The Generic Eclipse Modeling System

Jules White, Douglas C. Schmidt, Sean Mulligan
Vanderbilt University, EECS Nashville, TN, USA
{jules,schmidt}@dre.vanderbilt.edu

ABSTRACT

The Generic Eclipse Modeling System (GEMS), a part of the Eclipse Generative Modeling Technologies (GMT) project, gives domain experts the ability to generate the implementation of a graphical modeling tool from a visual language specification (metamodel). Domain experts that use GEMS can create an Eclipse-based graphical modeling tool without intimate knowledge of the Eclipse Modeling Framework, Graphical Editor Framework, Graphical Model Framework, or Eclipse Plug-in API. GEMS imbues each generated modeling tool with complex constraint checking and constraint-solver guided modeling assistance. Stylesheets can be applied to GEMS modeling tools to customize the graphical interface and separate visualization properties from Java code. Finally, GEMS provides a remoting infrastructure that allows modeling tools to connect to remote sensors, GEMS models, and other CORBA or Java RMI-enabled clients. This paper focuses on the stylesheet, remoting, and constraint-solver modeling guidance capabilities of GEMS. These features of GEMS are discussed in the context of an example application for building interactive television applications.

1. INTRODUCTION

Model-Driven Development (MDD) has become a popular approach to creating solutions in a large number of domains. Most visual MDD tools are built using the Model View Controller (MVC) pattern [1]. Eclipse provides numerous frameworks to support development of MVC-based graphical modeling tools. The Draw2D framework provides the basic visualization mechanisms to draw the Visio-like diagrams common in MDD tools. The Graphical Editor Framework (GEF) provides inversion of control for loading a graphical model, creating controllers for each model element, and constructing and associating views with the controllers. The Eclipse Modeling Framework (EMF) contains various APIs for building object graphs for a model, serializing and de-serializing object graphs, and enforcing basic constraints on the graphs. Finally, the Graphical Modeling Framework (GMF) allows developers to develop complex controller logic from specifications that map a model to the different elements of a view.

Even with all of these powerful frameworks, developing a model-driven development tool is still challenging. In most circumstances, stock visualizations from the framework are not acceptable and complex coding in GMF, Draw2D, or GEF is needed. Furthermore, all of these frameworks assume that developers possess intimate knowledge of Eclipse APIs and concepts and thus are not easy for domain experts to leverage. Even if a domain expert has the required skills to implement a tool, performing the complex coding to implement a graphical modeling tool subtracts from the time the domain expert spends on the key assets of these tools; the complex domain validation, code-generation, optimization, and simulation capabilities. Finally, if a tool is developed with a traditional manual EMF, GEF, and GMF approach, there is a significant cost associated with maintaining the implementation, as understanding of the domain deepens and the tool and modeling language requirements change.

The Generic Eclipse Modeling System (GEMS) is a part of the Eclipse Generative Modeling Technologies (GMT) project. GEMS allows domain experts to rapidly create complex graphical modeling tools for Eclipse without writing Graphical User Interface (GUI), EMF, or XML code. Developers create a visual specification of the metamodel for a modeling language and GEMS generates the requisite GEF, EMF, Draw2D, and Eclipse code to implement an editor for the modeling language. GEMS is an open-source project, that has been developed in conjunction with Siemens CT SE2, IBM, and Prismtech.

By automatically generating the needed code artifacts to implement a modeling tool, GEMS substantially reduces the implementation cost of a graphical modeling tool. With less implementation work needed for the MVC components of the modeling tool, developers can focus on the key aspects of their tool: the specification of the modeling language and the intellectual assets built around the use of the language, such as code generation capabilities. As the supporting EMF/GEF/Draw2D and Eclipse infrastructure evolves or the requirements of the modeling tool change, GEMS can regenerate the implementation of the tool to meet the new requirements. By using GEMS, developers save not only the initial implementation cost but the continued support and refactoring costs. A final key attribute of GEMS is that it separates the graphical specification into style-sheets and allows domain-experts or graphic designers to create complex visualization without GUI coding. This separation of the visualization properties from the metamodel provides a development process similar to web-based applications.

A GEMS-generated graphical modeling tool can have both static and dynamic visual behaviour specified through stylesheets. Graphic

designers or developers can create styles and icons that should be applied to elements when they are in specific states. For example, a developer building a tool for specifying the deployment of interactive television applications to set-top boxes could develop separate icons and styles for drawing an application when it is deployed and undeployed.

