

An Example of Constraint Weaving in Domain-Specific Modeling¹

Jeff Gray, Ted Bapty, Sandeep Neema
Institute for Software Integrated Systems
Vanderbilt University - Nashville, TN
{jgray, bapty, neemask}@vuse.vanderbilt.edu

Introduction

There is a growing interest in the area of Advanced Separation of Concerns (ASOC). This is evident in the numerous workshops on this topic that have been offered recently at the past OOPSLA, ICSE, and ECOOP conferences. An example of the work in this area is Aspect-Oriented Programming (AOP)². In AOP, new programming language constructs are provided that permit a better modularization of concerns that crosscut the solution space [Kiczales et al., 01].

A core characteristic of AOP is its ability to provide quantification and obliviousness [Filman and Friedman, 00]. *Quantification* is the notion that a programmer can write single, separated statements that introduce effects across numerous locations in the source code. Thus, quantification would provide the capability for saying the following: “In programs P, whenever condition C arises, perform action A” [Filman, 01]. The property of *obliviousness* holds when the quantified locations do not require modification in order to incorporate the effects of the quantification.

Our research focus has been the application of these new modularization techniques to the area of model-integrated computing (MIC). In our research, we are working at providing constructs that allow properties contained within a domain-specific model to be separated out from specific model elements. In essence, this allows us to quantify properties over a model, rather than adding a property manually to each model element.

In this brief paper, we will present an example application of our approach. A simple example will be given that illustrates the weaving of constraints representing processor assignment of tasks within a domain-specific model. A detailed description of the specific technique can be found in [Gray et al., 01]. We conclude the paper with a section that outlines a few of our goals in attending this workshop.

Problem Description

In order to introduce the sample problem, consider the diagram in Figure 1. This represents a simple model that contains 5 components. The first component is an inertial sensor. This sensor outputs, at a 100Hz rate, the position and velocity deltas. A second component is a position integrator. It computes the absolute position of the aircraft given the deltas received from the sensor. It must match the sensor rate such that there is no data loss. The weapons release component uses the absolute position to determine the time at which a weapon is to be

¹ This work has been supported by the DARPA Information Technology Office (DARPA/ITO), under the *Program Composition for Embedded Systems* (PCES) program, Contract Number: F33615-00-C-1695.

² See the special issue on AOP in the October 2001 issue of *Communications of the ACM*.

deployed. It has a fixed period of 20Hz and a minimal latency requirement. A mapping component is responsible for obtaining visual location information based on the absolute position. A map must be constructed such that the current absolute position is at the center of the map. Updates to the map must be low-latency. A fifth component is responsible for displaying the map on an output device. Notice the frequencies, latencies, and worst case execution times (WCET) of these components.

The Generic Modeling Environment (GME) is the tool that we use in our research [Ledeczi et al., 01]. An equivalent representation of Figure 1 can be found in the GME model of Figure 2. This model represents the interaction among the various components of the weapons deployment application.

Each of the components in Figure 2 has internal details that can also be modeled. For instance, the contents of the Compute Position component can be found in Figure 3. As can be noticed from the internals of this component, the series of interactions actually take place using a publish/subscribe model. The figure specifically highlights the attributes of a method called “compute” (see the bottom-right of the figure). The attributes provide the name of the method, the C++ source file that contains the method, and the method’s estimated WCET.

