Dynamically Reconfigurable Monitoring in Large Scale Real-Time Embedded Systems

Shikha Ahuja, Di Yao, Sandeep Neema, Ted Bapty, Shweta Shetty, and Steven G. Nordstrom
Institute for Software Integrated Systems
Vanderbilt University
s*hikha@isis.vanderbilt.edu*

Abstract

Many large scale real time distributed embedded systems are computation and communication intensive. Some examples of such systems are the data acquisition for high energy experiments performed in Fermi Lab (e.g. the BTeV experiment). This data system contains thousands of processors involved in performing real-time event filtering of particle collisions. A tremendous amount of message passing takes place continuously between these nodes. The physicists need to monitor such a large scale system to ensure its correct operation. Configurable user interfaces will enable the physicist to dynamically view data as well as error conditions in ways that aid analysis as well as enable them to configure and control the state of the system.

This paper presents a language that enables the configuration of the user interfaces dynamically, supporting large-scale system monitoring and control. These configurable user interfaces are implemented using Model Integrated Computing [1]. The domain specific Modeling language (DSML) and associated tools developed can be used to generate software for a variety of runtime platforms. Currently, the target environment for the user interfaces is Matlab. To transport data from many sources throughout the distributed system to many potential consumers, a publish-subscribe mechanism called Elvin is used in the system and is briefly described in the paper, along with its role in the system monitoring and control. The paper ends with a brief discussion emphasizing the use of this domain specific modeling language in the BTeV system.

1. Introduction

Software development for embedded systems is challenging as these systems are not just required to be computationally and functionally correct but are also heavily time bound. In addition, these systems are required to be robust enough to tolerate failures. A reasonable behavior is expected from these systems even under fault scenarios.

Monitoring of these systems is essential to ensure their correct functioning. However, monitoring different aspects of the system at different times would require several variations of the user interfaces, which are difficult to maintain. In order to meet the requirements of continuously changing user interfaces and in order to bridge the gap between the developers and the users of the system, the need for configurable user interfaces arises. However, in order to support the evolving requirements of the system, a low level programming based approach cannot be used to efficiently achieve configurability in the design of user interfaces. Automated tools abstracting the system are required to assist the developers in managing the complexity. The tools should also be capable of synthesizing low level implementations from higher level of abstractions.

In order to address these issues, we have applied the concepts of Model Integrated Computing (MIC) [1], a methodology for developing tool driven embedded software solutions developed and refined over two decades of research at Institute for software Integrated Systems(ISIS), Vanderbilt University.

MIC is an approach to designing complex computer-based systems. Models capture system design information, environmental interactions, and other constraints on system composition. The models are composed in a customized, multi-aspect, domain-specific language. Specialized software generators, called Interpreters, traverse these models to extract design information, make decisions on system implementation, and synthesize code. MIC has been successfully applied in several domains [2][3][4][5].

These domain-specific modeling languages are defined in an environment called Generic Modeling Environment (GME) [6]. The Generic Modeling Environment developed at the Institute for Software Integrated Systems at Vanderbilt University is a configurable toolkit for creating domain-specific modeling and program synthesis environments.

Using the capabilities of Model Integrated Computing and using the Generic Modeling Environment, we are developing tools for designing dynamically reconfigurable user interfaces. Using model-based approach to developing user interfaces has been talked about for a long time now, however not much work has been done in this domain. [7] briefly talks about the various advantages in developing model based user interfaces. The rest of this paper is organized in the following manner: Section 2 provides a description of the System development environment that includes the metamodeling environment as well as example domain models. Section 3 briefly describes the data transfer

using the Elvin publish-subscribe mechanism. Section 4 describes the system generation from the models.

2. System Development Environment

The application scenarios that we at Vanderbilt are working on are the data acquisition for high energy physics experiments performed in Fermilab, in particular the BTeV experiment. The BTeV experiment [8][9] includes a trigger that encompasses several thousands of processors distributed over three level trigger architecture. Time critical event filtering algorithms that run on the trigger farms are likely to suffer from a number of failures within the software and hardware systems. It is necessary for the system to run as efficiently as possible for long periods of time with faults present. A fault management software component called Adaptive, Reconfigurable and Mobile Objects for reliability (ARMOR) [10][14][15] developed by the University of Illinois is currently used in the system to provide fault tolerance. Modeling techniques are used to facilitate customizable fault handling in the system. A modeling language has been developed using GME that enables the physicists to introduce custom self adaptive behaviors in case of failures [11][22].

