Gravity: An Object-Oriented Framework for Hardware/Software Tool Integration*

Murali Rangarajan, John Penix, Perry Alexander, and Philip A. Wilsey
Department of ECECS, PO Box 210030, Cincinnati, OH 45221–0030

Abstract

Systems development is becoming more and more complex. It consists of many phases such as high-level design, low-level design, design validation, simulation, and so on. Throughout the design process, a variety of tools are used to assist and automate the various phases. These tools are often incompatible, forcing the design phases to be performed in isolation. This limits the iterative possibilities of the design process and forces the designer to make commitments, such as hardware or software implementation, early in the design process. The Gravity system is a tool integration framework designed to provide continuity throughout the design process. It is intended to be used to build domain-specific design environments. Gravity provides an easy way to construct a common store of objects involved a design, and to apply various tools to these objects. New languages and tools can be easily integrated within the framework. The system makes no distinction between hardware and software modules, thereby facilitating hardware/software co-design. Gravity is implemented in JAVA, providing portability, networking capability and graphical extensibility.

1 Introduction

As software and hardware become progressively complex, their design and validation are also becoming highly complicated. Design of present-day systems can no longer be performed with the simple tools that were sufficient a few years back. Typical hardware implementations use millions of transistors and software implementations involve hundreds of thousands of lines of code. In such large systems, it is very difficult to find bugs. Design errors found during the implementation phase or later are very costly to fix. Hence the need for automated tools to design and verify these systems.

The design of complex systems involves a large number of inter-related tasks. First, a high-level design of the proposed system is done, which is progressively split into smaller modules. Then the modules to be implemented in hardware and software are identified. Once some aspect of the system is designed, the design is validated. This can be done by formal analysis or simulation of the design. Most often, the results of validation indicate that changes must be made in the design. This results in an iterative design process.

In most cases, the tools used are specific to an individual design task. Therefore, providing support for the entire design process requires integrating a collection of tools. In spite of this need for tool integration, vendors have not done enough to make it an easy task. Therefore, providing support for the entire design process requires integrating a collection of tools. In spite of this need for tool integration, vendors have not done enough to make it an easy process [4]. They continue to use different standards, proprietary formats of storing information, and implicit assumptions about the environment in which their software is going to function. This has resulted in a number of problems in making these tools work together. The brunt of this problem is being faced by those trying to create development environments that integrate many different tools.

Gravity is a tool integration framework designed to provide continuity throughout the design process. It is intended to be used to build domain-specific design environments. Gravity provides an easy way to have a common store of objects involved in the design, and to associate various tools to these objects. It consists of a set of core classes that the user can specialize to suit his needs. Gravity assumes very little about the system being designed except that the system is being developed in a modular manner, which is the case for most present day systems.

The remainder of this paper is organized as follows. Section 2 provides an overview of Gravity and how it is envisaged that the system will be used. Section 3 explains the design considerations for Gravity. Sections 4 and 5 describe the design and subsequent implementation of gravity. Section 5 presents a brief review of related work. Sections 7 and 8 discusses the current status and future direction of our work. Lastly, Section 9 contains some concluding remarks.

2 Overview of Gravity

The process of building environments with the help of Gravity and using these environments is shown in the Figure 1. The Gravity system provides a framework that can be used by a tool builder to build a design environment by integrating a number of different tools. The tool builder also integrates a front-end (such as a graphical user interface (GUI)) with the Gravity system to make it easier to use. The design environment so built is used by the tool user to design systems.

A design is constructed by using three types of design objects: components, ports and connections. A component encapsulates...
some aspect of the system’s functionality. The inputs and outputs of a component are ports. Interactions between the various components are represented by connections between their ports. Each time a design object is used in a design, some new information must be stored. This information varies from design to design.

Thus, this conceptual model of systems is general enough to support hardware/software co-design. The distinction between hardware and software modules is not necessary and can be determined when the time is appropriate. In addition, both hardware and software solutions can be designed and their tradeoffs considered.

Gravity was meant to perform generic tool integration and as such had certain specific conditions that it had to satisfy. First of all it must be easily extendable by the tool builder. Only then can the tool builder effectively integrate all the tools required. Next, each tool builder might want to have a front end that is natural to use for the specific domain for which he is creating an integrated design environment. Therefore, it is necessary to design Gravity independent of any front end. Also, it is not known whether the tools being integrated would be used for the design of hardware, software or combined hardware/software systems. Hence, Gravity must be sufficiently generic to handle all these cases. Another important consideration is that the Gravity system must be independent of the storage mechanism. This gives the tool builder the freedom to use the storage mechanism that is best suited for the environment being developed by the tool builder. Finally, the tool should be easily portable across platforms and operating environments.

3 The Implementation of Gravity

The object diagram of the Gravity system is shown in Figure 2. Component, port and connection are all subclasses of design_object. Each design_object has a corresponding use class. The use classes contain information specific to a particular instance of the corresponding component, port or connection class. The design_object classes and their use classes together model a class-instance relationship.