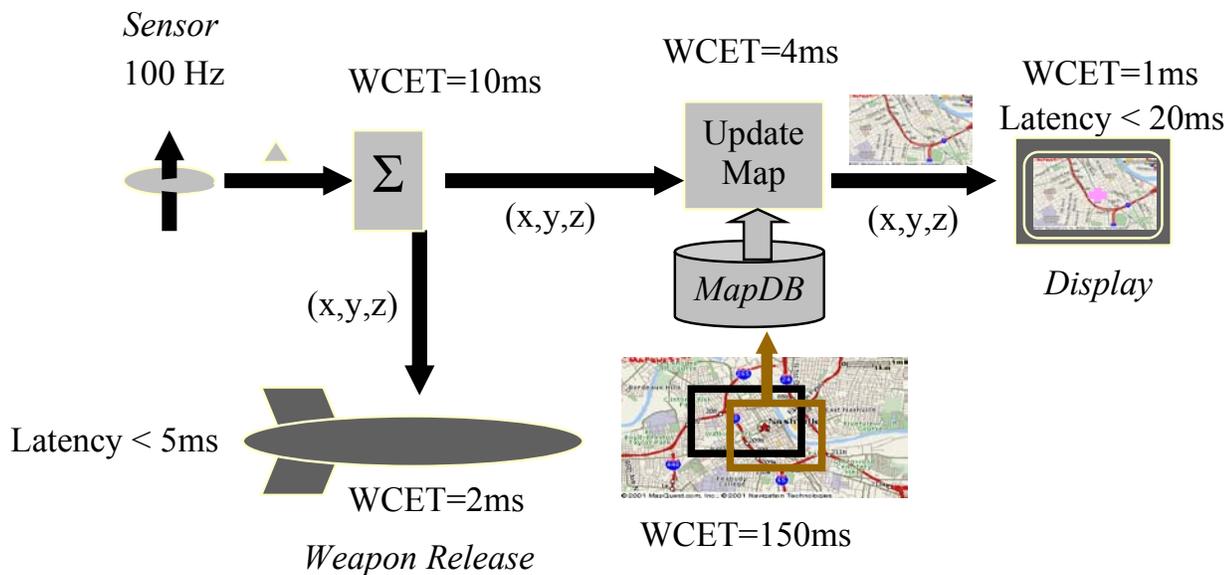


Figure 1. A Weapons Deployment Model

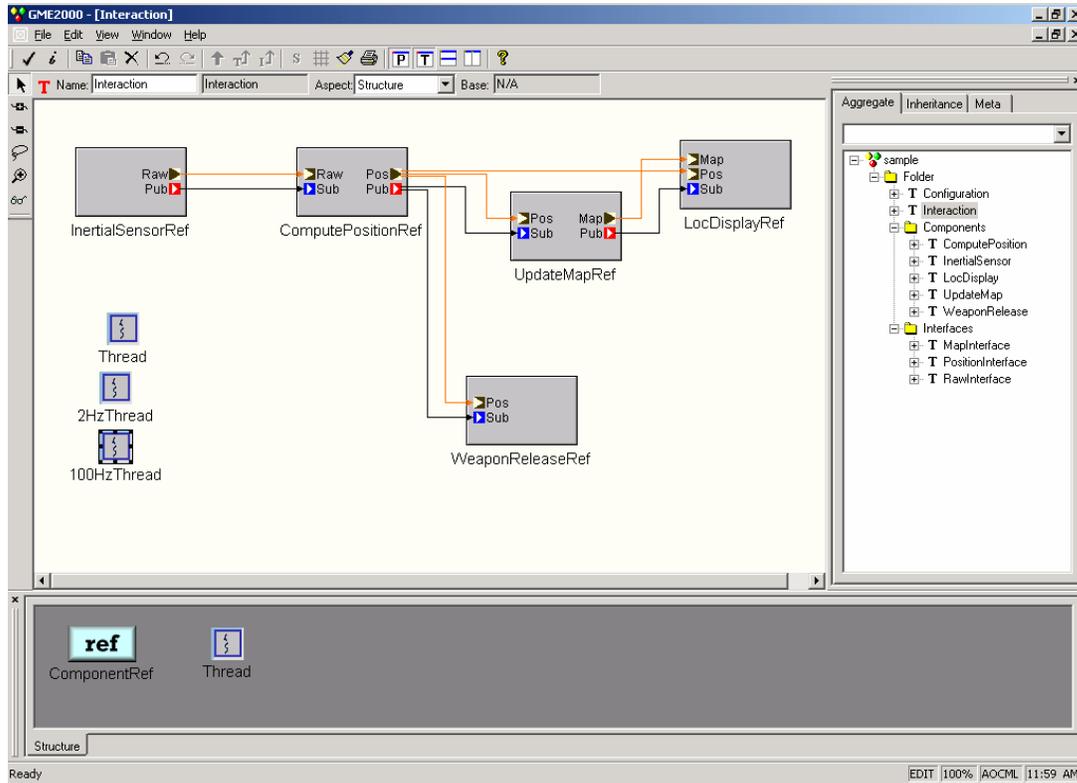


Figure 2. A GME Model of the Component Interactions

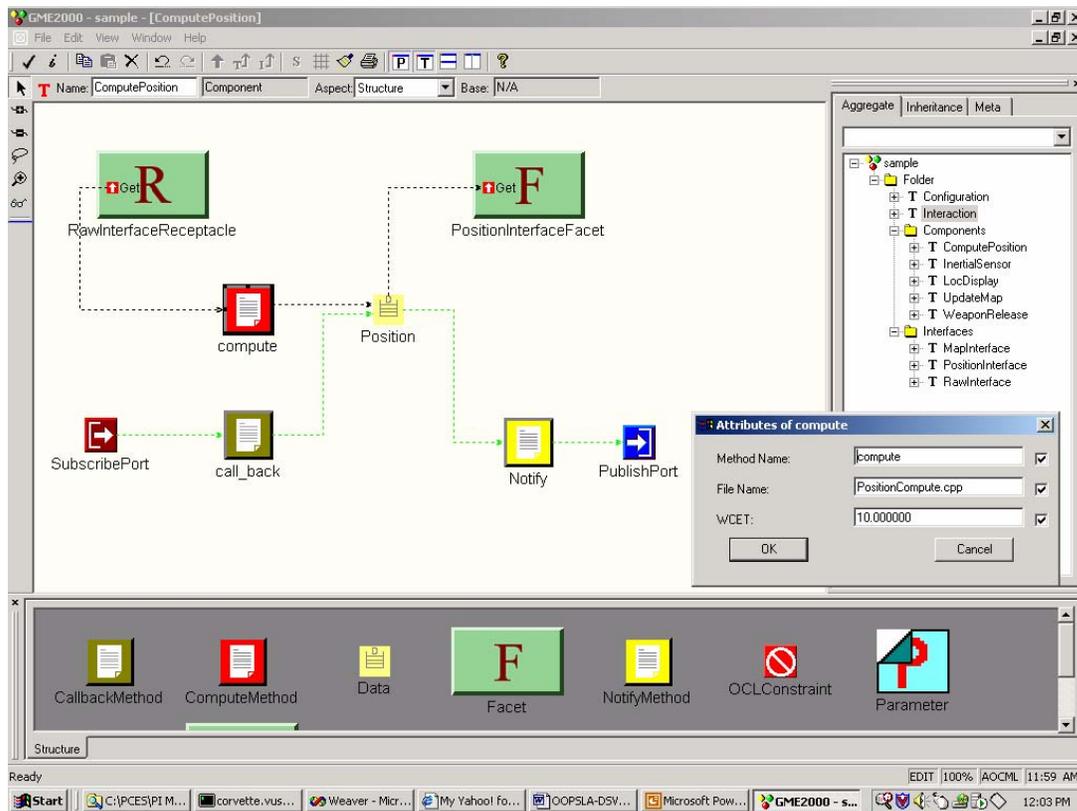


Figure 3. The Internals of Compute Position

Weaving Constraints: Processor Assignment

Suppose that we also wanted to model the processor assignment of each component. That is, based upon the expected execution times, the component methods are executed as tasks on various processors. A notation is needed to specify the assignment of methods/tasks to processors. The way that we chose to accomplish this is to specify the processor assignment as a constraint of the component model. A further piece of our work, which is not described here, is to use these constraints during design space navigation [Neema and Ledeczki, 01].

The way that processor assignment is typically modeled involves the application of a set of heuristics that globally assign tasks to processors based on certain properties of each task. In modeling, this is often done by hand and requires the modeler to visit each component or task to manually apply the heuristic. For a model with a large number of components, this can be a daunting task. It becomes increasingly unmanageable in situations where the modeler would like to play “what-if” scenarios. These “what-if” scenarios are used to drive the iterative evolution of the model, such that intermediate scenarios may even be discarded. This is helpful because a modeler may want to change the values of different properties, or even modify the details of the heuristic, in order to observe the effect of different scenarios. A manual application of a heuristic would require that the modeler re-visit every component and re-apply the rules of the heuristic.

To provide for quantification over models, we have developed a weaver that allows the separation of concerns from the hierarchical structure of the model. This would allow properties of models to be modularized. The process for using this weaver is shown in Figure 4. Here, the contents of a model are exported as an XML file. The modeler also writes specification aspects (these may be specific to a given model in a given domain) that are used to describe the locations in a model to apply a specific strategy, or heuristic. The weaver then takes as input both the XML model and the specification aspects. The weaver outputs a new XML model that contains the integration of the concerns into the model.

