A Model-Driven Approach for Generating Embedded Robot Navigation Control Software

Bina Shah, Rachael Dennison, Jeff Gray
Department of Computer and Information Sciences
University of Alabama at Birmingham
Birmingham, AL 35294 USA
{iambina, raeanne, jgg}@uab.edu

ABSTRACT

Real-time embedded systems are time-critical systems that are hard to implement as compared to traditional commercial software, due to the large number of conflicting requirements. This paper describes undergraduate research into the use of advanced modeling techniques to improve the development of embedded systems. In particular, we have developed domain-specific models that describe the configuration and layout of a hazardous environment, which is symbolically represented as an area contaminated with hazardous materials (e.g., land mines), as well as objects to be rescued (e.g., babies). The motivation is to model a disaster site that is too dangerous for humans to search for survivors. From the visual model specifications, model interpreters will generate the embedded code that will control two LEGO Mindstorms robots. The mission of the robots is to traverse the hostile terrain and rescue the surviving babies. The modeling environment and generative techniques are described.

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming Environments – *Integrated Environments.*

General Terms

Design, Languages

Keywords

Model Integrated Computing, LEGO Mindstorms, generative programming

1. INTRODUCTION

The majority of the total global computational cycles today are spent on controlling real-time and embedded systems, including cell phones, automobile engines and brakes, chemical factories, and avionics applications. In fact, it has been reported that over 90 percent of all of the world's microprocessors are used in systems that are not "traditional" computers [5]. Physical mechanical controls are being replaced every day by software controllers [3]. The reliance on these new devices has increased the quality of our lives in many ways, yet also has created a serious dependence on technology. The fact is that such systems are often very hard to design and deploy compared to traditional commercial software because there are many more conflicting requirements in embedded systems. For example, the weight and size of embedded systems are much smaller, and they often have limited power requirements and a smaller memory footprint [6]. This has resulted in a set of specialized techniques, resembling a "black art," by experts who design such systems.

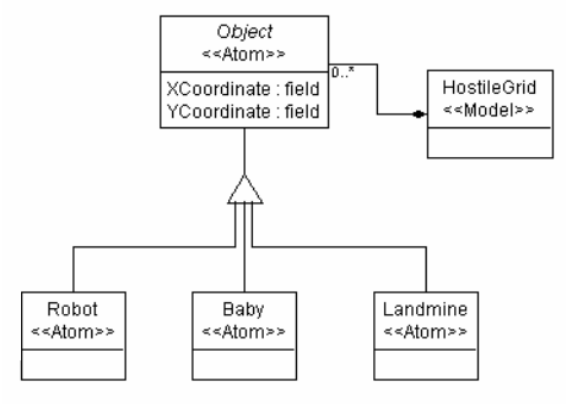
This paper describes undergraduate research into the use of advanced modeling techniques to improve the development of embedded systems. The motivating problem for the research was the realization that hard-coded software for real-time embedded robotics control systems requires manual adaptation for each new configuration. The goal of the project is to synthesize robot control software from high-level models that depict configuration of a hostile environment containing robots, landmines, and lost babies.

The approach that was investigated involves the use of a meta-configurable modeling tool called the Generic Modeling Environment (GME), developed at Vanderbilt University [2]. From within the GME, a meta-model was created that represents the hostile domain. In addition to the meta-model, a code generator was constructed that translates model information into robot control software. The code generator has deep knowledge of the robot and navigation planning.

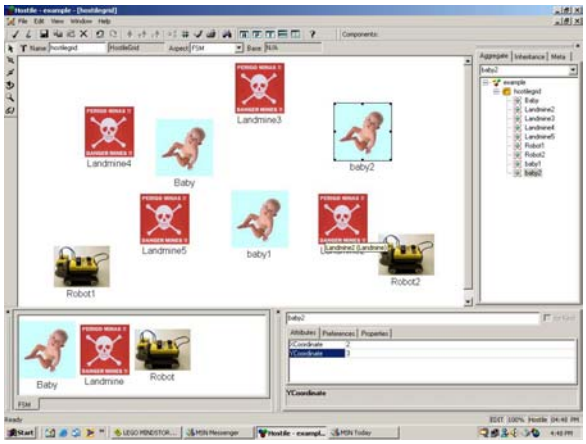
The next section provides an overview of the meta-modeling environment. Section three offers a discussion of constraints that are specified to limit the types of models that can be constructed. In Section 4, topics concerning code generators are covered within the context of the model interpreter that was constructed to generate the robot control code. The conclusion offers a summary of the research project.