This paper provides an introduction to GEMS and its remoting, stylesheet, and constraint solving capabilities. Each GEMS feature is presented in the context of an interactive television example. The remainder of this paper is organized as follows: Section 2 presents the interactive television example that is used throughout the paper; Section 3 presents the remoting, stylesheet, and constraint solving capabilities of GEMS in the context of the television example; and Section 4 presents concluding remarks.

2. MOTIVATING EXAMPLE

An application for building interactive television applications is used as the motivating example throughout the paper. The interactive television scenario is outlined in the workshop announcement [2]. After further discussion, the project managers for the interactive television modeling tool have identified several additional challenges for the project.

2.1 Challenge: No Developers with GUI or IDE Coding Experience

In the process of assembling the development team for the project, the managers have realized that few of the developers have any experience developing graphical modeling tools or GUI-based applications. Most of the developers have only done back-end image processing, data archival, and web application development. The organization's typical development projects are web-based and thus the User Interface (UI) components of the applications are built by graphic designers, web developers, and some developers with Java Server Page expertise. None of the developers have built tools for Eclipse, Visio, or any of the other potential modeling platforms.

Due to the lack of developers with UI experience available for the project, the managers have decided that they wish to use a tool that provides a development process more similar to a web-based application. The managers would like to have an IT-architect design the overall structure of the modeling language, a group of experienced back-end programmers develop the code generation portion, and a web-development team focus on the visualization aspects. The managers also do not want any of the developers to learn any techniques or APIs specific to the Integrated Development Environment (IDE) that the modeling tool will be built on top of.

2.2 Challenge: Remote Model Access

The advertising department for the organization has contacted the project manager and expressed its desire to be involved in the project. The advertising department would like to be able to use the modeling tool to monitor poll results as they arrive. The advertising department would also like to change the interactive television menus delivered to subscribers on the fly to include new advertisements targeted towards the trends from the interactive polls. The department has a polling statistics package, developed with Java, that they would like to interface with the tool.

The statistics package requires significant processing power and cannot be co-located with the modeling tool. The package needs to be able to remotely capture the contents of the models via Java RMI and publish updates to the interactive menus based on their analyses. Furthermore, the modeling tool needs to be able to trigger the menu code generation routines when these updates arrive and push the new interactive television code for the menus out to the server that communicates with the set-top television boxes. The server that communicates with the set-top boxes is only accessible via CORBA. The overall architecture for the dynamic ad placement mechanism can be seen in Figure 1.

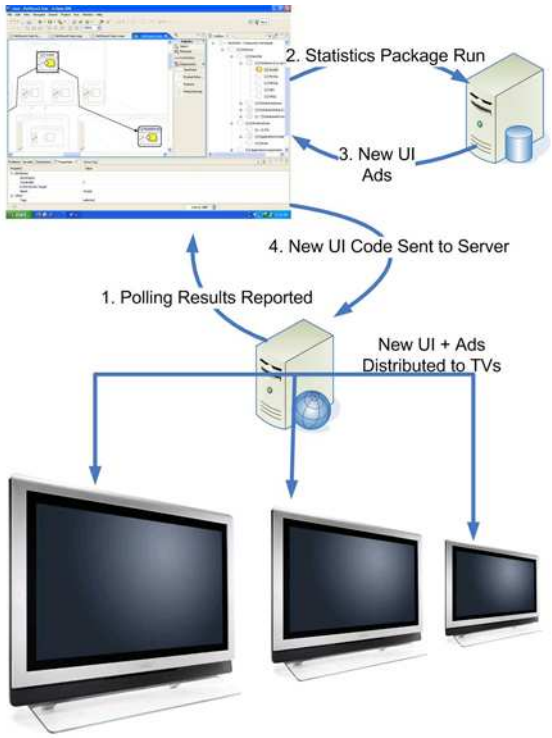

Figure 1: Architecture for Dynamic Ad Placement

2.3 Challenge: Heterogenous Set-Top Box Capabilities

The organization began rolling out its interactive television service over 5 years ago. Due to the long span of time over which the organization has offered interactive television service, they now have a large number of customers with multiple models of set-top boxes and interactive television software packages. Generally, once a customer signs up for the service their set-top box is installed by a technician and not replaced unless absolutely necessary. Over the five years, the organization has rolled out multiple types of hardware with differing CPU, memory, HDTV, and other capabilities. The boxes also have a variety of interactive television platforms installed.