Large-scale data transfer and message passing takes place both locally as well as between different nodes across the network. To transport data from many sources throughout the distributed system to many potential consumers, a publish-subscribe mechanism is used by the nodes to receive data. A specific publish subscribe mechanism called Elvin [12] is used in the system.

Real-time monitoring of data needs to be done to ensure that the system runs efficiently at all times. Configurable user interfaces will enable the physicists to dynamically view data as well as error conditions in ways that aid analysis as well as log data for comparison of results. User interfaces also enable them to dynamically configure and control the state of the system. A modeling language has been developed in GME that enables the physicists to configure user interfaces. In addition, another modeling language developed in GME enables the physicists to configure the data that is to be sent across the network. A detailed description of these languages is provided in the subsequent sections.

2.1. User Interface Modeling Environment

The language for configurable user interface modeling can be summarized as follows:

- The language should let the users create multiple user interface panels. It should hence provide a way for the users to specify these panels as well as the plots/controls that are a part of these panels. This can be depicted with containment hierarchy.
- For each of the plot/control, two primary properties have to be specified, namely its location as well as the data from outside the environment that is to be plotted/controlled. This can be depicted as two different aspects of the plot/control objects.
- The user also needs to specify the properties of all the objects in the environment. This can be depicted as attributes of each of the objects.
- The environment should be flexible enough to let the users create customizable plots that show certain properties e.g. Strip charts etc. This can be achieved by letting the users send the raw data to user-defined computation blocks where certain computations may be performed on the data before sending the data to the plots. This can be depicted as connections/associations.
- Once the user has modeled the system using the language provided, the tool should be capable of generating the code necessary to create the user interface.
- The tool should be capable of generating software for a variety of run-time platforms.

The Generic Modeling Environment (GME) supports all the features defined above i.e. hierarchy, associations, containment as well as multiple aspects.

2.2. MetaModeling Environment

The first step in the development of a new language is to specify the syntax and semantics of the language in the GME metamodeling environment.

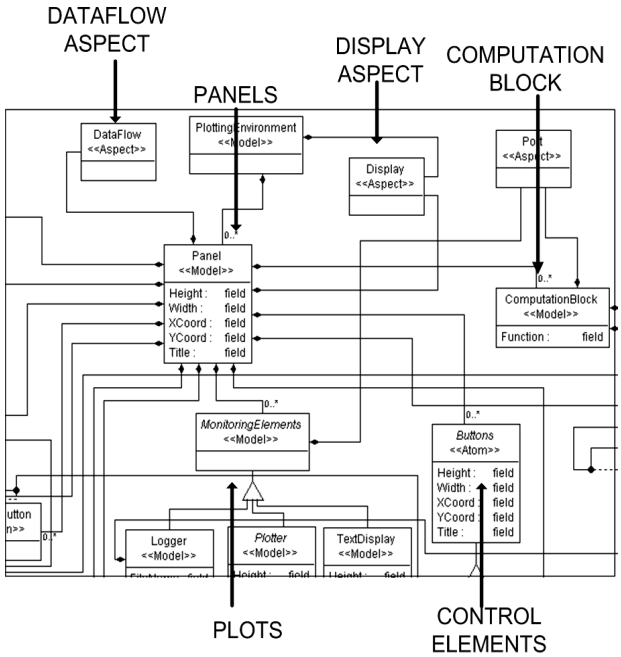

Figure 1. Metamodel describing the language

The metamodel which defines the syntax as well as the semantics of the language uses the UML based notations to describe the different associations and interactions between the components.

Figure 1 shows the metamodeling language modeled in GME. As can be seen from the figure, the panel can contain different types of plots as well as controls and the Plotting Environment can contain more than one of these panels. The metamodel shows two aspects – Display aspect as well as the Dataflow aspect. Each of the plots as well as controls has these two aspects. In the dataflow aspect, the user also needs to specify the data that is to be plotted or controlled. As shown in figure 1 and in figure 2, the environment provides Computation blocks that can be connected to the data to perform computations on the data before sending the data to the plots. These Computation blocks enable the users to create custom plots like strip charts etc which do not come as a part of the package.