2. THE "HOSTILE ENVIRONMENT" META-MODEL

In the GME, a meta-model is created for each domain that is to be modeled. The meta-model provides a specification of all of the entities that exist within the domain and their associations. The meta-model is constructed using the Unified Modeling Language (UML) and the Object Constraint Language (OCL). The top part of Figure 1 shows the simple meta-model that represents the hostile domain. This metamodel contains entities representing the major actors in the domain, such as the landmines, babies, and robots. Each attribute contains an X and Y coordinate that represents the entity location within the physical space modeled in the tool. Because each entity has these coordinates, a new modeling type was created to generalize this fact. Otherwise, each entity would have to contain its own attributes.



a) The "Hostile" Metamodel



b) An Instance Model

Figure 1. The Hostile Environment Metamodel (a) and Instance Model (b)

The bottom part of Figure 1 illustrates a specific example that was created as an instance of the metamodel. In this particular instance, there are four landmines and three babies, in addition to the two robots. It is not shown in Figure 1a, but there is provision within the GME to specify visualization icons for each metamodel entity. This permits visual clues in the instance models, as evidenced by the connotations suggested by the icons of Figure 1b.

The contribution of this part of the project provided an easy to use modeling environment for specifying the configuration of the hostile environment.

3. CONSTRAINING THE META-MODEL

There are some things that need to be specified in the metamodel, but cannot be captured with notations like that of Figure 1a. Some examples of things that need to be specified in the metamodel are listed in the left of Figure 2. For instance, it is not possible in UML class diagrams to indicate that the coordinates of entities must be unique. In order to specify these constraints, the OCL is used in conjunction with the UML. As shown in Figure 2, there are six constraints that have been specified in the hostile metamodel. These constraints can be as simple as stating that the X and Y coordinates cannot be less than 0. Other constraints, like the one that ensures unique coordinates, can be much more involved. For example, the coordinate uniqueness constraint could be determined by the following:

1. Return the number of babies, landmines, and robots with given X and Y coordinates.

2. If the number > 1, the X and Y coordinate pair is not unique.

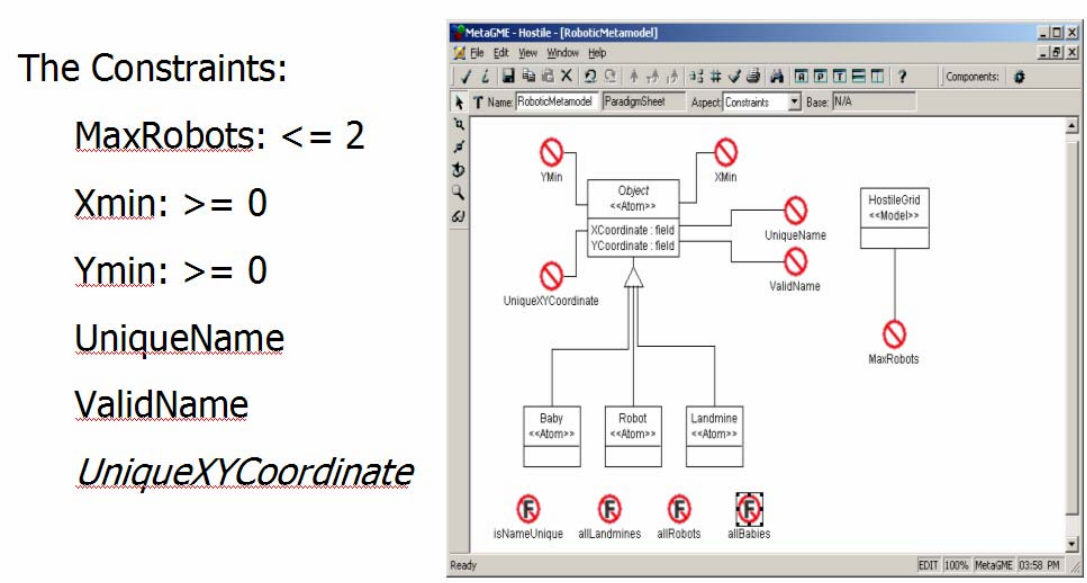


Figure 2. Constraints of the Hostile Metamodel

Alternatively, the check for uniqueness can be specified in the OCL as:

```
let count = project.allRobots(self.XCoordinate,
                              self.YCoordinate) +
            project.allBabies (self.XCoordinate,
                               self.YCoordinate) +
         project.allLandmines(self.XCoordinate,
                              self.YCoordinate) in
    if (count <= 1) then
        true
    else
        false
    endif
```

As the instance model is created, the constraint checker interprets all of the associated constraints and reports if a violation is encountered. The interpretation of metamodel constraints helps to ensure a "correct by construction" approach toward well-formed instance models.

4. SYNTHESIZING ROBOT NAVIGATION CODE

Consider the situation where manual techniques are used to program the robots for the hostile environment. For each new adaptation that is made to the configuration of the environment, new control software must be downloaded into the robot. If the control software is constructed manually, this could result in much time being spent in generating new software to conform to the new situation.

Model-driven techniques offer a better alternative to the ad-hoc manual approach just described. With a model-driven approach, a user can quickly reconfigure the hostile environment by manipulating the modeling abstractions in the tool. Then, a model interpreter, or code generator, simply can synthesize automatically the code that is needed. Thus, much time can be saved from the automatic generation of code from configuration models.

