Systems Integration of Large Scale Autonomic Systems using Multiple Domain Specific Modeling Languages

Shweta Shetty, Steven Nordstrom, Shikha Ahuja, Di Yao, Ted Bapty, Sandeep Neema
Institute of Software Integrated Systems, Vanderbilt University,
2015 Terrace Place, Nashville, TN 37235
(shweta,steve-o,shikha,dyao,bapty,sandeep)@isis.vanderbilt.edu

Abstract

Software design, development and maintenance for large scale systems has been one of the most difficult and expensive phases of the software development life cycle. Design and maintenance is especially difficult when the system includes autonomic features. As the system size and variety of autonomic behaviors scale up, it increases the chance of many unexpected and unwanted interactions. Separate design tools can hide these potential interactions. To face these challenges, we propose an autonomic system integration platform where holistic design models capture system structure, target system resources, and autonomic behavior. The fault mitigative, autonomic behavior can be explicitly coupled to the components and underlying resources of the system. System generation technology is used to create the software that implements these coupled specifications, including communication between components with custom data type marshalling and demarshalling, system startup and configuration, fault tolerant behavior, and autonomic procedures for self-correction. This modeling schema, along with the tools to generate the various system components are described in this paper.

1. Introduction

System research has long been a part of the computer based systems landscape.Computer scientists and engineers have examined ways of combining components- whether individual transistors, integrated circuits, or devices - into large scale computer-based systems to provide improved performance and capability. Large Scale systems are difficult to design, build, and operate. The high number of components and component interactions also makes the system susceptible to component failure. The challenges stem from the characteristics of the system itself - large scale, high complexity and heterogeneity. The applications in which they operate demand extreme flexibility, trust worthiness, and distributed operation and administration.Large systems, however, do not need to be complex by definition. Their design can be greatly simplified if the component behavior can be decoupled from system structure, especially if the components can be linked in a linear fashion and information flows in a single direction. These design aspects must also be separated from the autonomic concerns of the system. Tools are needed that can support these multiple representations covering these orthogonal aspects of a system design.

2. Motivation and Challenges

One assumption in applying Model Integrated Computing [1] [2] [3](MIC) to a variety of systems and application domains is that every aspect of the system can and should be described in a single modeling language, a single model,and a single modeling environment. Clearly there are benefits of a single modeling language, since the interactions between different aspects of a system are precisely and explicitly defined.The common model, however, becomes problematic with multiple designers.Integration with version control systems, such as CVS, is also a challenge. A key question then, is how to extend the best practices of large-scale system development to MIC, without sacrificing the benefits of precisely understood interactions between multiple aspects of a system.

Our research is motivated in the context of a large-scale real-time physics system, being developed at Fermi National Accelerator Laboratory (Fermilab) [27]for characterizing the sub-atomic particle interactions that takes place in a high-energy physics experiment. There are several different aspects of this complex system, ranging from hardware topology, communication architecture, to software component configuration, fault-tolerance policy specification, data and message-types, message passing interfaces, run control, logging, online diagnosis, and deployment. There are large

groups of physicists and engineers (users) involved with designing these different aspects of the system. Different aspects of the system interact in varying degrees, and are evolving on different timelines. In addition, there is a need to version control the evolution of designs and design artifacts. All these aspects must be meaningfully incorporated in a single logical "super" modeling language. This is the focus of an ongoing effort to design and apply these tools.

3. Related Work

The idea of model based autonomic systems focussing mainly on the software architecture has existed for some years in variety of context. The use of architectural models as a centerpiece of model based adaptation has been explored by number of researchers [28]. The systems specified in there focussed mainly on the use of specific style to provide intrinsically-modifiable architectures.IBM's Tivoli monitoring is a systems manager using expert systems to isolate problems and correct them on a local machine [29]. Here the architecture is the operating system itself. The monitoring software maintains an external model which is accessed by scripts that isolate problems and repair faults. Garlan and Schmerl provide a more general approach for externalized adaptation of distributed applications [30]. Externalized adaptation favours a centralized system organization. In centralized model-based adaptation, an architecture manager maintains, analysis and corrects the system model.

As we describe below , our work has focussed mainly on the more pressing issue which is the integration of the several components which makes the large scale systems. In our previous publication [17] we have described the hierarchical autonomic fault mitigation architecture as the centre piece of our research , and this paper presents the implementation of this fault mitigation environment using the ARMOR framework which is the addition to the previous work along with the other challenging issuse which raises in the model-based adaptation research such as the Graphical Monitoring Environment (How do we add visual monitoring information on the fly in non-intrusive ways ? what kinds of things can we monitor visually?)and Automatic Build Systems of large scale system.

