Generating a Generator

Jeff Gray
Department of Computer and Information Sciences
University of Alabama at Birmingham
gray (at) cis.uab.edu http://www.gray-area.org

Abstract. This brief paper describes future extensions to a framework that supports the creation of aspect weavers for domain modeling. The current implementation of the weaver framework is tied to a specific modeling tool. The two framework extensions outlined here will provide the variability needed to use the weaver with other metamodeling tools. A key to this future work will be the generation of a generator from an XML DTD or schema.

1. Introduction

The Generic Modeling Environment (GME) is a metamodeling tool that has been under development for over a decade at Vanderbilt University's Institute for Software Integrated Systems (ISIS) [9]. The GME provides the ability to build domain-specific modeling environments from a metalevel specification of the domain [7]. Once a tailored GME environment has been created, models are constructed using idioms and visual representations that are peculiar to a particular domain. Domain-specific interpreters are also created that can synthesize the domain models into other artifacts (e.g., source code, VHDL, FPGA's).

It is often the case that the domain metamodel forces the emergence of a dominant decomposition [10] that imposes the subjugation of other concerns. It has been observed that some properties of a domain-specific model are crosscutting [4]. That is, a single concern (e.g., latency or power in a real-time system) is scattered throughout the model in many different locations. Modifications to a crosscutting concern require a modeler to visit *each* location that contains the concern and make manual modifications – a difficult, if not impossible, task for anything but a simple model.

To provide better modularization and separation for crosscutting modeling concerns, a combination of an aspect-oriented [6] and generative programming [2] approach was applied to domain modeling within the GME (please see [4, 5] for more details). The solution provides a metaweaver framework that supports the creation of new weavers for each domain (see Figure 1). To provide an instantiation of the framework, domain-specific strategies are written to specify heuristics for each particular domain. The Embedded Constraint Language (ECL) was developed to provide the capability to navigate a model and specify transformations used in strategies. As shown in Figure 1, a code generator translates the ECL specifications into C++ code that is then compiled for each weaver instance. Once a new weaver is created, modelers can then modularize concerns by writing specification aspects. A specification aspect supplies the capability needed to properly separate crosscutting concerns.

The intent of the framework is to provide variability for different domains. That is, it gives the benefit of being able to generate new weavers for each new domain. However, the two main components of the framework (StratGen and the XML Parser) are very much coupled to the GME tool. It would be interesting to explore the possibilities of applying weaving to models created from other modeling tools. The rest of this paper describes two additional degrees of variability that will be explored in order to make domain weaving available from within other tools.

