Meta-Model Search:
Using XPath to Search Domain-Specific Models

Rajesh Sudarsan
Virginia Tech
Department of Computer Science
Blacksburg VA24060
540-922-2313
sudarsar (at) vt.edu

Jeff Gray
University of Alabama at Birmingham
Department of Computer Science
Birmingham AL 35205
205-934-8643
gray (at) cis.uab.edu

Abstract

A common task that is often needed in many software development tools is the capability to search the artifact that is being created in a flexible and efficient manner. However, this capability is typically absent in meta-programmable modeling tools, which can be a serious disadvantage as the size of a model increases. As a remedy, we introduce a method to search domain models using XPath – a World Wide Web Consortium (W3C) standard that uses logical predicates to search an XML document. In this paper, an XPath search engine is described that traverses the internal representation of a modeling tool (rather than an XML document) and returns those model entities that match the XPath predicate expression.

Keywords: XPath, domain-specific models, GME, model search

1. Introduction

The theory of Information Retrieval (IR) [7] is focused on the science of data collection and management, in association with mechanisms that provide rapid access to the collected information. In comparison to data retrieval techniques, IR generally searches for items that partially match the given requirements and then selects the best item out of the selection. In many domains, the vastness of available information makes searching a difficult task because of the noise generated from false positive matches. In such cases, it is important to retain and organize the search results in such a way that rapid retrieval is possible. Thus, an efficient search tool often leads to a substantial increase in productivity regarding the retrieval of the correct information.

Search tools are a common feature in most application programs. For example, in the domain of text-based searching (e.g., find/replace utility in Microsoft Word) or web-based searching (e.g., Google keyword-based search engine), the advanced search option provides the user with the capability to add flexibility to the search query. Searching also plays an important role in software development environments. Integrated development environments, such as Eclipse and Visual Studio, provide basic mechanisms for searching through source code in order to identify places in the code that have specific properties. The ability to search is particularly needed in software modeling tools, which may contain multiple levels of hierarchy among many connected entities.

A generic modeling tool can be used to specify the characteristics of a software system at a high-level of abstraction. The properties of the system are preserved in the models and the interaction between different entities of the system is denoted using the models in the domain. As users modify the system to meet the changing requirements, the domain model also has to evolve to accommodate the corresponding changes. Thus, an efficient search tool is required that is capable of searching for models with specific properties. Unlike programming environments, most modeling tools do not support advanced search capabilities, such as regular expressions or wild cards for locating a particular kind of modeling element. Absence of advanced search capabilities can be overlooked in modeling tools if the search space is small. However, advanced automated support becomes

necessary when the size of a model increases. In modeling tools, when the number of modeling elements grows exponentially (e.g., some models contains thousands of entities), then an efficient, scalable search technique is required to match the attributes to locate the model elements.

This paper discusses the development of a scalable solution for adding improved flexibility to the search feature of modeling tools. The solution provides a method to search efficiently for a modeled artifact using XPath [9]. The search predicate is specified using a query string written in XPath, and the output of the search is a list of all model entities that match the given predicate. XPath is typically applied to XML documents, but our search tool uses the XPath language as a notation for specifying a search query within a domain-specific modeling tool.

The paper is organized as follows. Section 2 presents a brief overview of XPath and domain-specific modeling in order to set the context for the paper. Section 3 gives a brief description of the problem to be solved. The core of the paper is found in Section 4, which introduces the XMOS search plug-in. A case study is presented in Section 5. The contribution and future work are summarized in Section 6.

2. Background

The two main concepts discussed in this paper are XPath [9] and domain-specific modeling [3][4][10]. This section provides a brief background of these two topics.

2.1 XML Path Language (XPath)

There are numerous tools and techniques that are based on the principle of searching using regular expressions, such as JQuery [1], XQuery, and XPath [9]. JQuery is a flexible Java source code browser, but XQuery and XPath are query languages that are typically applied to XML files. The difference between XQuery and XPath lies in the fact that XQuery uses the structure of XML intelligently to express queries across all types of data that are represented in an XML document. XPath is a language for selecting parts of an XML document and is used in both extensible stylesheet language transformations (XSLT) [12] and XPointer [9].