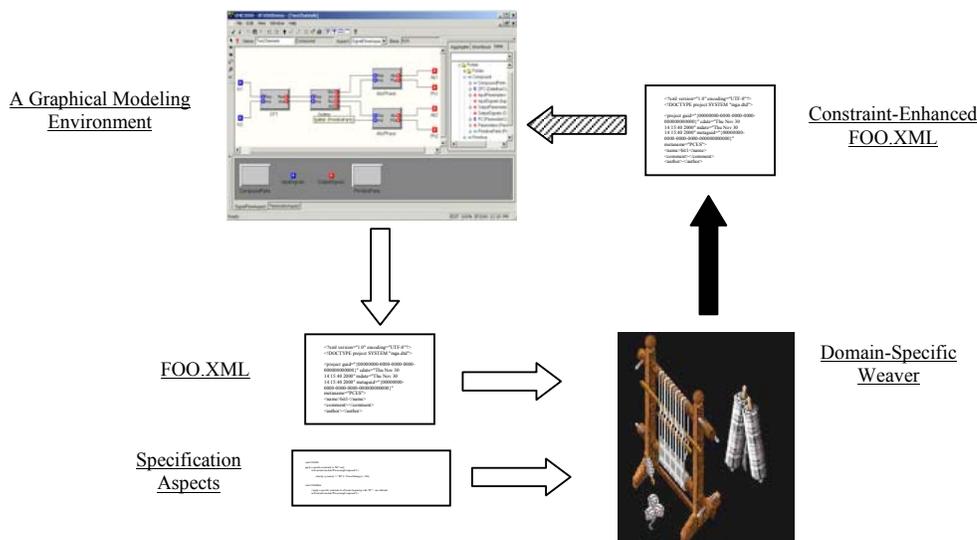


Figure 4. Process of Using a Domain-Specific Weaver

An example of a specification aspect and strategies can be found in Figure 5. We have developed a language called the Embedded Constraint Language (ECL) to specify the application of global properties/concerns throughout a model. The ECL is based on an extension of the Object Constraint Language (OCL). The interpretation of the aspect called ProcessorAssignment is that an iteration is specified over all of the modeling elements that are of type “Component.” The strategy called Assign is then invoked on each of these modeling components (here, a parameter bound to the value 10 represents a threshold of the execution time for each processor load). The purpose of the Assign strategy is to look into the “compute” method of each component and find its WCET. The WCET of each component are accumulated. Whenever this accumulated value reaches past the threshold, a new processor is created for component assignment. Assign will finally call another strategy, named ApplyConstraint, which will add a new constraint to the model. The new constraint, in this case, represents the processor assignment.

Figure 6 shows the same component that was given in Figure 3. The only difference is that the component now contains a constraint that was added by the weaver as a result of applying the strategies described by the specification aspect. Notice that the constraint has assigned this component to processor 1. An examination of all the other components involved in this interaction would reveal that different components are assigned to different processors based on their WCET and the parameterized threshold.

Conclusion and Future Research

Our goal in this paper was not to present an overview of our research; that can be found in [Gray et al., 01]. Rather, our intent was to present a simplified example of how an aspect weaver is utilized within domain-specific modeling.

As shown in Figure 4, the current weaver is detached from the modeling environment. We did this with this first weaver in order that we might explore the possibilities of using the approach in tools other than the GME. We are continuing to investigate these possibilities so we required a tool-independent way of weaving. It certainly is possible for the weaver itself to be included within the modeling tool. For example, the GME supports the concept of model interpreters that can be executed from within the GME. A weaving tool could certainly be constructed as an interpreter. That may be a future area of investigation for us.

Another potential area of future research is in the representation of specification aspects, or even strategies. Currently, as shown in Figure 5, this is done using a textual language. It would be interesting to investigate visual representations of these concepts.

```

strategy ApplyConstraint(constraintName : string, expression : string)
{
    addAtom("OCLConstraint", "Constraint", constraintName).addAttribute("Expression", expression);
}

strategy Assign(limit : int)
{
    <<static int accumulateWCET = 0; static int processNum = 1; int currentWCET; >>
    findAtom("compute").findAttribute("WCET").getInt(currentWCET);
    <<accumulateWCET = accumulateWCET + currentWCET; >>
    if (limit < accumulateWCET) then
        <<accumulateWCET = currentWCET; processNum++; >>
        self.ReportNewProcessor();           -- do some output here - omitted for brevity
    endif;
    <<ComBSTR aConstraint = "self.assignTo() = processor" + XMLParser::itos(processNum); >>
    self.ApplyConstraint("ProcessConstraint", aConstraint);
}

aspect ProcessorAssignment()
{
    modelParts("Component")->forAll(Assign(10));
}