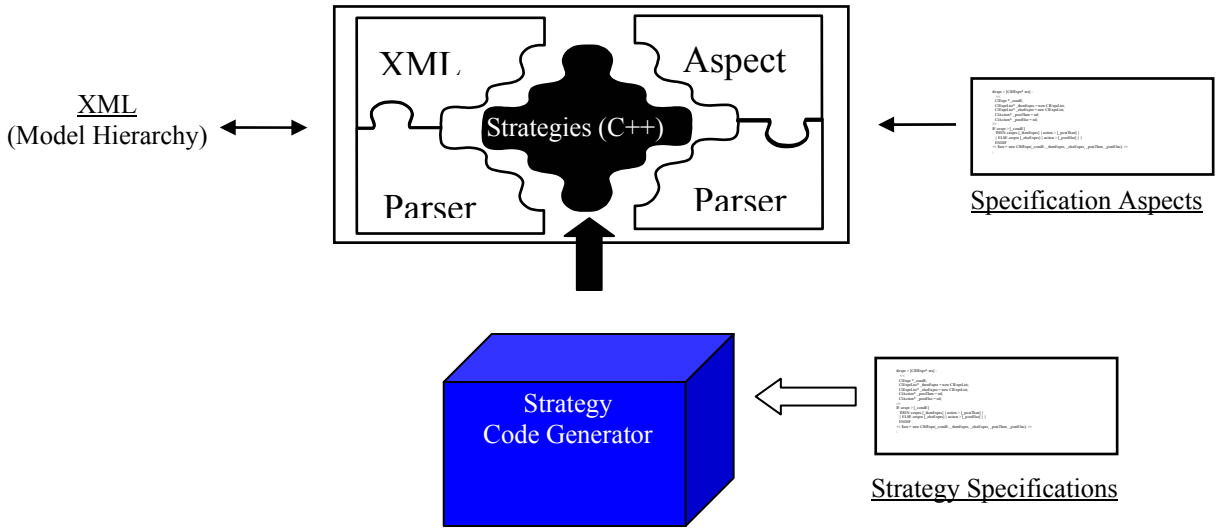

Figure 1: Metaweaver Framework

2. Variability with Respect to Modeling Tools

The current incarnation of the weaver assumes that the separation of modeling concerns is being performed on models created with the GME. In fact, this assumption is built into the XML Parser that was shown in Figure 1. The limitation imposed by this assumption precludes other modeling tools (that also can export models using XML) from being able to employ the benefits of an aspect model weaver. In addition to the GME, other examples of domain-specific visual modeling tools are Honeywell's Domain Modeling Environment [3], and metaEdit+ (from metaCASE) [11]. It is possible that these, and other modeling tools (such as Ptolemy, from UC Berkeley [8]) and notations (such as an XML representation for EXPRESS [1]), could benefit from an aspect-oriented modeling approach.

Figure 2 illustrates the manner in which a new code generator could be inserted into the metaweaver framework in order to provide an added measure of variability. From the modeling tool's Document Type Definition (DTD), the functionality of the DOM wrappers provided within the XML Parser can be generated. For example, the ECL provides several reflective operators that allow details of a model to be obtained from within a strategy or specification aspect (examples of such operators would be `findModel`, `findAtom`, and `findAttribute`). These reflective operators are actually implemented as wrappers within the XML Parser.

The details of the implementation of the XML Parser reveal that it is tightly coupled to the DTD that GME uses during the import and export of models. A subset of the GME DTD is shown in Figure 3. That figure specifies the definition of GME models, atoms, and attributes. The definition of other modeling entities (e.g., connections and references, among others) would be specified similarly. Other tools, where the DTD may not contain modeling elements called "model," "atom," or "attribute" would require different adapters for accessing the XML DOM.