The organization does not want the producers to be limited to modeling GUIs that can be run on the lowest common denominator. Many of their customers pay for HDTV and they would like to provide each customer with the best possible GUI that their box can support. Thus, the modeling tool needs to be able to capture the requirements of the interactive applications and determine which boxes can support which GUIs.

Due to the large number of box types and range of capabilities, the modeling tool must provide support for helping modelers to

understand what applications can run on what boxes. Without this type of modeling guidance, it is unlikely the modelers will fully understand their design decisions. Furthermore, the tool must be able to use the requirements of the GUIs and capabilities of each box type to deduce the GUIs that will run on each box type. This capability to deduce the correct GUI for each box type must also be remotely accessible and automated so that the boxes can be updated on the fly by the dynamic ad injection mechanism described in the last challenge.

3. USING GEMS TO ADDRESS THE CHALLENGES OF INTERACTIVE TV MODELING

The challenges outlined in Section 2, require that the modeling platform chosen for the project be able to: 1) allow developers to create a graphical modeling tool and integrate it with an IDE without coding; 2) provide a visualization interface that can be styled in a manner similar to web pages; 3) be capable of exposing model instances remotely through CORBA and Java RMI; and 4) provide complex constraint solving infrastructure to derive the valid GUIs for a set-top box. In this section, we show how GEMS provides the capabilities to address each of these requirements.

3.1 Code-less UI Development and Stylable Interfaces

GEMS is based on the Model-Integrated Computing (MIC) paradigm [5] developed at Vanderbilt University's Institute for Software Integrated Systems (ISIS). MIC was originally pioneered in the Generic Modeling Environment [3]. With the rise in popularity of Eclipse and the Eclipse Modeling Framework, the Distributed Object Computing (DOC) group at ISIS has implemented a similar meta-programmable environment for Eclipse.

As shown in Figure 2, modelers can use GEMS to visually describe a metamodel in Eclipse and use GEMS to generate the EMF, Draw2D, and GEF code to implement a modeling tool for the language described by the metamodel. In step 1, domain experts use the modeling tool to build the metamodel for the language using GEMS' GEF-based visual metamodel editor. In step 2, the domain experts invoke the GEMS code generators to produce the GEF code, Eclipse plugin descriptor, EMF code, and other source artifacts to implement a graphical modeling tool for the language. In step 3, graphic designers use CSS style sheets to change the look and feel of the generated modeling tool's UI to match common domain notations. In step 5, domain experts or developers can attach OCL, Prolog, and Java constraints to the modeling language or specific model instances. In step 6, domain experts use the generated modeling tool to build instances of the modeling language.

With GEMS, domain experts do not need to write Java code, Ecore XML models, GMF code, or Eclipse plug-in descriptors to produce a modeling tool for Eclipse. GEMS takes care of generating all of these required artifacts from the visual metamodel specification. This code-generation frees domain experts from having to worry about the low-level implementation details typically associated with developing a modeling tool for Eclipse.

For the interactive television example, the producers would first use GEMS to specify a metamodel for the interactive television application. GEMS would then take care of generating the requisite Java code to implement the editor. This alleviates the producers

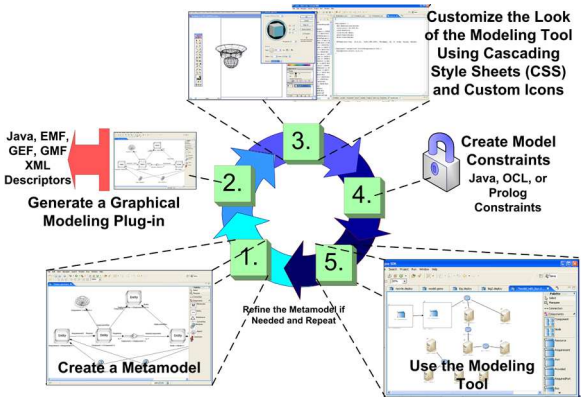

Figure 2: Developing an Eclipse Modeling Tool with GEMS

and developers from having to figure out how to implement the specification of the producers' desired modeling language.

3.2 CSS Styles

One of the novel aspects of GEMS is the ability to apply Cascading Style Sheet (CSS) styles to the generated modeling tools. Just as with web development, the GEMS approach to modeling-tool development separates the specification of the content and content structure from the visualizaton properties of the data. A domain-expert uses the GEMS metamodel to specify the structure and content captured in the models and graphic designers use styles to customize the look of the content.