```

Figure 5. Sample Strategies and Specification Aspects

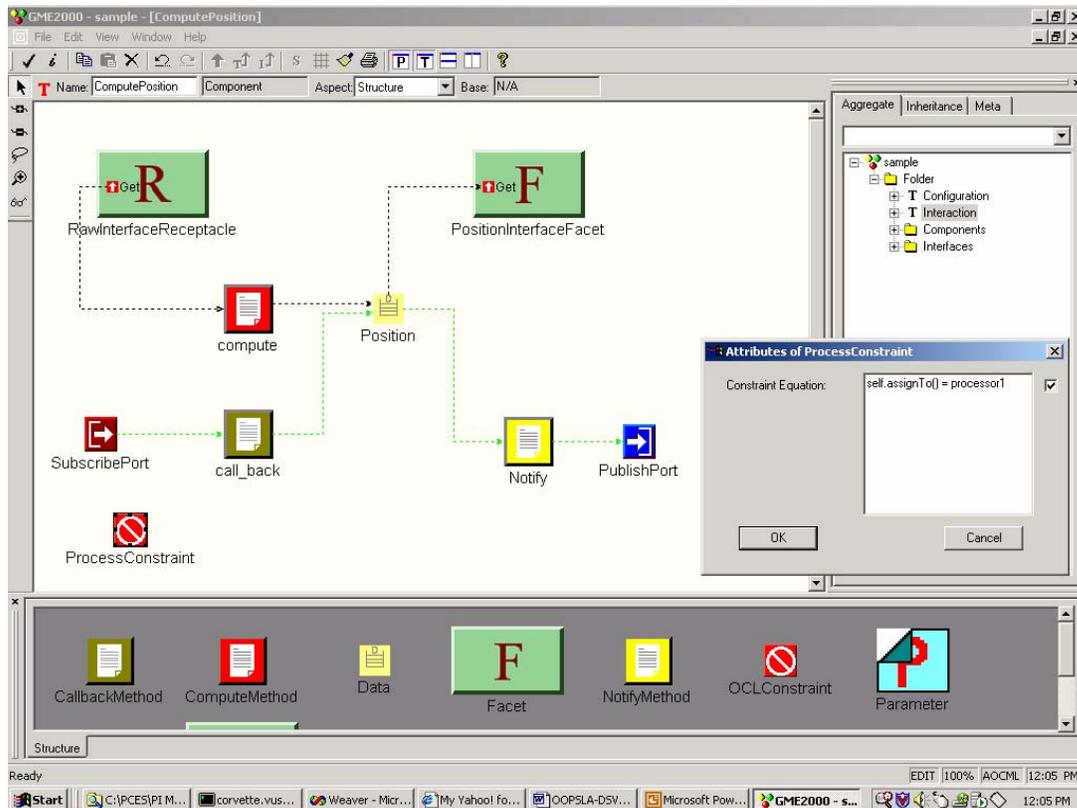


Figure 6. Component with Weaved Constraint

Addendum: Workshop Goals

There are a few goals that we would like to accomplish at this workshop. One of our main interests is to interact with others who have experience in the area of domain-specific visual modeling. We would like to explore potential collaboration opportunities. Specifically, we are interested in seeing if the concept of a weaver would be useful with other tools that offer the capability of persistently storing models as XML. We would like to see if our weaver can be applied to tools other than the GME.

One of our new application areas of this technology is in modeling CORBA middleware and components. In fact, the GME models shown in this paper are from a domain-specific paradigm for modeling CORBA components. We would be interested in hearing if anyone else at the workshop has done related work.

Finally, we would like to have some discussion on the topic of model evolution/migration. Each model that is created from within a visual modeling tool is based upon an underlying meta-level description, or paradigm. When the paradigm evolves, this can have a negative effect on existing models. The reason for this is due to the fact the variations made to the paradigm often represent syntactic and semantic changes, thus, potentially rendering previous models invalid under the new modified paradigm. How have other workshop participants overcome such problems? We have a few ideas that we can present on this, but we would like to see it as an item open for general discussion at some point in our interactions.

References

- [Filman and Friedman, 00] Robert Filman and Dan Friedman, "Aspect-Oriented Programming is Quantification and Obliviousness," *OOPSLA Workshop on Advanced Separation of Concerns*, Minneapolis, MN, October 2000.
- [Filman, 01] Robert Filman, "What is Aspect-Oriented Programming, Revisited," *ICSE Workshop on Advanced Separation of Concerns*, Toronto, Ontario, Canada, May 2001.
- [Gray et al., 01] Jeff Gray, Ted Bapty, Sandeep Neema, and James Tuck, "Handling Crosscutting Constraints in Domain-Specific Modeling," *Communications of the ACM*, October 2001.
- [Kiczales et al., 01b] Gregor Kiczales, Eric Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William Griswold, "Getting Started with AspectJ," *Communications of the ACM*, October 2001.
- [Ledeczi et al., 01] Akos Ledeczi, Arpad Bakay, Miklos Maroti, Pter Volgyesi, Greg Nordstrom, Jonathan Sprinkle, Gabor Karsai, "Rapid Composition of Domain-Specific Design Environments," *IEEE Computer*, November (?) 2001.
- [Neema and Ledeczi, 01] Sandeep Neema and Akos Ledeczi, "Constraint Guided Self-Adaptation," *International Workshop on Self-Adaptive Software*, Balatonfured, Hungary, May 2001.