The XPath query language is an industry standard for representing search expressions in hierarchical-based systems. XPath exploits the input tree structure by traversing either from the root node or any other context node, and selects nodes depending upon the return value of the matching predicate. The selection of nodes in an XML document is based on various properties (e.g., element type, attribute value, character content, and relative position). The basic expressions in XPath include string, numerals, Boolean, location paths, function calls, variable reference; the complex expressions include unions, filters, and relational expressions. The evaluation of an XPath expression produces an object, or a collection of objects, that exist in the associated XML document. These objects can be a node-set, Boolean, number, and string. Examples of XPath search expressions are provide in Section 5.2.

2.2 Domain-specific Modeling and GME

The term *domain* generally denotes a group of entities that share the same characteristic or exhibit similar functionality. From a software engineering perspective, domain refers to a class of existing systems in a product-line architecture (PLA) [6]. All the products developed using this architecture will share common properties of the product family in addition to the specific properties that makes the product unique.

Models are an abstract representation of real-world systems or processes. Model Integrated Computing (MIC) [10] employs domain-specific models to represent the software, its environment, and their relationship. With MIC, a modeling environment operates according to a meta-model, which contains a description of the entities, attributes, and relationships that are relevant in the domain [3]. With a domain-specific modeling environment, the intention of an end-user can be captured using idioms and icons that are familiar to the domain expert. This permits the description of the problem at a higher-level of abstraction from which other software artifacts (e.g., source code and simulation scripts) can be generated.

The Generic Modeling Environment (GME) [5] is a meta-programmable tool that is based on the principles of MIC. The GME provides a universal design environment that can be configured over a wide range of domains. GME

supports various concepts, such as hierarchy, multiple aspects, set, references and explicit constraints, for building complex domain-specific models [4]. The GME can be configured from a meta-model to work with any domain. In Section 4, the case study model of Figure 4 shows a GME model that represents a real-time avionics application.

The GME allows third-party tools to be attached as plug-ins and invoked from within a modeling environment. The focus of the following section is a GME plug-in that permits search queries to be expressed in XPath.

3. Problem Description

Searching in a modeling tool involves comparisons between the user-provided search string and the property values of every modeling entity. The search feature currently available in GME (shown in Figure 1) is capable of searching through the models in a simplistic manner by restricting the user to selecting one or more fixed checkboxes of element types. The number of attributes that can be matched is limited to those that correspond to Roles, Kinds, and Attributes; additional criteria beyond this fixed set are not permitted. Furthermore, the current search utility does not support composition of two or more search conditions to be expressed in a query. This leads to a serious disadvantage when the search domain contains thousands of entities. Another limitation in the original search utility is the lack of case-sensitive searching.
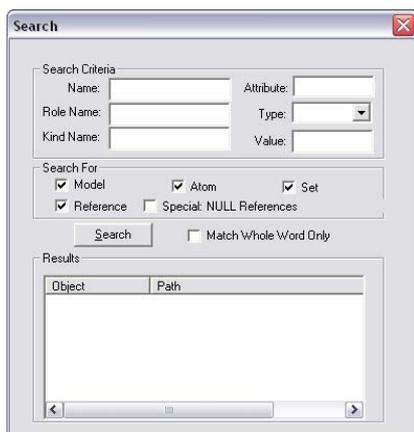

Figure 1. Existing search utility in GME

In the search dialog shown in Figure 1, the user cannot specify regular expressions, wildcard characters, or inequalities (such as < or >) as part of the search string. This reduces the flexibility in the search capability. To improve the efficiency of searching, the user may desire the ability to synthesize a new query by composing smaller sub-queries. This paper discusses the development of a new search plug-in for GME that uses an XPath predicate to specify the search criteria. XPath was chosen to be the input language for specifying the search predicates because it matches the hierarchical tree structure present in GME. It also provides the flexibility to compose new search queries by joining two or more sub-queries. A key challenge was to force XPath, which was designed to work with XML documents, to understand the internal representation of a model stored in the GME.

