Aspectifying Constraints in Model-Integrated Computing[1]

Jeff Gray, Ted Bapty, Sandeep Neema

Institute for Software Integrated Systems (ISIS)
Vanderbilt University, Nashville TN 37235
http://www.isis.vanderbilt.edu
{jgray, bapty, neemask}@vuse.vanderbilt.edu

Abstract

This position paper identifies some modeling problems that arise when the dominant decomposition is based upon the functional hierarchy of a physical system. In particular, constraints that are used to specify global system properties (e.g., latency, power consumption, precision, timing) often cross-cut the boundaries of the model hierarchy. This not only produces much repetition with respect to the placement of constraints, but it also makes it difficult to reason about the overall effects on the system. The paper presents a few ideas that are being considered to ameliorate this problem. An aspect-oriented approach is being pursued that provides a separate module for specifying constraints and their propagation.

Introduction

> "The crucial choice is, of course, what aspects to study 'in isolation', how to disentangle the original amorphous knot of obligations, ***constraints*** and goals into a set of 'concerns' that admit a reasonably effective separation." [Dijkstra, 76]

Among the new techniques for handling separation of concerns, much of the core focus has been at the programming language level. While there has been work on applying these techniques at the design level (see [Clarke et al., 99] and [Herrero et al., 00]), there are obvious benefits in taking these techniques to even higher levels of abstraction. For instance, the modularization of cross-cutting concerns can also prove beneficial in certain kinds of modeling. The dominant form of decomposition provided by most modeling tools is concentrated on the functional hierarchy of the system being modeled. In such modeling environments, all other concerns must be spread about the model in a non-modular manner.

In the types of models that we build at ISIS/Vanderbilt to support our research, constraints represent a type of cross-cutting concern. In these models, constraints are used to specify things like bit precision, latency, and timing concerns. We use a variant of the OCL to specify these things. The same problems that result from tangled-code in programming languages, as identified in [Kiczales, 00], also occur in the tangled-constraints of our models. To wit,

- Redundant constraints: Often, the same constraint is repetitiously applied in many different places in a model. It would be beneficial to describe a common constraint in a modular manner and designate the places where it is to be applied. With respect to code, an amazing amount of redundancy can be removed using aspect oriented techniques [Lippert and Lopes, 00]. We think the same will apply to our models and constraints.
- Change management: It is difficult to consistently change a constraint if it has been "gunshot" across the model. This is an error prone activity that also makes it nearly impossible to play "what-if" scenarios based upon varying sets of different constraints.
- Modular reasoning: It is difficult to reason about the effects of a constraint when it is spread out among the numerous nodes in a model. A modular mechanism for separating this concern can be an aid toward improving the understanding of a given model.

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

Problem Background

Model-Integrated Computing (MIC) has been developed over a decade at Vanderbilt University for building embedded software systems. It is an approach to developing systems that directly addresses the problems of system integration and evolution by providing rich, domain-specific modeling environments including model analysis and model-based program synthesis tools. This technology is used to create and evolve integrated, multiple-view models using concepts, relations, and model composition principles routinely used in the domain specific field. MIC also facilitates systems/software engineering analysis of the models, and provides for the automatic synthesis of applications from the models. The approach has been successfully applied in several different applications, including automotive manufacturing [Long et al., 98], digital signal processing [Sztipanovits et al., 98], and electrical utilities [Moore, 00], to name a few.

A core tool in MIC is the Graphical Model Editor (GME). The GME is a domain-specific modeling environment that can be configured and adapted from meta-level paradigm specifications [Nordstrom et al., 99]. Thus, based upon the paradigm, the GME can be adapted quickly from a UML modeling tool to a domain specific tool that represents an environment for modeling an automotive manufacturing plant. Figure 1 shows an example screen shot from a particular sub-component of a model in GME. Note the different kinds of domain-specific modeling types (atoms) that are available, as partially shown in the right-hand side of the screen.