2.3 Domain Modeling

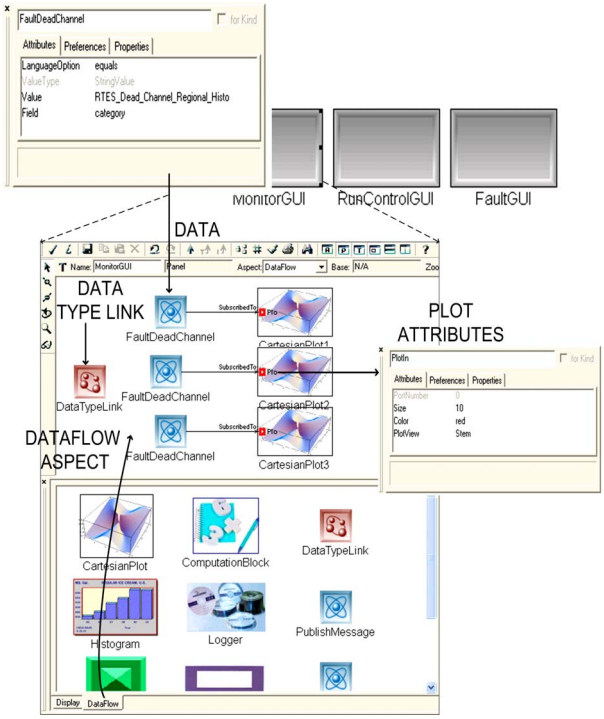

Figure 2. A simple example of a modeled user interface

A simple example of a user interface model is shown in figure 2. This example shows three panels called MonitorGUI, RunControlGUI and FaultGUI. These panels are simplified examples of three user interfaces panels required in the BTeV system.

In this simplified example, RunControlGUI is used to start and stop the system application. MonitorGUI is used to plot different data/messages in the system, as configured by the user. FaultGUI notifies the operator of different faults that occur in the system. As an example, it notifies the user in case the memory usage exceeds a certain threshold value.

In the model above, MonitorGUI contains three plots. The figure shows the dataflow environment aspect. This aspect shows the data that is to be plotted in each of the plots. The structural information for the user interface is obtained from the display aspect. The properties for each of the plots are shown as plot attributes.

3. Data transfer using Elvin

As mentioned previously, due to the presence of thousands of distributed processors, large-scale data transfer and message passing takes place both locally as well between different nodes across the network. A publish-subscribe mechanism is used to transfer data between the nodes. A specific publish-subscribe mechanism called Elvin is currently being used in the system. The user interface in this scenario can be a producer as well as a consumer of messages. In case of monitoring messages, the user interface acts as a consumer and in case of control messages, the user interface acts as a producer. In figure 2, the MonitorGUI subscribes to monitoring type of messages and RunControlGUI publishes start/stop messages. The Data Type Link element shown in the model above is a link to a model that defines the structure of the message. This message is modeled using another language defined in GME called the Data type Modeling Language (DTML). The data shown in the figure above are monitoring type messages and the three Cartesian plots connected to them plot these data.

3.1. Data Type Modeling Language

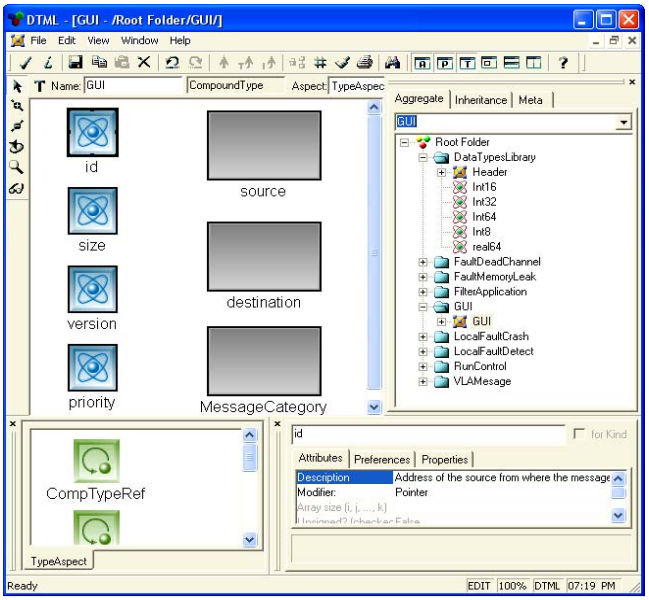

Figure 3. An example of a Data type model

The Data type Modeling Language (DTML) in the BTeV system serves the following purposes:

1. Provides communication abstraction APIs thus providing a standard way of defining the structure of data as well as a standard way of serializing and

deserializing the data using the Elvin Communication protocol

2. Since the DTML language provides an abstraction to the internal implementation details of the communication system, the communication scheme may change later without affecting the users.

An example of a domain model using the DTML language is shown in figure 3.

The Data Type Modeling language allows the specification of simple data types like floats and integers as well as composite data types such as struct, enum, union etc. Composite types can contain simple types as well as other composite types. The environment is flexible enough to support the modeling of data types supported by C programming language.

4. System Generation

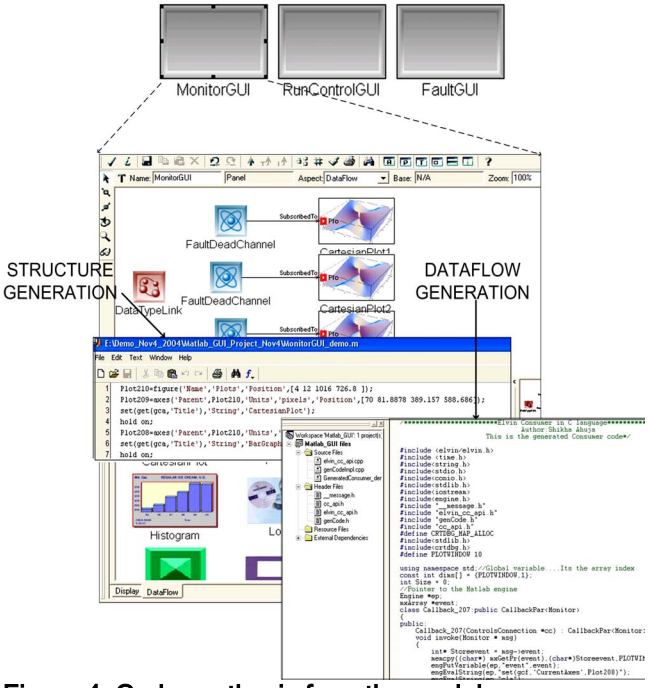

Figure 4. Code synthesis from the models

Once the user interface has been configured using the models by the user, the code for the UI needs to be generated. The models may be used to generate software for a variety of run-time platforms like Matlab, Experimental Physics and Industrial Control System (EPICS)[13], etc. Currently the target environment for the user interface is Matlab.

As shown in the figure4 below, the code synthesis process involves the generation of the following from the models:

1. *Structure* - The structure of the user interface e.g. the positioning of the various components as well the width, height of the components is a direct mapping from the GME models to the generated Matlab user interface. The code for the structural information is generated as Matlab .m files
2. *Dataflow* - The data that needs to be plotted in the model. The user interface receives data continuously. As mentioned previously, the user interface could be an Elvin consumer or an Elvin Producer. In case of Elvin consumer, once the data is received by the consumer element, it needs to be plotted in Matlab. The Matlab Component Object Model (COM) interface is used for this purpose. The code for the data flow aspect is generated in C++.