4. Proposed Solution- Model Based Autonomic Computing

As systems become more interconnected and diverse, architects are less able to anticipate and design interactions among components, leaving such issues to be dealt with at runtime, often with an implicit solution. Addressing this concern are the four pillars of autonomic computing [21], self-configuring, self-healing, self optimizing, self-protecting which offer a viable alternative for such large scale systems. Automated tools are required to assist the system developers in managing the complexity in designing an autonomic response system. These tools should offer:

- Higher-level abstractions for designing adaptive autonomic behaviors, which are easier to manipulate and maintain.
- Analysis of the autonomic responses to assess the systems ability to adapt with respect to different failure scenarios with a graphical user interface.
- The ability to synthesize low-level programming artifacts from the higher-level abstractions.
- Configuration of the system artifacts automatically from higher level of abstractions.
- The ability to hide details of the underlying communication protocols while living within their constraints.

This paper describes a tool suite addressing these needs that is based on the principles of Model Integrated Computing (MIC) [3] [2] [6]. The key elements of this approach are a Generic Modeling Environment (GME) as demonstrated in [6] that is used to instantiate multiple Domain-Specific Modeling Languages (DSME's) and a suite of translators [7]that assist in the transformation of domain models to low-level programming and simulation artifacts. The tools incorporate multiple "narrowly focused" domain specific modeling languages which will be explained in detail in sections:

- System Integration Modeling Language(SIML) - High level specification of the system.
- Data -Types Modeling Language (DTML) - Data type modeling and creation of a library of abstraction communication protocol API.
- Fault Mitigation Modeling Language (FMML) - Modeling of fault managers.
- GUI- Configuration Modeling Language (GML)- Modeling of required User Interface and synthesis
- Run Control Modeling Language (RCML)- Modeling of Run Control component using hierarchical finite state machine

The following sections describe each of these languages in detail.

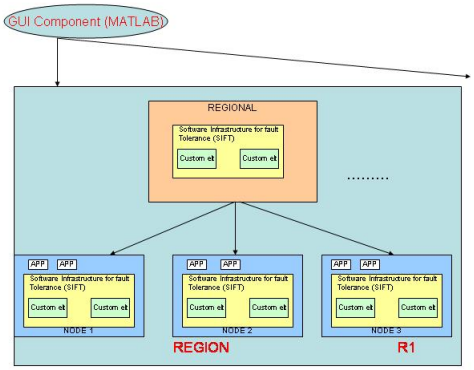

Figure 1. System Level Specification

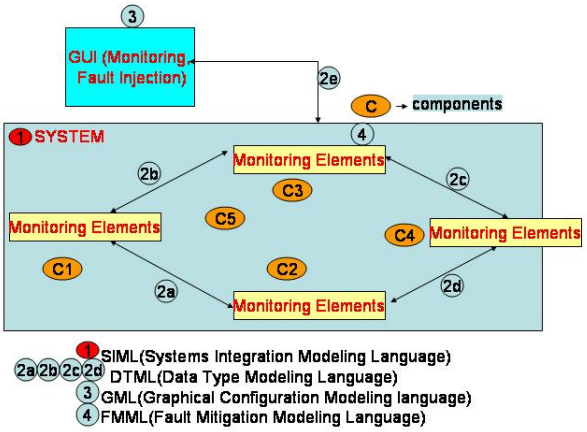

Figure 2. Component Interactions and Modeling Languages

5. Systems Integration Modeling Language

The System Integration Modeling Language (SIML) is the highest level of systems modeling language. SIML is a loosely specified model of computation used for capturing components, component hierarchy, and interactions within the system. It also allows users to model the information relevant for the system configuration. The proposed systems architecture has three levels of hierarchy - Regions (R1, R2 RN), each Region has Nodes (N11 N12 N13 N1N, N21 N22 ..), and each Node has applications running as shown in Figure 1. Each region is governed by a Regional Manager; there may be several regions, each of which has a set of nodes which the Regional Manager monitors. Each node has an individual Local Manager to monitor the applications running locally on that node. The Systems level will have the overall high level component specification, but the behavioral aspect of the components will be defined in a separate language. To address this challenge of representing the models in multiple languages we are leaning upon the experiences from the textual programming world. A fairly simple and intuitive notion that can be borrowed is that of 'import' or 'include'. Notice that an import or include does not merge the artifacts together, instead tools such as preprocessors, compilers, and linkers do all the backend work transparent to the end user.In languages which rely on elements defined in other languages, we introduce the concept of a Link type. A `Link` has attributes to identify the modeling language of the linked object, the file path of the linked object (relative to a CVS working-directory), and a persistent ID of the linked object as shown in Figure 3. A `Link` is an abstract type which can be specialized into concrete types within the language. Note that the abstract `Link`