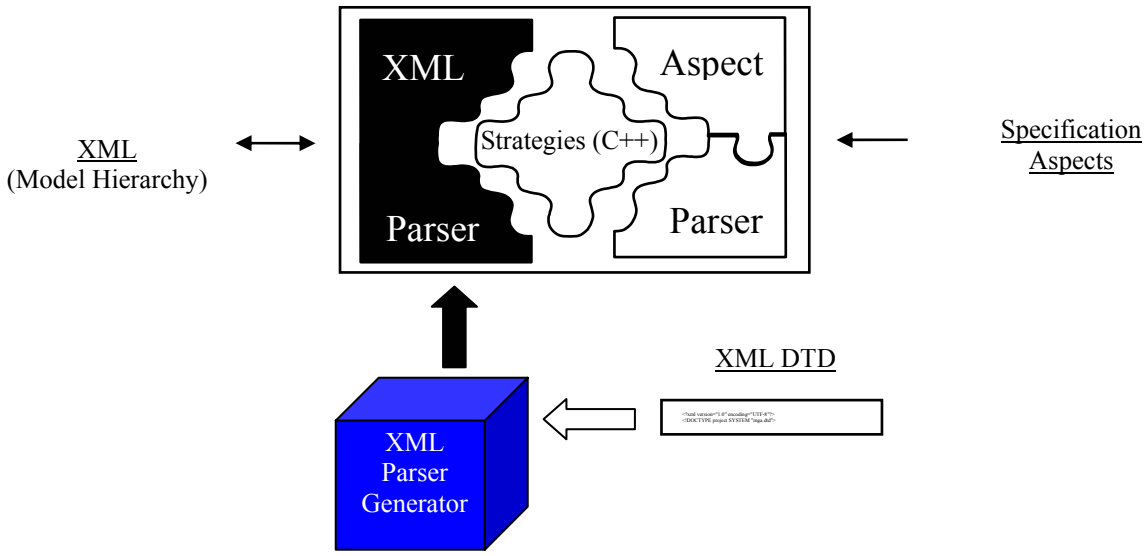

Figure 2: Variability with Respect to Modeling Tool

Given the XML element definitions from the Figure 3, there is a straightforward mapping to many of the XML Parser methods. A few such methods are listed in Figure 4. Throughout the code listing in that figure, it can be observed quite easily that the element definitions from the DTD have greatly influenced the methods defined in the XML Parser (to see this, just look through the source in Figure 4 for strings like "model," "atom," and "id"). Similar routines could be generated from the metalevel definition (found in the DTD) of other modeling tools that use XML for model persistence.

```
<!ELEMENT model (name, (constraint|attribute|model|atom|reference|set
                        connection)*)>
<!ATTLIST model
        id                      ID                      #IMPLIED
        kind                    NMTOKEN         #REQUIRED
        role                    NMTOKEN         #IMPLIED
>

<!ELEMENT atom (name, (regnode|constraint|attribute)*)>
<!ATTLIST atom
        id                      ID                      #IMPLIED
        kind                    NMTOKEN         #REQUIRED
        role                    NMTOKEN         #IMPLIED
>
<!ELEMENT attribute (value, regnode*)>
<!ATTLIST attribute
        kind                    NMTOKEN         #REQUIRED
>
```

Figure 3: Subset of GME DTD

In Figure 4, the `addAtom` method simply calls another XML Parser support method named `addNode`. This method makes the necessary call to the DOM in order to attach a new node to the XML model (a structurally equivalent `addModel` method is coded in the same manner – atoms and models, as shown in the DTD, have the same attribute list). The `findModel` method basically executes an XPath query to the DOM in order to search for a model with a specific name (`submitXPath` is itself an adapter method whose details are not shown here). The `findFolder` and `findAtom` methods are written in the same style. The "id" attribute of any modeling element can be obtained by calling the XML Parser `id` method. Other attribute accessor methods are written in the same style. Again, though, these methods are tied to the GME DTD. The future research will explore the generation of the code in Figure 4 from the tool's DTD.

```
nodeType XMLParser::addAtom(nodeType self, CComBSTR kind,
                            CComBSTR role, CComBSTR name)
{
       return addNode(self, "atom", kind, role, name);
}
nodeType XMLParser::findModel(nodeType aNode, CComBSTR name)
{

  CComBSTR bstrFind(L"./model[name=\"");
  nodeType res;

  bstrFind.Append(name);
  bstrFind.Append("\"]");

  res = submitXPath(aNode, bstrFind);

  return res;

}
CComBSTR XMLParser::id(nodeType aNode)
{
  CComBSTR res;
  CComPtr<IXMLDOMNode> attr = XMLParser::findAttribute(aNode, "id");
  XMLParser::getStr(attr, res);
  return res;
}
```

Figure 4: Sample Subset of XML Parser Methods

3. Generating a Code Generator

A perusal of the strategies specified in [4. 5] exposes the fact that the following operators are referenced in many strategy definitions: `connections`, `models`, `refs`, `connpoint`, `findFolder`, `findModel`, and `findAtom`. This suggests that tool-specific knowledge has crept into the intentions that can be expressed from within the ECL. A survey of the methods within the StratGen code generator will reveal the presence of GME-specific concepts (it is recognized that many tools would use terms such as "atom" and "model" to denote specific modeling concepts, but the presence of methods like `findModel` is the result of a dependence on the GME, not a generalization of all modeling tools). This can be viewed in Figure 5, which contains the code to generate the C++ strategy for calling the `findModel` method that is in the XML Parser (see the second method in Figure 4). The generation methods for `findAtom`, `findConnection`, and a host of other tool-specific methods are constructed in an analogous manner by making reference to the methods provided in XML Parser.

```
void Generator::GenerateFindModel()
{

  static findModelCounter = 0;

  genOut << indentStr << "nodeType aModelFind" << findModelCounter <<
                  " = XMLParser::findModel(" << lastVariable << ", ";

  lastVariable.Format("%s%d", "aModelFind", findModelCounter++);

}
```