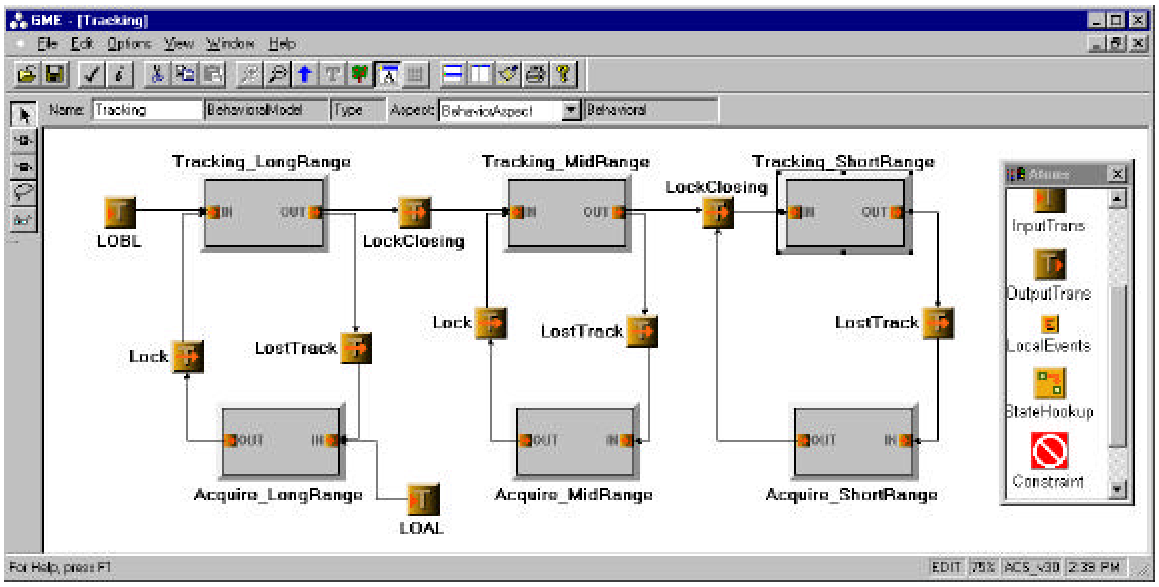
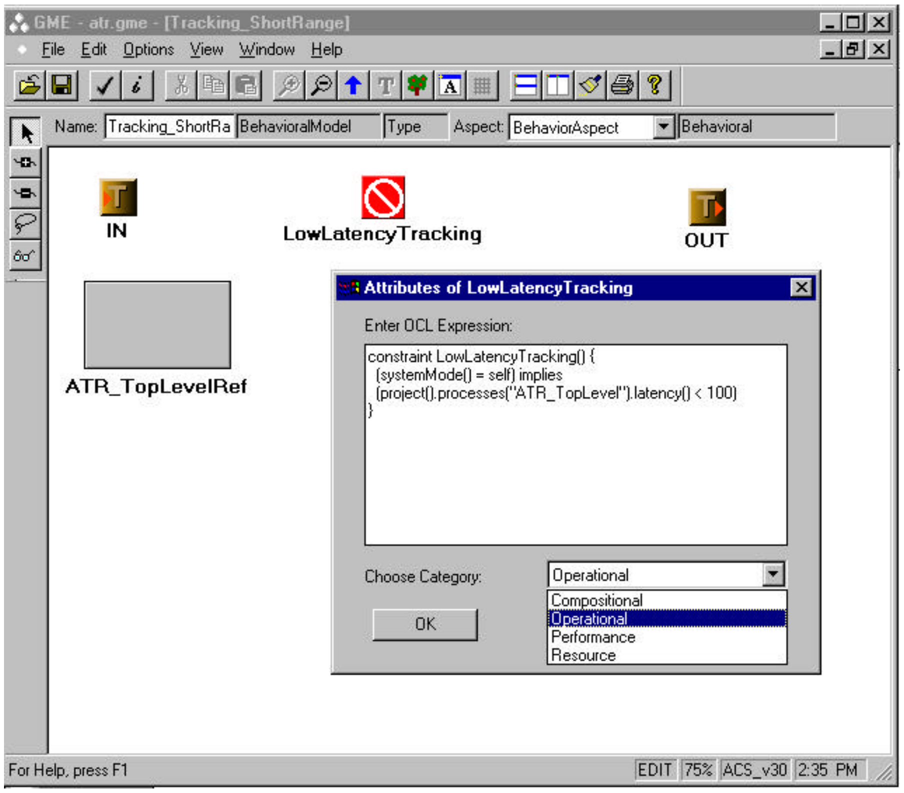