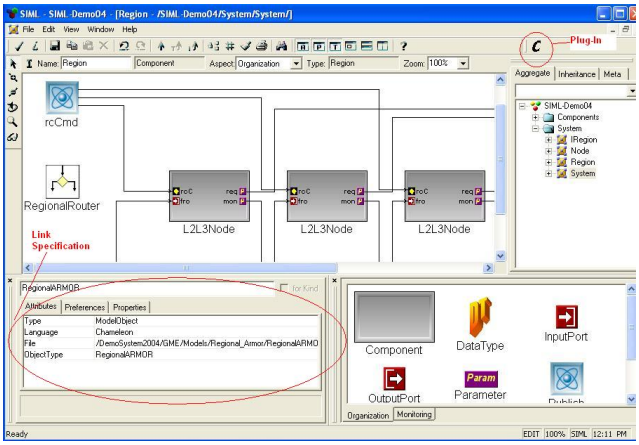

Figure 3. Link Specification.

type and its specialization to a concrete type in the modeling language capture the mapping and interactions between different modeling languages precisely. In addition to this basic augmentation to the modeling languages, we have developed plug-ins that facilitates the process of link creation and link navigation. The creation of a `Link` is non-trivial since it may require a mapping between concepts in different modeling languages. The onus of link creation, however, is placed on the plug-in developer and not on the domain users. There is now an additional challenge, particularly for synthesis in the sense that model translators now

need to navigate across multiple models and modeling languages. However, this challenge is similar in to those that have been addressed by pre-processors, compilers, and linkers, and there is no reason why similar techniques can not be brought to bear.

5.1. Data Type Modeling Language

While SIML allows the system developer to specify a hierarchical organization of components and fault-managers in the system, the communication between each of the managers and the other components is modeled using another language specific for message data type modeling. Elvin [26]is a network communications product based on a publish/subscribe paradigm. Elvin is similar to other publish/subscribe message passing systems in that messages are routed to one or more locations via client subscriptions which are based on the content of messages. Elvin is a commercial product which has high performance, low-latency and is massively scalable. Large scale processing farms such as the ones in for the BTeV Level 2/3 [8] [9]require both massive bandwidth for high data rate processing and low-latency communication for fault control and diagnostics. In order to hide the complexities of the Elvin communication, we have developed an abstract layer on the top of the chosen message passing system. The abstract layer was created in order to 1) to hide the complexities of the communication protocol and 2) to match the messaging API to our requirements. Message marshalling and de-marshalling code that is generated from DTML models conforms to the Elvin API. The data type modeling language in the BTeV environment is used for several purposes. First of all to accurately simulate the communication between various components in the model.

Furthermore it also helps in using the communication abstraction API such that there would be a standard way of defining these data types and also a standard way of marshalling and demarshalling the data using the Elvin Communication protocol. The BTeV data type modeling paradigm allows the specification of both simple and composite types. Simple types such as floats and integers specify their representation size i.e. number of bits used. Composite types can contain simple types and other composite types like struct, enum, union etc. Attributes of the field specify extra information such as array size or signed/unsigned representations. The environment is flexible enough to support the modeling of data types supported by several message passing systems. This data type modeling is used by other modeling environments such as the graphical user interface, fault managers etc. to model the messages for communication.

6. Autonomic Fault Mitigation using ARMOR framework

This section is an extension of the previous autonomic fault mitigation environment [17]. BTeV trigger system uses a reconfigurable, fault-tolerant framework based on reconfigurable processes known as Adaptive Reconfigurable Mobile Objects of Reliability (ARMOR) [13] [14] [15]. Each ARMOR process provides a particular fault mitigation service. Collectively these processes and their services form a reconfigurable fault-tolerance framework that is able to respond to faults from the surrounding environment. The services provided include, but are not limited to, error detection and recovery of applications on each node. By customizing the services that each process provides as well as the layout of the processes on a network of computing nodes, we can configure the framework to respond to different kinds of faults, thus making the distributed system more reliable. Configuration and customization for a large-scale system, however, is a complex task.