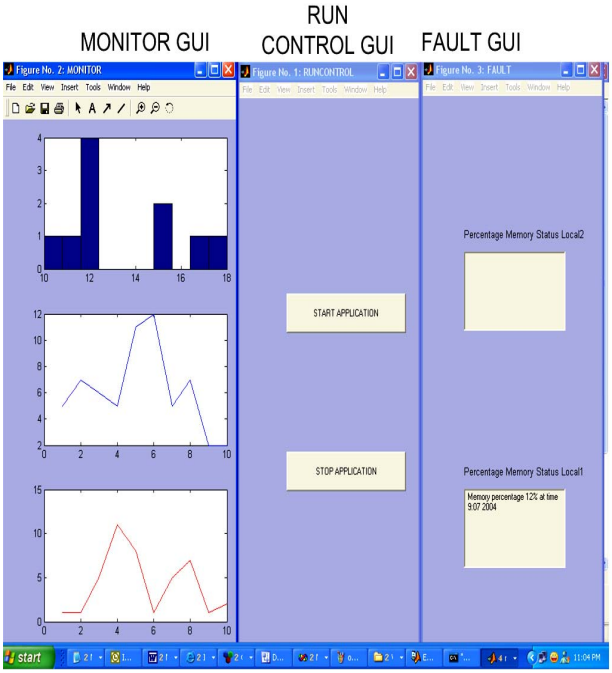

Figure 5: Generated user interface from the models

5. Related Research

Our research can be summarized as a demonstration of the use of model based user interface development technique for the development of configurable user interfaces for large scale real-time distributed embedded applications.

A wide range of model-based user interface tools have been developed often specialized to the different levels of the model [21]. These tools range from text editors to build textual specifications of models (ITS [16]), forms-based tools to create and edit model elements (Mecano [17]) and specialized graphical editors (Humanoid [18], many others). Most of these tools fall mainly in two categories:

1. Automatic Interface design tools- The primary goal for tools that fall in this category is to automate as much as possible the design and implementation of the user interface. Most tools in this category are oriented towards database

applications and produce interfaces that allow the end-users to browse the database, to edit the contents of objects, to define new objects, and to delete objects. E.g. Janus [19].

2. Specification-based tools – The tools that fall in this category do not try to automate interface design, but rather give developers convenient languages for expressing designs. Most tools in this category are oriented towards data management applications. Most business-oriented applications fall in this category, but many engineering and data visualizations applications do not, because they have interfaces whose graphical components are too complex to be expressed in their interface specification languages. E.g. ITS, Humanoid.

6. Discussion

Most of the tools presented above are oriented towards applications related to databases and data management. Some of these tools suffer from drawbacks like restraining the user from restructuring the raw information provided to them as well as their inability to show elements such as plots as pointed in [20].

The tool defined in this paper is specific to real-time applications where reconfigurability of user interfaces by domain experts is desired. In addition, the tool overcomes some of the challenges faced by traditional model based user interfaces e.g. its ability to configure plots as well as letting the user restructure raw information using customizable computation blocks.

By adding an extra layer of abstraction, there is one extra level of indirection which slightly affects the performance of the system, however, it significantly reduces the time and saves the efforts of the physicists, that would otherwise be required to change the user interface in order to monitor different aspects of the system as well as to control the state of the system at different times.

7. Conclusions

This paper presents a tool that enables dynamic reconfigurable monitoring and control in large scale real-time embedded systems and thus fulfils the demands of the users for evolving user interfaces. This is achieved through model based user interfaces. The model based approach provides higher level abstractions, thus allowing the users to configure new user interfaces dynamically without knowing the underlying implementation details.

Prototype implementations of software generators have been implemented to directly transform models into target user interfaces. These user interfaces execute under the Matlab runtime environment and interface with the BTeV prototype system.

This tool along with other modeling tools developed for the BTeV prototype system provides a framework for achieving fault-tolerance in large-scale distributed real-time embedded systems.

8. Acknowledgments

This work is supported by NSF under the ITR grant ACI-0121658. The authors also acknowledge the contribution of other RTES collaboration team members at Fermi Lab, UIUC, Pittsburgh and Syracuse University.

9. References

[1] Sztipanovits J.,Karsai G. “Model Integrated Computing”, IEEE Computer, pp. 110-112, April, 1997.

[2] Bapty T., Neema S., Scott J., Sztipanovits J., Assad S. “Model-Integrated Tools for the Design of Dynamically Reconfigurable Systems”, VLSI Design, 10, 3, pp. 281 – 306, 2000.

[3] Davis J., Scott J., Sztipanovits J., Martinez M.: Multi-Domain Surety Modeling and Analysis for High Assurance Systems, pp. 254-260, Nashville, TN, March, 1999.