Figure 1 – Graphical Model Editor (GME)
ATR Behavioral Model, Tracking Drill-Down

The Adaptive Computer Systems (ACS) project is a DARPA funded effort. The goal of ACS is to develop high-level system design tools for the construction of dynamically reconfigurable, high performance embedded systems using adaptive computing technology [Bapty et al., 00]. The objective is to extend the state of the art in the design of dynamically reconfigurable systems. The requirements mandate a highly customizable computational platform that must conform to the underlying algorithms, optimize the use of hardware resources to achieve performance goals, and minimize the hardware used to meet size and power goals. The computational platform cannot, however, be optimized to a single algorithm, since the best algorithm will depend on the changing environment and functional requirements. The platform must dynamically reconfigure itself, constantly optimizing the use of extremely limited physical resources. We are applying MIC and the GME to produce a domain-specific environment that is customized to the application area.

The focused challenge problem that we selected for ACS is an embedded Automatic Target Recognition (ATR) application for Air-to-Ground missiles. The models for this problem define a design space of potential alternative implementations along several dimensions. This design space can be very large (10^{24} alternatives or larger). Integrated tools have been developed to help the user in selecting potential designs.

Due to the large number of conflicting design criteria in reconfigurable systems, constraints are used to aid in the reduction of the number of design states that must be examined. In the current ACS modeling paradigm, there are four categories of constraints that can be specified. Operational constraints can be used to restrict design space alternatives based upon the operational mode of the model. Composability constraints express compatibility between different alternatives. They can be used to restrict alternatives that are not compatible with each other. Resource constraints are used to indicate specific hardware resources that are needed by software modules. Performance constraints are widely used in our models. These constraint expressions indicate the end-to-end latency, throughput, power consumption, and bit precision.



Name: Tracking_ShortRa BehavioralModel Type Aspect: BehaviorAspect Behavioral

Enter OCL Expression:

```
constraint LowLatencyTracking() {
  (systemMode() = self) implies
  (project().processes("ATR_TopLevel").latency() < 100)
}
```

Choose Category: Operational

Figure 2 – Example Constraint

Figure 2 shows a simple constraint from an ATR model. This constraint states that the overall latency for the system, while in short-range mode, should be less than 100ms. Simple constraints like this are spread throughout the model in various locations. Much more detailed constraints can be given that are expressed in terms of the parents and children of the component that contains the constraint. Often, what we would like is the ability to express a global system-wide constraint and have it propagated to all relevant components. We cannot do this now, but we feel our investigation in applying aspect-oriented techniques will assist in this desired ability to disentangle our constraints.

A New Approach

As noted in the Introduction, a drawback of our current approach can be seen in the sprinkling of many constraints throughout various levels of our model. It is difficult to maintain and reason about the effects and purpose of constraints when they are scattered about. We need a mechanism to separate out this concern.

The requirements for our new approach necessitate a different type of weaver from those that others have constructed in the past. A new weaver is desired that will allow propagation of constraints to sub-objects. As the weaver visits a particular node in the object graph, it must know how to diffuse that constraint to its underlings. Depending on the type of node, the diffusion process may be performed in different ways. To provide the weaver with the needed information to perform the propagation, a new type of aspect is needed.

We call this a “strategy aspect.” (We chose this name because it is similar in intent to the Strategy pattern [GOF, 95]). Strategy aspects are specified independently of any particular model and are joined to the object graph by the weaver.