Figure 5: Code Generation for `findModel` (located within StratGen)

To reduce the tool dependency bias within the StratGen code generator, portions of StratGen itself could be generated from a tool's DTD, as suggested in Figure 6. This is the second part of the future work that is planned in order to bring the aspect weaving idea to other tools.

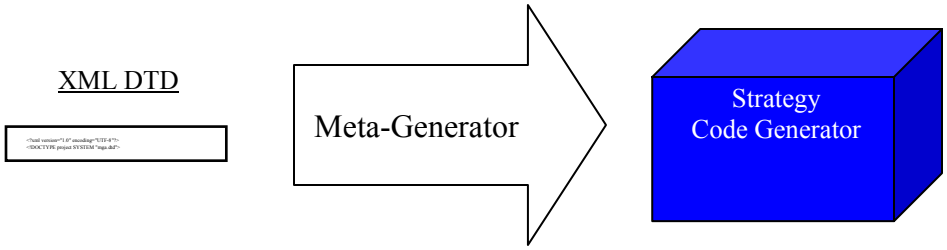

Figure 6: Generating StratGen from a Tool-Specific DTD

4. Conclusion

The initial effort for providing support for aspect-oriented domain modeling (as described in [4, 5]) was coupled to a particular modeling tool. It would be interesting to experiment with the possibilities of a more generalized framework. This brief position paper outlined some ideas for future work that will be explored in order to provide support for better separation of modeling concerns that are crosscutting in nature.

References

1. Edward Barkmeyer and Joshua Lubell, "XML Representation of EXPRESS Models and Data," *ICSE Workshop on XML Technologies and Software Engineering*, Toronto, Ontario, Canada, May 2001.
2. Krzysztof Czarnecki and Ulrich Eiseneker, *Generative Programming Methods, Tools, and Applications*, Addison Wesley, 2000.
3. http://www.htc.honeywell.com/dome/
4. Jeff Gray, Ted Bapty, Sandeep Neema, and James Tuck, "Handling Crosscutting Constraints in Domain-Specific Modeling," *Communications of the ACM*, October 2001, pp. 87-93.
5. Jeff Gray, "Aspect-Oriented Domain-Specific Modeling: A Generative Approach Using a Metaweaver Framework," Department of Electrical Engineering and Computer Science, Vanderbilt University, Doctoral Dissertation, March 2002.
6. Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin, "Aspect-Oriented Programming," *European Conference on Object-Oriented Programming (ECOOP)*, LNCS 1241, Springer-Verlag, Jyväskylä, Finland, June 1997, pp. 220-242.
7. Ákos Lédeczi, Arpad Bakay, Miklos Maroti, Peter Volgyesi, Greg Nordstrom, Jonathan Sprinkle, and Gábor Karsai, "Composing Domain-Specific Design Environments," *IEEE Computer*, November 2001, pp. 44-51.
8. Edward Lee, "Overview of the Ptolemy Project," Technical Memorandum UCB/ERL M01/11, March 6, 2001.
9. Janos Sztipanovits and Gábor Karsai, "Model-Integrated Computing," *IEEE Computer*, April 1997, pp. 10-12.
10. Peri Tarr, Harold Ossher, William Harrison, and Stanley Sutton, "N Degrees of Separation: Multi-Dimensional Separation of Concerns," *International Conference on Software Engineering (ICSE)*, Los Angeles, California, May 1999, pp. 107-119.
11. Juha-Pekka Tolvanen and Steve Kelly, "Visual Domain-Specific Modeling: Benefits and Experiences of Using metaCASE Tools," *ECOOP Workshop on Model Engineering,* Cannes, France, June 2000.