The CSS style sheet language exposed by GEMS adds domain-specific modeling concepts. Traditionally, a CSS style sheet uses selectors that match structural elements in an HTML page. For example, the style:

```
p { font-size: large;}
```

would match all paragraphs (denoted by "p" HTML elements) and set the font size for those paragraphs to large. In GEMS CSS style sheets, the selectors match against the types defined in the metamodel. For example, in our implementation of the interactive television modeling application, *Menu* is a type of modeling element. To create a style sheet that changed the font used by the figure drawing the Menu modeling element, a graphic designer could create the following style sheet:

```
Menu { font-name: Verdana;}
```

In this example, "Menu" is being used as the selector and the name of the font face used by Menu elements is being set. The domain-specific selectors used by GEMS can rely on the inheritance hierarchies from the metamodel. For example, an element of type HDTVMenu that is derived from Menu, can have the following style applied to override the font color:

```
HDTVMenu { font-color: RGB(100,100,100);}
```

The HDTV menu will have both the Menu style and any properties from the HDTVMenu style applied. Properties overriden in the

HDTV style will supercede those from the Menu style. Styles can set complex properties of figures, such as layouts, text alignments, label formats, attributes attached to connections, and icons.

3.3 Dynamically Changing Styles

A typical scenario with a modeling tool is that the domain expert wants the way that a modeling element is visualized to change based on different types of domain analyses. In the interactive television example, a domain expert may wish to have a large red background applied to Menu that is so large that it will extend off the customers' television screens. Furthermore, the domain expert may want a small popup to appear above the element specifying the error that has created the red X.

Most modeling approaches use the Model-View Controller paradigm. Creating a graphical change, driven by a domain analysis, requires placing complex domain analysis code into the controller of the view. Implenting a domain analysis in a controller requires intimate understanding of the underlying GUI APIs and tedious and error-prone graphical coding. Often, domain experts change their mind about when these visualizations should be triggered and what they should look like, which causes costly code refactoring.

With GEMS, each model element can have a group of tags associated with it. GEMS tags are analogous to CSS class attributes. A CSS class attribute acts as a filter to reduce the scope that a style is applied to. For example:

```
p.menucontent { font-size: large;}
```

would only match HTML "p" elements with the "class" attribute set to "menucontent." The class attribute can then be used by the content producer to more specifically annotate different sections of the document. GEMS provides a similar facility for its modeling elements. GEMS styles can be scoped by the tags that are currently applied to a model element.

GEMS provides facilities to use domain analyses written in OCL, Java, Prolog, and other languages to dynamically apply tags to model elements in GEMS. For example, an OCL expression can be built that applies an "overflow" tag to any Menu element that has more than 10 children. A style can then be created that adds the appropriate red "X" to the menu and invokes a popup error message. GEMS' active constraint framework allows OCL, Java, and Prolog expressions to be dynamically invoked as the model changes and trigger actions if the expressions evaluate to true. A dynamic OCL or Prolog analysis can then swap styles on elements and create complex visual behaviors. The decoupling of the domain analyses from the controller and the separation of the analysis and visualization specification allows domain experts to create complex triggers from domain-specific events and graphic designers to specify complex visualizations for elements that are attached to the triggers.

To enable marking menus in red and displaying a popup error message, an OCL expression can be created to identify overflowing menus. In the example that follows, the evaluation of an OCL expression that checks for overflowing menus is set to be triggered by changes in the number of children of a menu. The analysis code is:

```
on: Menu.Children
evaluate: (self.children->size() * 10)
                > self.container.height
trigger: context.addTag("overflow")
```

This expression checks to ensure that the container (the panel displaying the menu) is tall enough to encompass all of the children of the menu. The expression evaluated by a trigger can be any Java, OCL, or Prolog expression that returns a boolean value. The graphic designer can leverage the domain analysis by creating a style that is applied when a menu element has the overflow tag. The style is:

```
Menu.overflow {
  background-color:RGB(200,0,0);
  show-message: This menu is too tall for its container.;
}
```

With a custom built editor using GEF and EMF, achieving this type of analysis-based dynamic visualization change would require writing complex controller logic. The controller logic would not be accessible to the graphic designer and could not be overriden on a per-model basis. Furthermore, each change to the desired trigger condition or visualization would require rewriting the controller logic.

3.4 Remotely Accessible Models

Typically, to enable remote communication with a modeling tool, complex remoting mechanisms must be developed from scratch. In some cases, generic remoting infrastructure can be developed and reused across models but these generic mechanisms often lack domain-specificity and are hard to develop complex domain-specific applications with. Domain-specific remoting mechanisms can also be developed but they tend to have limited reusability.