The intent of a strategy aspect is to provide a hook that the weaver may call in order to process the node-specific constraint propagations. Strategy aspects must be weaved prior to the weaving of constraint aspects. As the weaver visits each object node during constraint weaving, it may call upon the strategy aspect to aid in applying the constraint to that object. Thus, strategy aspects provide numerous ways for instrumenting object nodes in the object graph with constraints. An important design goal of this aspect language will be to decouple the strategy aspects from the constraint aspects.

The high-level block diagram of our new approach is shown in Figure 3. Existing object graph hierarchies define a system design space. These object hierarchies are augmented with an Aspect-Oriented system constraint language. Along with the constraints, a Propagation/Distribution strategy must be specified to describe how constraints are mapped and allocated to objects. The weaving process gathers these specifications and weaves the AO constraints into low-level constraints applied to individual components in the object hierarchy.

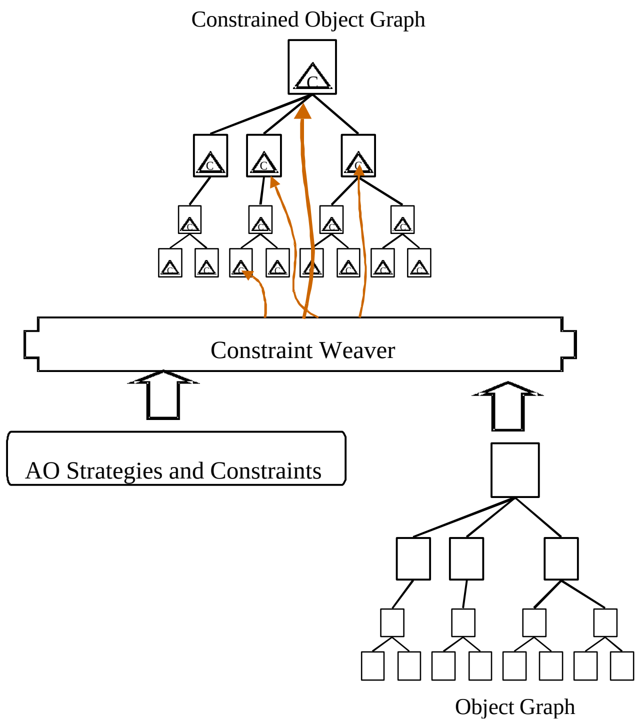


Figure 3 – Input/Output of Constraint Weaver

To illustrate the proposed approach, consider a scenario of stages that the weaver would enter:

- First, the weaver will process all of the strategy aspects and join them with object nodes in the object graph. Each strategy will expose a hook from the object. This hook represents a particular method for processing a constraint and propagating that constraint to children.
- Once the strategies have been weaved, it is then possible to join the strategized object graph with the aspect constraints. The different inputs to the constraint weaver do not mean that the user must invoke the weaver twice. Rather, it indicates the two phases that occur within the weaver and the input to each phase.
- Figure 4 shows how strategy aspects diffuse a constraint aspect. An object node in the object graph may receive a constraint aspect from either the weaver or a parent strategy (1). When applying the constraint, the strategy may modify the container object (2) as well as diffusing the constraint to underlings (3). Note that we have specified the direction of propagation to go from parents to sub-objects. At a later time we plan to investigate strategies that propagate upward.