Each component has one or more representations. A representation can be any information regarding the component such as a behavioral specification, a source code representation or a composite picture of its sub-components. Each representation has a language associated with it and each language has a corresponding tool set. The tools determine the actions that can be taken on the representations of a component. A component can contain more than one representation of the same language. By extending from these classes, a tool builder can configure the environment to the needs of the user. Therefore, the Gravity system is easily extensible.

A library is a collection of related components. The use of libraries makes it easier to keep track of the many components that can be used in a design. There can be many such libraries and a given component can belong to more than one library.

Gravity can be used with any front end of the tool builder’s choice. The diagram_object class provides a way to attach any front end to gravity. It allows the use classes to know that they have a graphical representation. However, the form of that graphical object is not constrained in any way. This makes it very easy to configure any front end to work with the Gravity system.

For a number of reasons, the Gravity system is implemented in the JAVA [3]. One of the chief reasons for the selection of Java is portability. A program written in JAVA runs on any platform that implements the JAVA Virtual Machine. Another advantage in using JAVA is automatic garbage collection. This relieves the programmer of the problematic and tedious task of memory management. JAVA eliminates some error-prone features in other languages such as pointer manipulations, thereby making programs more robust. JAVA also provides libraries for tasks such as developing user interfaces, interfacing with the network. And finally, another benefit of using JAVA is that it is designed to support sending applications over the net. This provides the possibility of tools being dynamically delivered and integrated within an environment.

4 Example: Front end Integration

The Block Diagram Editor (BDE) is a front end developed by us to demonstrate the effectiveness of the design of Gravity. It allows the designer to manipulate the design information in the data store just by manipulating blocks on the screen. The diagrams formed using these blocks represent the current design.
The Gravity-BDE supports both top-down and bottom-up design methodologies. For top-down design, the user proceeds as follows. The user first creates a block for the component he wants to design, with trigons (triangles) for its ports. He then 'explodes' the block to bring up a blank canvas in which he can design the next level of the component using new blocks or blocks representing components in a library. In bottom-up design, the designer first creates new sub-components, if needed, and adds them to the libraries. Then the designer uses the components to construct the new system at any level of abstraction as desired. Then, the design is 'imploded' into a new block, which can be added to a library. A typical design session would include instances of both types of design.

The object model of the BDE is shown in the Figure 3. The classes above the thick horizontal line are Gravity classes and those below that line are the BDE classes.

The BDE is represented by the BDE class. The BDE too can have many different libraries. These libraries are managed by the BDELibraryManager class. A library is represented by the BDElibrary class that provides a graphical representation for a Gravity library.

The BDE Canvas class represents all the designs currently being used. All the other designs that are not being used are represented by diagrams. The BDE Canvas is a diagram drawn on a canvas or drawing surface.

The BDE has the classes box, trigon and net corresponding to the classes component, port and connection respectively in Gravity. Just as components, ports and connections are all design objects, the boxes, trigons and nets are all diagram objects. By having such a correspondence between the objects in Gravity and the objects in BDE, it is possible to modify the contents of the data store just by modifying the diagrams on the screen. The BDE has another class of type diagram object, namely: anchor. This class is used to represent the inputs and outputs of the current design as a whole.

The interface representation, which is a subclass of the representation class of Gravity, represents the whole of the current design by a single box, with the anchors in the current design as trigons on the box.

BDE included the ability to load classes dynamically over the net. This means that the languages and tools need not be stored by the user but rather got from the tool builder when needed. This also ensures that the user is working with the latest version of the classes.

No storage system was used in our experimental implementation. Since the Gravity system must be able to work with any storage mechanism, what is important is that the objects required must be in memory when required. It is fairly straightforward to implement a mechanism that loads the required object into memory when it is needed by gravity. Therefore, Gravity need not be aware of the storage mechanism.

A screen shot of the BDE is provided in Figure 4. The BDE window is divided into three sections. The central portion is the main drawing canvas. To the left is the current library that displays the components in the current BDElibrary. A component from the current library can be added to the current design by dragging that component from the library and dropping it at the required place in the design. The user can switch between the various libraries using a pull down menu at the top. The third section is a set of buttons at the top that lets the user do common tasks such as selecting a block prior to invoking some action on it, moving the blocks around to make the diagram easier to comprehend, creating nets to connect the various blocks together, and so on.
5 Example: Tool Integration

We now present an example of a design environment construction activity using Gravity-BDE. An overview of the environment is shown in Figure 5. At the center of the Gravity system is the collection of design objects used for the design activity, represented by a pair of horizontal lines. The various tools such as configuration management tools, simulation tools, synthesis tools, library management tools and analysis tools can be interfaced to the gravity system. These tools act directly on the information stored in the Gravity objects. The solid lines in the figure depict the flow of stored data between the various tools and the collection of Gravity objects. The designer interacts with a front end, through which he performs various actions on the information stored in the Gravity objects. The invocation of an action results in some tool being applied to some Gravity object. This invocation of tools from the front end is depicted by the dashed lines in the figure.