The benefit of model-driven synthesis is realized by the intelligence that is built into the code generator. The interpreter can capture much of the experiential knowledge that was traditionally used in the manual ad-hoc approach. Once this knowledge is embedded into the interpreter, then it can be reused in many different situations.

In the GME, the internal structure of the model can be accessed from API calls. The internal structure of the model can be traversed in the same way that a compiler navigates across a parse tree. The underlying structure of the model can be synthesized into numerous artifacts, such as input to analysis tools and even pure code generation. The following code illustrates the API calls used to construct a model interpreter. The code snippet shown below simply shows how access to the root model is obtained, as well as how attribute values are retireived.

```
//Get the hostileGrid model
const CBuilderAtomList allRobots =
            hostileDiagram->GetAtoms("Robot");
pos1 = allRobots->GetHeadPosition();
CBuilderAtom *Robot = allRobots->GetNext(pos1);
…
//obtain the robot's (X,Y) coordinates
int RobotX, RobotY;
Robot->GetAttribute("XCoordinate", RobotX);
```

```
Robot->GetAttribute("YCoordinate", RobotY);
```

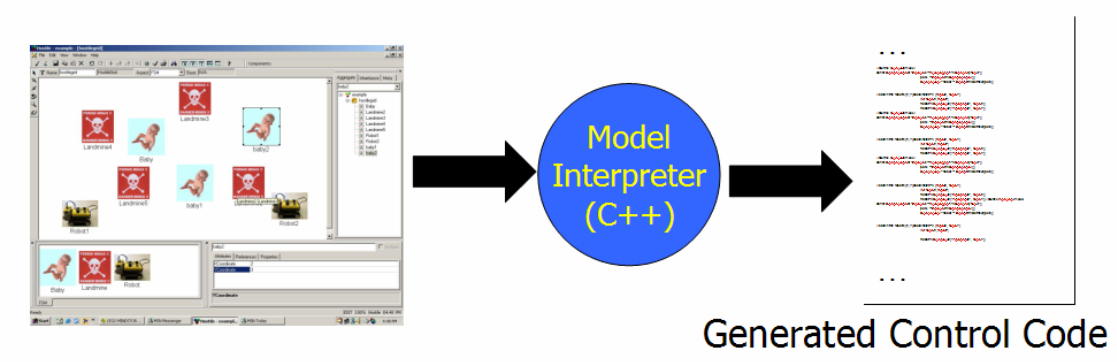


Figure 3. Synthesizing Robot Control Code from High-Level Models

The GME allows the model interpreter to be an extensible part of the modeling environment through a plug-in architecture. In such a case, a specialized icon appears on the GME toolbar representing an interpreter for a particular domain. Figure 3 shows the process of invoking a model interpreter to generate LEGO Mindstorms code to control a robot.

5. CONCLUSION

This paper describes a modeling environment that was created to symbolically represent an area infected with land-mines, or a disaster site where it is too dangerous for humans to search for survivors. From this modeling environment, model interpreters were developed to generate the embedded code that will be downloaded into a LEGO Mindstorms robot. The code that is generated assists robots in maneuvering around the dangerous obstacles. Thus, high-level models of the terrain are all that are needed to drive the navigation of the robot. Changes to the models can represent different configurations of the hostile environment and produce different code for the robot. When a new environment is encountered, all that is needed are modifications to the models. This saves time in development because low-level code does not need to be written by hand; it is generated from the models (i.e., changes to the models result in representative changes to the generated embedded code to control the robot) [4]. The intelligence of the specialized techniques for engineering such an embedded system is built into the code generators, rather than residing in the minds of embedded domain engineers.

More information about the project can be found at http://www.gray-area/org/Research/CREW

6. ACKNOWLEDGMENTS

Summer support for this research was sponsored by the NSF-ITR led Summer Internship Program in Hybrid and Embedded Software Research (SIPHER) at Vanderbilt University. Fall 2004/Spring 2004 support is sponsored by the Computing Research Association's special program for Collaborative Research Experience for Women (CRA-CREW).

7. REFERENCES

[1] Bagnall, B., Core LEGO MINDSTORMS Programming: Unleash the Power of the Java Platform, Prentice Hall PTR; 2002.

[2] Lédeczi, A., A. Bakay, M. Maroti, P. Volgyesi, G. Nordstrom, J. Sprinkle, and G. Karsai, "Composing Domain-Specific Design Environments," *IEEE Computer*, November 2001, pp. 44-51.

[3] Lee, E., "What's Ahead for Embedded Software?" *IEEE Computer*, September 2000, pp. 18-26.

[4] Neema, S., T. Bapty, J. Gray, and A. Gokhale, "Generators for Synthesis of QoS Adaptation in Distributed Real-Time Embedded Systems," *First ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering (GPCE '02)*, Pittsburgh, PA, October 6-8, 2002, pp. 236-251.

[5] Santo, B., "Embedded Battle Royale," *IEEE Spectrum*, December 2001.

[6] Simon, D., *An Embedded Software Primer*, Addison-Wesley, 1999.