To achieve these goals, our fault mitigation approach has two components: a software implemented fault tolerance environment (SIFT) and a Fault-Mitigation Modeling Language (FMML). The SIFT environment operates over the heterogeneous non-fault-tolerant nodes in a system through a set of reconfigurable software processes. A SIFT process can be further decomposed into sub-processes, each implementing a particular fault-mitigation function. SIFT processes on a node collectively form a mitigation strategy. These processes monitor the application software on each node silently until a failure occurs, at which point the fault-mitigation functions of one or more processes are invoked to perform a mitigation action based on the nature of the fault. As a system's fault-tolerance requirements may change, the fault-tolerant framework may need to be reconfigured in order to meet these new requirements. Fault-Mitigation modeling language allows system operators to manage this reconfiguration process easily by providing a high-level interface on top of the implementation details of the SIFT environment. Reconfiguring the SIFT environment involves two procedures. The first procedure entails a decision on the placement of reconfigurable processes on each node of the system. This decision involves choosing processes with specific mitigation functionality as well as their quantity. The modeling language provides graphical constructs that represent these reconfigurable processes along with attributes to specify the mitigation functionality. The second procedure involves specifying fault-mitigation functionality for the sub-processes that compose a SIFT processes. The modeling language utilizes Statecharts-like [12]notation for this purpose. System states can be defined along with transitions between states. Mitigation actions are deployed as a consequence of switching between different states. Users

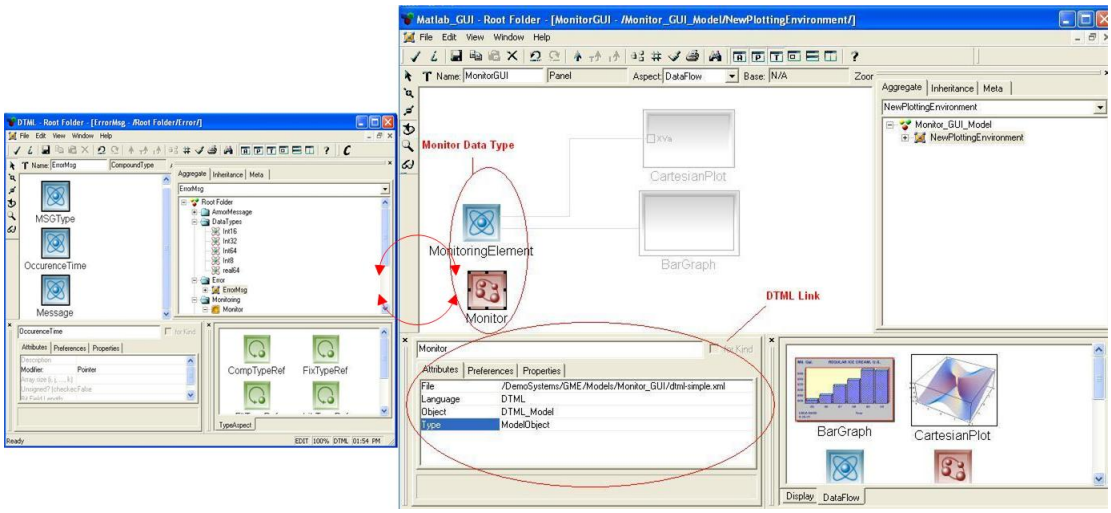

Figure 4. GUI (Graphical User Interface) using the Data Type Modeling Environment.

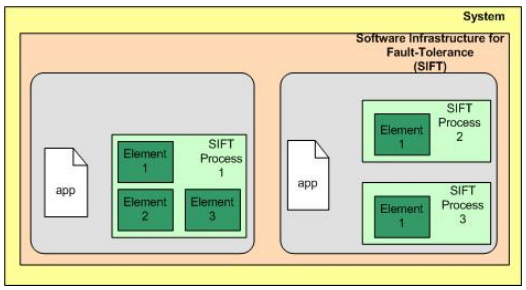

Figure 5. Fault Mitigation Architecture showing the SIFT Environment.

can choose to perform one or both of the procedures during the reconfiguration process. In addition,low-level programming artifacts are synthesized by a translator from the higher-level abstractions in the modeling tool to SIFT environment specific constructs. Specifically, two types of artifact is generated. One type of artifact is directly tied to quantitative and structural placement of reconfigurable SIFT processes. The second artifact of FMML is the C++ implementation of all SIFT sub-processes' mitigation functions. Each time a new configuration is specified in the modeling tool, the artifacts will be automatically regenerated conforming to the new configuration. The artifacts is then used to deploy the SIFT environment.