This example uses bottom-up design methodology and is for a hardware component. In the first phase, the user designs the system by putting components together and creating intermediate components that are used again with other intermediate components. For each of these components, the user creates a VHDL [1, 7] representation. After the design is complete, the user verifies the design by simulating it. First, the user uses the SAVANT parser [10] to obtain the C++ classes corresponding to the VHDL representations. Then, the user runs the TyVIS [11] simulator to simulate the design.

Let us now look at the new classes that must be created by the tool builder to let the user do the design the way it is described above. The new classes are shown in the Figure 6. The tool builder must create new VHDL and C++ classes that are sub-classes of the language class. Then, the builder must add the editor and SAVANT parser tools to the VHDL class, and the TyVIS simulation kernel to the C++ class. These new tools are sub-classes of the tool class. The front end that the tool builder integrates with the Gravity system must provide a library of components and must support bottom-up design. Also, the front end must provide the ability to invoke tools on the objects used in the design. All these facilities are available in the BDE and so, it can be used for this design activity. This minimal functionality is sufficient to support the described design activities.

6 Related Work

The problems in designing huge and complex hardware and software were realized many years back, and since then, considerable work has been done attempting to make the design process easier. One of the directions in which work has been done is the methodologies used for the design of systems. This direction has resulted in such methodologies as object oriented design [9]. The other direction has been towards more tool support for the design process. This approach has resulted in a plethora of tools such as Computer Aided Software Engineering tools, Computer Aided Design tools, simulation tools, Integrated Design Environments, and so on. As is natural with any system, there are features in each of these that a user likes, and features that is not quite what the user wants. The user would typically like a mix of the tools to satisfy his requirements. Hence, in recent times, a great need for integrating these various tools has been felt.

The Aesops system [4] is very similar to what we have designed. The Aesop system is a “style-oriented architectural devel-
Figure 4. A screen shot of the BDE front end
**GRAVITY**

![GRAVITY Diagram]

**BDE**

![BDE Diagram]

Figure 5. Overview of the design environment

Figure 6. New classes required for tool integration
The Aesop system has some basic differences with the Gravity system described herein. First, Aesop generates development environments that are tailored to particular classes of applications, whereas Gravity produces more general purpose environments. This occurs because the Aesop system makes more assumptions to allow more automation. Next, the tools used in Aesop are highly integrated, whereas the tools in Gravity can be integrated to the extent desired by the tool builder. Finally, while Aesop uses an object-oriented database, Gravity does not make any assumptions about the type of data storage mechanism used.

With a view of sharing data between two systems, Linden and Verkamo [6] have proposed a transformation technique that transforms the data in the format of one system to that of the other. But one of the major drawbacks of this approach is that separate transformation routines must be written for each pair of tools to be integrated. Thus, this does not provide a general solution.

More recently, Cutkosky et al [2] have tried to use the web for integrating the tasks involved in the design and development of systems. In this project, the work was divided into smaller tasks based upon the domain they fall into, such as electronics, mechanical and optical. Each task was distributed to different groups for further design and implementation. The web pages were used as design documents that can be referred to by the whole project team and the web itself was used for synchronous and asynchronous communication among the members of the project teams.

7 Current Status

The Gravity system is still under development. The basic framework as shown in the object model (Figure 2) is in place. Also, a front end (BDE) was developed to test the effectiveness of our design. The languages used with that system included VHDL, VSPEC and LSL. The tools included the emacs editor, VSPEC parser, the Larch Prover and the PVS system. The BDE also included a set of libraries of components. The user can drag-and-drop a component from any library, connect it to other components, invoke emacs editor and type in the VSPEC representation and parse this representation. The components can even be loaded dynamically from anywhere on the net.

8 Future Work

A few important goals have been planned for the near future. First is the expansion of the basic framework to facilitate analysis activities. The user must be able to store test plans, test configurations, test results, and so on. within the framework of Gravity. Design on this work has already started and the implementation will be done soon. The other activity planned is the integration of Gravity with front ends developed by third parties so as to make sure that the integration is made as simple as possible. Work on integrating Gravity with the UCI Graph Editing Framework [8] is currently being undertaken. Gravity has been integrated with many tools at the University of Cincinnati. Finally, we plan to attach the data storage and retrieval mechanisms to JDBC [5]. This will make it even simpler to use any JDBC compliant data store mechanism with the Gravity system.

9 Conclusion

Designing any medium to large scale system is a very complex task. A number of tools are usually used to perform the various sub-tasks. These tools are usually incompatible, forcing the various sub-tasks to be performed in complete isolation from other sub-tasks. This limits the possibilities of designs and forces the designer to make decisions such as hardware or software implementation very early in the design process. By providing a common set of objects and a uniform treatment of all types objects, Gravity allows development of environments which integrate tools for different languages. Gravity is not yet a complete system, but it’s advantages are already apparent.

References