[4] Karsai G., DeCaria F. “Model-Integrated On-line Problem-Solving Environment for Chemical Engineering”, IFAC Control Engineering Practice, 5.5, pp. 1-9, 1997.

[5] Misra A., Karsai G., Sztipanovits J., Ledeczi A., Moore M. “A Model-Integrated Information System for Increasing Throughput in Discrete Manufacturing”, International Conference and Workshop on Engineering of Computer Based Systems, pp 203-210, Monterey, CA, March 24, 1997.

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

[7] José A. Macías and Pablo Castells. *An EUD Approach for Making MBUI Practical.* Workshop on "Making Model-Based User Interface Design Practical: Usable and Open Methods and Tools". Intelligent User Interfaces and Computer-Aided Design of User Interfaces Conference (IUI/CADUI'2004). Funchal, Madeira Island, Portugal, 13-16 January

[8] BTeV Webpage http://www-btev.fnal.gov/

[9] http://www-btev.fnal.gov/DocDB/0021/002115/012/part-4.pdf

[10] Kalbarczyk Z., Iyer R.K., Bagchi S., Whisnant K., “Chameleon: A Software Infrastructure for Adaptive Fault Tolerance”, IEEE Transaction on Parallel and Distributed Systems, vol. 10, no. 6, pp. 560-579, June 1999.

[11] Shetty, S., S. Neema, et al. (2004). Model-based Self-Adaptive Behavior Language for Large Scale Real-Time Embedded Systems. IEEE Conference on Engineering of Computer based Systems (ECBS-2004). Brno, Czech Republic.

[12] Elvin Webpage http://elvin.dstc.edu.au/

[13] Epics Webpage http://www.aps.anl.gov/epics/

[14] Bagchi S., Srinivasan B., Whisnant K., Kalbarczyk Z., Iyer R.K., "Hierarchical Error Detection in a Software Implemented Fault Tolerance (SIFT) Environment", IEEE Transactions on Knowledge and Data Engineering, vol. 12, no. 2, pp. 203-224, 2000.

[15] Whisnant K., Kalbarczyk Z., Iyer R.K., "A System Model for Dynamically Reconfigurable Software", IBM Systems Journal, Special Issue on Autonomic Computing, vol. 42, no. 1, pp. 45-49, 2003.

[16] Wiecha, C., Bennett, W., Boies, S., Gould, J., Green, S.: ITS: *A Tool for Rapidly Developing Interactive Applications.* ACM Transactions on Information Systems, Vol.8, No. 3, 204-236 (July 1990).

[17] Puerta, A.: *The Mecano Project: Comprehensive and Integrated Support for Model-Based Interface Development. In:* Vanderdonckt J. (ed.): Proceedings of CADUI'96. Namur: Presses Universitaires de Namur: Presses Universitaires de Namur 1996(pp. 19-36).

[18] Luo, P., Szekely, P., Neches, R.: *Management of Interface Design in Humanoid.* In Ashlund S., Mullet K., Henderson A., Hollnagel E., White T. (eds.): Proceedings of INTERCHI'93. New York: ACM Press 1993 (pp. 107 – 114).

[19] Balzert, H., Hofmann, F., Kruschinski, V., Niemann, C.: *The JANUS Application Development Environment- Generating More than the User Interface.* In: Vanderdonckt J. (ed.): Proceedings of CADUI'96. Namue: Presses Universitaires de Namur 1996 (pp. 183-207).

[20] Harning, M.: *An Approach to Structured Display Design- Coping with Complexity.* In: Vanderdonckt J. (ed.): Proceedings of CADUI'96. Namur: Presses Uni versitaires de Namur 1996(pp. 121-138).

[21] P. Szekely, "Retrospective and Challenges for Model-Based Interface Development", Proc. of 3 rd Int. Workshop on Computer-Aided Design of User Interfaces CADUI'96 (Namur, 5-7 June 1996), Presses Universitaires de Namur, Namur, 1996, pp. xxi-xliv.

[22] S.Neema, Ted Bapty, S.Shetty, S.Nordstrom Developing Autonomic Fault Mitigation Systems, Special Issue on Autonomic Computing and Grids at the Journal- Engineering Application of Artificial Intelligence, Elsevier Publication, 2004.