4. XPath Model Search (XMOS)

The XPath Model Search (XMOS) plug-in uses XPath predicates to define a search query on a domain-specific model. The plug-in can be used to locate the desired objects in a model that match the search criteria. The XPath query is specified from the entity names present in the domain's meta-model. The query predicates, when parsed using an XPath parser, output tokens that can be used for searching through a model. The main modules of XMOS are:

1. XPath Evaluator
   a. XPath Expression
   b. XPath Parser Interpreter module
2. Builder Object Network (BON) API
3. Search Module
4. User Interface

The architecture of the XMOS plug-in is shown in Figure 2. The XPath Evaluator parses the input search predicate (an XPath expression) and provides a parse tree to the Interpreter module. The Interpreter module organizes the tree into searchable tokens and stores them in a customized data structure. The XMOS plug-in uses these tokens to search through the GME Builder Object list, which provides an interface to the internal GME representation of a model. The final result produced by XMOS is a list of objects that match the search criteria. The user interface module displays the results of the search. Each of these modules is described briefly in this section.
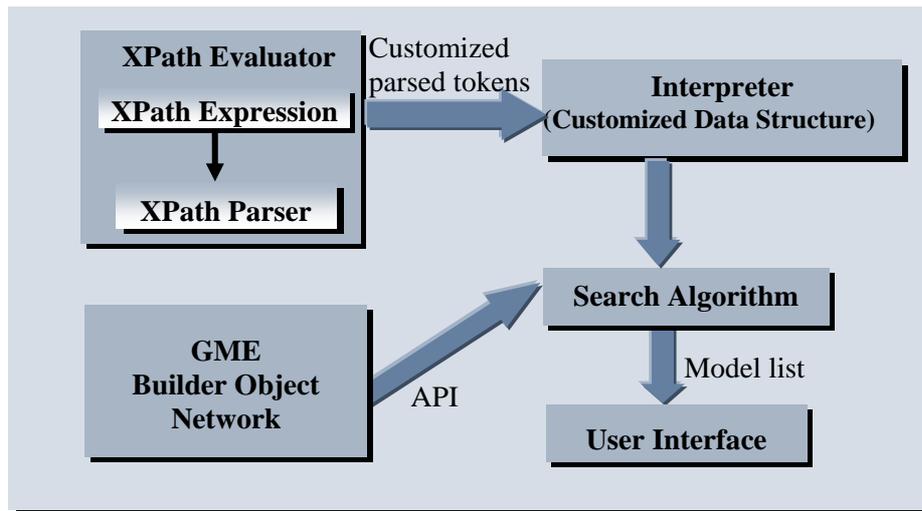
Figure 2. Architecture of the XMOS plug-in

4.1 XPath Evaluator

The XPath Evaluator is divided into two main sub-modules: a module for receiving the search string from the user, and a module for evaluating the input expression for its syntactic correctness and subsequent generation of the appropriate parse tree. The input dialog box reads the search expression and passes it to the XPath parser. The XMOS XPath parser is based on the TinyXPath [11] parser, which is an open source implementation. The parser supports most of the XPath expressions and conforms to the W3C XPath specification.

4.2 XPath Interpreter

The interpreter module uses tokens from the parser for interpretation. From the token list, every token is interpreted with respect to the previous token. For example, if the first token in the XPath expression is a slash (/), then the next expected node can be slash (/), star (*), or a node name. The first slash indicates that the searching starts from the root node. Thus, when slash (/) is encountered as a second consecutive token, the tokens are combined to interpret it as a double-slash and the searching technique is set to recursive descent. If a star (*) is encountered as the second token, then the expression is interpreted to search for any kind of GME object. The node name is used to search for an object with the name that is a child of the root node.

4.3 GME BON

To facilitate the automatic processing of model data, GME supports access to the underlying model information through the Builder Object Network (BON) API, which provides a powerful but easy to use interface for writing application programs that can be attached to GME as a plug-in. BON provides a network of C++ objects, where each of these objects represents an entity in the GME model database. The C++ methods provide access to read/write object properties, attributes, and relations.

4.4 Search Module

The algorithm used for searching the objects in a GME model involves the use of BON objects in association with the XPath interpreter described in Section 4.2. Whenever a token is received, the process of searching is performed in a sequential manner. Depending upon the interpretation, corresponding BON objects are invoked and a search is performed for the particular token. For example, if the search token is a model name, then a BON CBuilderModelList object is queried to compare all the models returned by the reflective method GetModels(). A search is performed on this list to match the model name given in the search predicate. For the next token, the matched list of CBuilderObjects is searched for a match for the next token. By proceeding in a similar manner, the search is invoked for all the tokens given in the search predicate. For the first token, the input list encompasses all the CBuilderObjects in the project. The result of the search is the list of all CBuilderObjects returned after all the tokens have been processed.