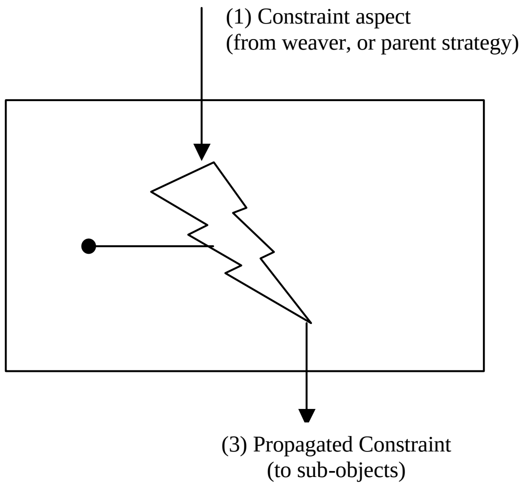


Figure 4 - Application of Strategy to a Graph Node

Note that if strategies are defined on each object, then a single constraint can cause the weaver to visit each object in the graph. That is, one global aspect is diffused across the entire object graph. Also, strategy aspects disappear from the resultant object graph after weaving is completed. The only noticeable change in the object graph is the addition of the constraint aspects.

The Adaptive Programming community has recently addressed issues concerning the separation of aspects from their join points [Lieberherr et al., 99]. In their approach, called "aspectual components," a definition that is separate from the aspect description is used to describe the assembly of join points. We would like to explore further this idea with our own language so that our constraint aspects are more generic and can be reused in other object hierarchies that we may generate.

Summary

> "Even for this let us divided live…That by this separation I may give that due to thee which thou deservest alone." William Shakespeare, *Sonnet XXXIX*

The lack of support for separation of concerns along multiple dimensions is not solely indigenous to the coding phase. We have found that the source of some of our modeling problems is directly related to a lack of support for modularizing constraints. As we adopt an aspect-oriented approach to our modeling, we feel that the maintainability, understandability, and evolvability of our models will be greatly enhanced.

References

[Bapty et al., 00] Ted Bapty, Sandeep Neema, Jason Scott, Jans Sztipanovits, and S. Asaad, "Model-Integrated Tools for the Design of Dynamically Reconfiguable Systems," *VLSI Design*, to appear, Fall 2000.

[Clarke et al., 99] Siobhan Clarke, William Harrison, Harold Ossher, and Peri Tarr, "Subject-Oriented Design: Towards Improved Alignment of Requirements, Design, and Code," *In Proceedings of OOPSLA '99*, November 1999, pp. 325-339.

[Dijkstra, 76] E.W. Dijkstra, *A Discipline of Programming*, Prentice-Hall, 1976.

[GOF, 95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[Herrero et al., 00] Jose Herrero, Fernando Sanchez, Fabiola Lucio, and Migeul Torro, "Introducing Separation of Aspects at Design Time," *ECOOP 2000 Workshop on Aspects and Dimensions of Concerns*, June 2000.

[Kiczales, 00] Gregor Kiczales, "AspectJ™: aspect-oriented programming using Java™ technology," *JavaOne*, June 2000.

[Lieberherr et al., 99] Karl Lieberherr, David Lorenz and Mira Mezini, "Programming with Aspectual Components," NU-CCS-99-01, College of Computer Science, Northeastern University, March 1999.

[Lippert and Lopes, 00] Martin Lippert and Cristina V. Lopes, "A Study on Exception Detection and Handling Using Aspect-Oriented Programming," *In Proceedings of ICSE'2000*, Limmerick, Ireland. June 2000.

[Long et al., 98] Earl Long, Amit Misra, and Janos Sztipanovits, "Increasing Productivity at Saturn," *IEEE Computer*, August 1998, pp. 35-43.

[Moore et al., 00] Michael Moore, Saeed Monemi, Jianfeng Wang, James Marble, and Steve Jones, "Diagnostics and Integration in Electrical Utilities," *IEEE Rural Electric Power Conference*, May 2000.

[Nordstrom et al., 99] Greg Nordstrom, Janos Sztipanovits, Gabor Karsai Akos Ledeczi, "Metamodeling - Rapid Design and Evolution of Domain-Specific Modeling Environments," *IEEE ECBS '99 Conference*, Nashville, Tennessee, April 1999.

[Sztipanovits et al., 98] Janos Sztipanovits, Gabor Karsai, and Ted Bapty, "Self-Adaptive Software for Signal Processing," *Comunications of the ACM*, May 1998, pp. 66-73.