these approaches provides a range of benefits and challenges. Javascript, for example, tends to be lightweight. It, however, has security limitations and depends on the browser’s settings for proper functioning. Java modules enjoy the full scope of the language’s capabilities, but require server resources. Web services, which have become popular in the past few years, allow efficient data integration, but can be impacted by third-party accuracy and reliability. Within DAREonline, the stemmer, the lexical analyzer, and some of the domain analysis components (e.g., systems and experts), are implemented in javascript. Java modules comprise most of the system’s business logic (e.g., definition of components and enforcement of relationships), while the synonyms queries are performed over a web service. This web service is our own implementation based on WordNet [6], and done as a means to investigate some of its reuse benefits.

In terms of implementation challenges, several items can be noted. First, DAREonline relies on third-party software which cannot be guaranteed for their accuracy. In fact, we have found words that the stemmer seemed to not handle accurately. It raises the question of reusing external systems that bear no accountability on the consuming system. Another issue is related to web services, which come in different flavors. While some use traditional Simple Object Access Protocol (SOAP) with Web Service Definition Language (WSDL) documents, others provide custom APIs. The various implementation types impose design, development, and maintenance overhead. In our experiments, establishing web service calls proved somewhat cumbersome as not all services were available at all times. As for ease of use, javascript functions as well as Java modules were less challenging possibly because the code was readily available to be scrutinized. A third issue arises when different browsers are used. DAREonline uses ActiveX Components to access the users’ file system (needed to save facet tables). ActiveX is supported on Internet Explorer, but not on other browsers, such as Firefox. Compatibility among browsers is a hindrance to systems integration. Facet tables use Java applets which may not always load properly due to differences in Java Runtime Environments (JRE). Online software collaboration has attractive qualities, such as wealth of variety and functionality. However, it leaves open many questions related to quality, security, reliability, and usability. These questions represent areas of research to improve software reuse.

In terms of implementation challenges, several items can be noted. First, DAREonline relies on third-party software which cannot be guaranteed for their accuracy. In fact, we have found words that the stemmer seemed to not handle accurately. It raises the question of reusing external systems that bear no accountability on the consuming system. Another issue is related to web services, which come in different flavors. While some use traditional Simple Object Access Protocol (SOAP) with Web Service Definition Language (WSDL) documents, others provide custom APIs. The various implementation types impose design, development, and maintenance overhead. In our experiments, establishing web service calls proved somewhat cumbersome as not all services were available at all times. As for ease of use, javascript functions as well as Java modules were less challenging possibly because the code was readily available to be scrutinized. A third issue arises when different browsers are used. DAREonline uses ActiveX Components to access the users’ file system (needed to save facet tables). ActiveX is supported on Internet Explorer, but not on other browsers, such as Firefox. Compatibility among browsers is a hindrance to systems integration. Facet tables use Java applets which may not always load properly due to differences in Java Runtime Environments (JRE). Online software collaboration has attractive qualities, such as wealth of variety and functionality. However, it leaves open many questions related to quality, security, reliability, and usability. These questions represent areas of research to improve software reuse.

3. Conclusions

Over time, the DARE framework has been implemented on different operating systems (e.g., Windows and Linux) and supported by several programming languages (e.g., C and VB). We now present DAREonline, a web-based application that assists users implementing domain engineering tasks under the DARE framework. We target SOA components for their strong reuse potential across distributed environments. Our goal is to provide a repository of functions that is helpful in domain analysis while maintaining ease of use throughout the application. DAREonline is designed with open-source tools, some of which are developed and maintained in-house, and others provided by third parties. We envision many enhancements that can be introduced in order to make DAREonline more robust. For example, we plan to investigate how workflow languages (e.g., EBPL, WSFL) can be leveraged to identify and schedule the execution of reusable services as part of our SOA research. We must also understand how to assure software quality given that web services are often disconnected components over which service consumers have little control. In addition, we plan to introduce upload/download capabilities for documents as well as a search tool. We continuously examine tools that can
be added to the repository, such as software for code analysis. Our initial results motivate us to continue pursuing intelligent options to support the DARE framework.

4. References