As an example of the search process, if the search string is "//*" (which would return all elements in the model), the first token to be processed is slash (/). Interpretation of this token sets the starting point of the search as the root of the GME model tree. When the next token is also a slash, then the previous token and the current token are combined together and interpreted as double-slash (//). Interpretation of this token will set the search strategy as recursive descent. The next token star (*) acts as a wildcard character. This token, when interpreted with respect to the previous token (double-slash), results in the recursive search for all CBuilder objects present in the project starting from the root to the leaf.

4.5 User Interface
The XMOS user interface is a dialog box that consists of an edit box for entering the XPath expression. It also has a list control box that is used to display the search results as a multi-tab output. The different tabs in the output window are the Object name, location path, location coordinates, and the type of the object (atom, model, reference, set, or connection). The path tab displays the path of the component from its level in the hierarchy to the topmost model. The Location tab gives the exact coordinates of the components within the window in which it is located.

4.6 Algorithm
The overall model search algorithm used in the plug-in is shown in Figure 3.

1. Read the input search string. Tokenize the string and store the tokens in a vector list.
2. While the token list is not completely traversed, do the following
   a. Get the next token from the list
   b. Set the value of CURR_TOKEN as the current token.
   c. Interpret this token with respect to the previous token in PREV_TOKEN
   d. Query the GME Builder Object Network for the objects corresponding to the interpreted token.
   e. Store the list of matched GME objects, their location, and the type of object.
   f. Set PREV_TOKEN=CURR_TOKEN
3. Output the final list of matched GME objects to the user

Figure 3. XPath model search algorithm

5. Case Study
As a case study, the XMOS plug-in is applied to models developed in the Embedded Systems Modeling Language (ESML). The ESML was primarily designed by the Vanderbilt DARPA MoBIES team, and can be downloaded at http://www.isis.vanderbilt.edu/Projects/mobies. Section 5.1 describes the models developed in ESML for denoting scenarios in the Boeing Bold Stroke Mission Computing Avionics Framework [2]. Two example search expressions on an ESML model are presented in Section 5.2

5.1 Boeing Bold Stroke and the ESML
Bold Stroke is a product-line architecture written in C++ that was developed by Boeing in 1995 to support families of mission computing avionics applications for a variety of military aircraft [8]. Mission computing software such as Bold Stroke is responsible for controlling navigational sensors, weapon deployment subsystems, and cockpit panel displays that are used by a fighter pilot. Bold Stroke is a multithreaded, real-time system that has hard and soft deadlines. It is a very complex framework with several thousand components implemented in over a million lines of code.

The ESML is a domain-specific graphical modeling language in GME for specifying realtime mission computing embedded avionics applications, such as Bold Stroke. The goal of ESML is to address the issues arising in system integration, validation, verification, and testing of embedded systems. An example ESML model of the internal properties of a CORBA real-time event channel is shown in Figure 4. It illustrates the components and interactions for a specific scenario within Bold Stroke. This scenario provides insight into a small subset of aspects that are related to weapon system avionics software development, and at the same time addresses issues of component distribution. There are representative ESML models for all of the Bold Stroke usage scenarios that have been defined by Boeing.
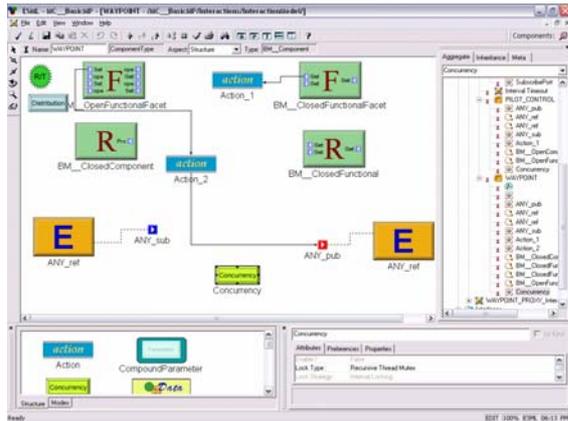
Figure 4. Bold Stroke component interaction in ESML

From an ESML model, deployment and assembly files are generated that specify the usage scenarios for a specific product-line configuration of Bold Stroke. An ESML model may contain several hundred modeling elements that need to be configured to explore different design alternatives. Searching through a large number of hierarchical models (in order to change configuration properties) can be very time consuming without the aid of a flexible search feature.