GEMS addresses the remoting challenge by automatically generating a domain-specific remoting mechanism with each modeling tool. GEMS-generated modeling tools include remoting capabilities for both CORBA and Java RMI. A unique property of the remoting mechanism exported by GEMS is that the interface to the CORBA and Java RMI classes is the same across modeling languages but the application-level remoting protocol used to interact with the models is domain-specific.

The domain-specific application-level protocol used by GEMS for each modeling tool is based on the terminology from the metamodel. If Menu is a type in the metamodel, Menu becomes a first class entity in the protocol. For example, the low-level protocol messages:

```
Menu(1),Name(1,"Foo"),SubMenus(1,[2,3])
```

would create a Menu item with ID 1, Name "Foo", and with SubMenu associations to the model elements with ID 2 and 3. Each statement in the protocl specifies a predicate, an element to which the predicate is applied, and the value that should be applied to the element. The statement "SubMenus(1,[2,3])" specifies that the element with ID 1 should have its "SubMenus" role set to the elements with IDs 2 and 3. Each attribute name, association role, containment role, and element type from the metamodel becomes a first-class entity in the protocol.

3.5 GEMS Remote Collaboration Architectures

The remoting mechanism generated by GEMS can be used to obtain data from devices, such as sensors, and update the model in real-time. In the interactive television example, the CORBA remoting mechanism can be used by the server receiving poll results from set-top boxes to update the model as results arrive. The generated remoting mechanism also provides a publisher/subscriber framework that allows clients to receive events as the model changes or to update the model by publishing events to specific topics. A menu-item is available for starting and stopping the remoting infrastructure.

The generated modeling tool includes a basic remote collaboration mechanism. A modeler can subscribe to any remotely accessible GEMS model. As the remote model is updated, the client model will receive and add these updates to itself. A single client model can subscribe to multiple remote models and perform remote model aggregation into a single model on the client. In the interactive television scenario, this could be used to subscribe to polling results from multiple poll result streams. This remote model aggregation architecture is shown in Figure 3.

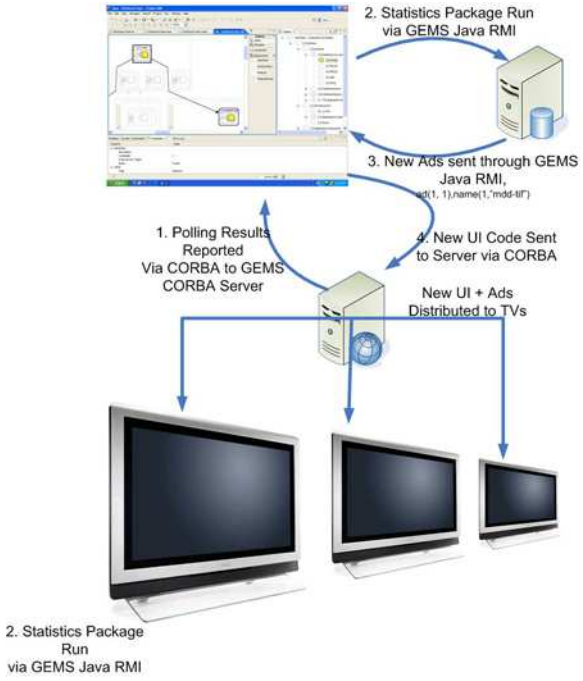

Figure 3: A Remoting Architecture in GEMS

3.6 GEMS Modeling Guidance and Constraint Solving

Constraint checking is an integral part of domain-specific modeling. Constraint checking allows developers to catch errors that typically would not be discernible in source artifacts. The errors that are caught can rely on complex sets of domain-specific information available in the model.

Although constraint checking can improve solution correctness, most tools focus on constraint checking and not reasoning with the constraints. Often, models are so large or the constraints so complex that it is infeasibe or extremely tedious to manually produce a correct solution [4]. In these cases, automation is needed to relieve the developer of performing complex and repetitive modeling tasks.

GEMS provides two types of modeling guidance. First, when a modeler is attempting to modify a model by creating a connection or containment relationship, a constraint solver can be executed to deduce the valid model elements that may participate as the endpoint of the relationship. Once the valid endpoints are deduced, visual queues are used to show the modeler the correct endpoints.