5.2 Example search expressions
The XMOS plug-in was applied to several different test cases using the Bold Stroke ESML models. This sub-section presents two examples of the search test cases and the results.

Case 1: *Lock Type* and *Lock Strategy* are attributes of the concurrency atom that specify the concurrency mechanism in a model. The following XPath expression finds all concurrency elements in the scenario that have the attribute *Lock Type* set to *Null Lock*, and *Lock strategy* not set to *Recursive Thread Mutex* (result is shown in Figure 5):

//concurrency[@locktype="Null Lock" and @lockstrategy != "Recursive Thread Mutex"]

Case 2: *Data* and *log* elements are used in an ESML model to store additional information. The following XPath expression finds all elements in which the *value* of the *data* element equals 5, and the *kind* attribute of the *log* element is set to *On Read* (result is shown in Figure 6):

//data[@value=5] and //log[@kind= "On Read"]

Figure 5. Result for case 1

Figure 6. Result for case 2

6. Conclusion
The lack of flexibility in the previous GME search utility necessitated the development of an efficient algorithm and plug-in to enable customized searching of domain-specific models. The XMOS plug-in exploits the hierarchical structure of GME to search for artifacts that match an XPath predicate expression. In very large models, there is often a need to search for a particular set of modeling entities to change their properties. XMOS provides the flexibility needed to locate these entities based on attribute values.

To the best of our knowledge, there has been no prior work that has addressed the problem of searching domain-specific models efficiently.

Regarding future work, the following tasks will be pursued:

- A feature that will be developed will have a direct link between the results and the actual location of the models in the domain, such that when a user clicks on the link on the output screen, a new window will be opened to show the exact location of the matched modeling elements.
- The XPath Evaluator will be extended to include more complex expression for searching.
- We believe that XMOS can be generalized for other modeling toolkits to add flexibility and efficiency to the searching technique.

Additional information about XMOS, including a plug-in download page, can be found at: http://www.cis.uab.edu/info/alumni/sudarsar/xmos/

7. References

[1] Andrew Eisenberg and Kris De Volder, "JQuery: Finding Your Way through Tangled Code," *AOSD 2004 Demonstration*, March 2004, Lancaster, UK.

[2] Jeff Gray, Jing Zhang, Yuehua Lin, Hui Wu, Suman Roychoudhury, Rajesh Sudarsan, Aniruddha Gokhale, Sandeep Neema, Feng Shi, and Ted Bapty, "Model-Driven Program Transformation of a Large Avionics Framework," *Generative Programming and Component Engineering (GPCE 2004)*, Springer-Verlag LNCS, Vancouver, BC, October 2004, pp. 361-378.

[3] Gábor Karsai, Miklos Maroti, Ákos Lédeczi, Jeff Gray, and Janos Sztipanovits, "Composition and Cloning in Modeling and Meta-Modeling," *IEEE Transactions on Control System Technology* (special issue on Computer Automated Multi-Paradigm Modeling), March 2004, pp. 263-278.

[4] Ákos Lédeczi, Arpad Bakay, Miklos Maroti, Peter Volgyesi, Greg Nordstrom, Jonathan Sprinkle, and Gabor Karsai, "Composing Domain-Specific Design Environments," *Computer*, November 2001, pp. 44-51.

[5] Ákos Lédeczi, Miklós Maróti and Péter Völgyesi, "The Generic Modeling Environment," *Technical Report*, Institute for Software Integrated Systems, Vanderbilt University, 2004.

[6] David L. Parnas. "On the Design and Development of Program Families," *IEEE Transactions on Software Engineering*, March 1976, pp. 1-9.

[7] C. J. van Rijsbergen, *Information Retrieval*, Butterworths, London, 1979.

[8] David Sharp, "Component-Based Product Line Development of Avionics Software," *First Software Product Lines Conference (SPLC-1)*, Denver, Colorado, August 2000, pp. 353-369.

[9] John Simpson, *XPath and XPointer*, O'Reilly Publications, 2002.

[10] Janos Sztipanovits and Gábor Karsai, "Model-Integrated Computing," *IEEE Computer*, April 1997, pp. 10-12.

[11] TinyXPath : a tiny C++ XPath processor, http://tinyxpath.sourceforge.net/

[12] World Wide Web Consortium, "XSL Transformations (XSLT)." W3C Recommendation. http://www.w3.org/TR/xslt