Second, GEMS provides the ability to create constraint solver-driven batch processes. These batch processes take a global constraint as input and derive a valid set of model elements that satisfies the constraint. The batch process then creates a series of connections, containment relationships, or attribute values for the model based on the constraint solver's results. An example batch process developed to perform constraint-based deployment of menus to deployment servers is shown in Figure 4.

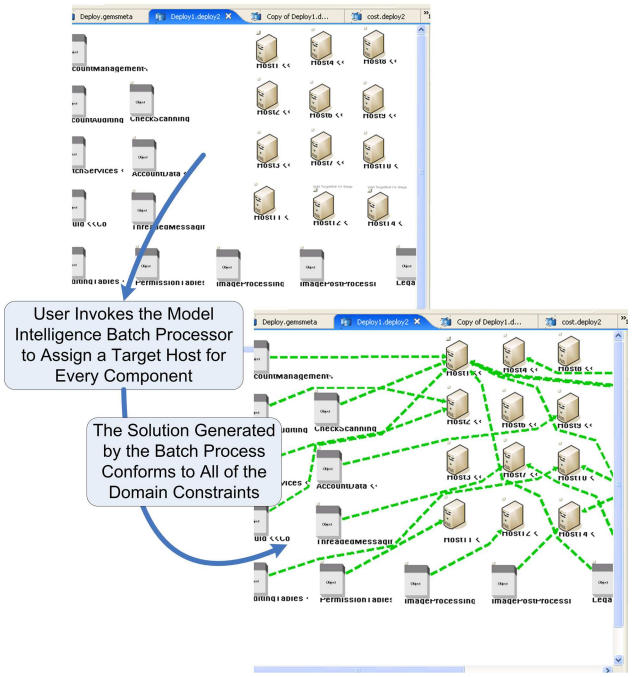

Figure 4: A GEMS Constraint Solver-Driven Batch Process

Each constraint in GEMS is associated with a model element or relationship between two model elements. Constraints can be developed using OCL, Prolog, and Java. OCL and Prolog constraints can be used by GEMS modeling guidance. Constraints can be added globally to all instances of a particular modeling language or to a specific model instance. Changes to constraints on individual model instances become active immediately.

After an OCL or Prolog constraint is added to the model, it is automatically indexed by the GEMS modeling guidance framework. To add a constraint, a user right-clicks and accesses the constraint set modification menu. From there, new constraints can be applied to arbitrary types and relationships.

For the interactive television application, a solver-guided batch process can be created to assign each menu to a set-top box that can support it. Each menu has an attribute called, OSVersion, which specifies the required version of the interactive television software

on the target set-top box. The following Prolog constraint checks to ensure that the set-top box has both the required OS version and has a display with sufficient height to accomodate the menu:

```
is_a_valid_menu_settopbox(Menu, TVBox) :-
self_version(Menu, MenuRequiredVersion),
self_version(TVBox, TVVersion),
TVVersion >= MenuRequiredVersion,
self_display_height(TVBox,TH),
self_menu_height(Menu,MH),
TH >= MH.
```

To create a deployment solver for the menus, the domain expert creates one further Prolog rule:

```
assign_all_menu_settopbox(Result).
```

This rule tells GEMS to invoke the Prolog constraint solver, find a valid set-top box for each menu, and create a connection from the menu to the box.

4. CONCLUSION

GEMS focuses on allowing developers to create Eclipse-based modeling tools without writing plug-in descriptors, GMF mapping files, GEF code, or EMF code. Furthermore, GEMS provides facilities to support external specification of visualization details so that they can be managed by graphic designers and other individuals not experienced with complex GUI coding. All modeling tools developed in GEMS can expose their models remotely via CORBA and Java RMI as well as receive remote changes. Finally, GEMS provides a constraint solver infrastructure to help guide modelers through complex modeling changes. GEMS is an opensource project available from: http://www.sf.net/projects/gems.

5. REFERENCES

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[2] S. Kelly. Interactive television example, 2007.

[3] A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle, and P. Volgyesi. The Generic Modeling Environment. *Workshop on Intelligent Signal Processing, Budapest, Hungary, May*, 17, 2001.

[4] A. Nechypurenko, E. Wuchner, J. White, and D. C. Schmidt. Application of Aspect-based Modeling and Weaving for Complexity Reduction in the Development of Automotive Distributed Realtime Embedded System. In *Proceedings of the Sixth International Conference on Aspect-Oriented Software Development*, Vancouver, British Columbia, March 2007.

[5] J. Sztipanovits and G. Karsai. Model-integrated computing. *Computer*, 30(4):110–111, 1997.