7. Autonomic Graphical User Interface

Monitoring and diagnostics of these large-scale autonomic systems are essential to ensure the system is functioning correctly and adapting properly to different fault scenarios. However, monitoring different aspects of the system at different times would require several variations of 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. Model based approach has been used to achieve configurability in the design of user interfaces [24]. Configurable user interfaces in the BTeV system enable the physicists to dynamically view data and error conditions in ways that aid in system analysis. User interfaces also provide the mechanisms for system operators 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. The language provides the following features:

- The language allows the users to create multiple user interface panels as well as specify the plots/controls that are a part of these panels. This is depicted with containment hierarchy.

- For each of the plot/control, two primary properties are to be specified, namely its location as well as the data from outside the environment that is to be plotted/controlled. This is depicted as two different aspects of the plot/control objects.

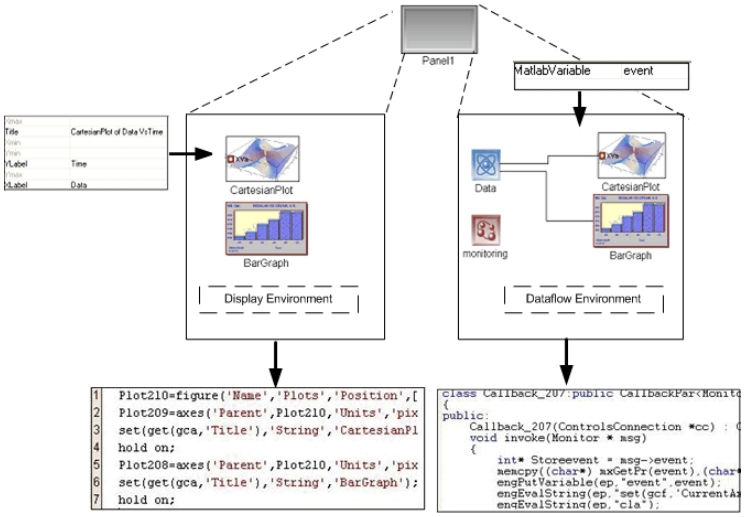

Figure 6. Graphical User Interface Specifications and Generation showing the mapping of the models to the Matlab code generation.

- The user may specify the properties of all the objects in the environment. These properties are depicted as attributes of each object.

- Once the user has modeled the system using the language provided, the tool is capable of generating the code necessary to create the user interface.

- The tool is capable of generating software for a variety of run-time platforms some examples being Matlab, Experimental Physics and Industrial Control System (EPICS) [25].

The user interfaces are configured using the GUI configuration modeling language. Once the user interface has been configured using the models, the code for the UI is generated. The models may be used to generate software for a variety of run-time platforms. Currently the target environment for the user interface is Matlab. Figure 6 shows an example user interface configured using the modeling language developed. The figure also shows the generation of code artifacts from the specified models which include the following:

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 models positions to the generated Matlab user interface. The code for the structural information is generated as Matlab .m files.

2. Dataflow - The data flow aspect models the data type to be plotted in the User Interface. The UI receives data continuously. This uses the Link to the Data Type Modeling Environment (DTML) where the data to be plotted is modeled. The data transfer in the system takes place through publish-subscribe mechanism as mentioned above. The user interface subscribes to specific messages that contain data to plot. The Matlab Component Object Model (COM) interface is used for this purpose. The code for the data flow aspect is generated in C++. Figure 7 shows a generated user interface obtained from the structural code and data flow models shown in Figure 6.

8. Autonomic Configuration and Heterogenous Build Systems

An interesting duality exists between traditional code compilation procedures and the graphical model translation process. In fact, one can assert that from a human perspective, the two processes are the same in principle, with the only difference being the higher level of abstraction leveraged by the modeling process [13]. Systems that incorporate both the interpretation of models and the compilation of hand-written source code present a unique challenge to system designers regarding the maintenance and automatic building and testing procedures using such a heterogeneous source-based environment.

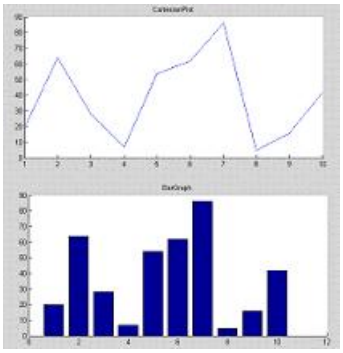

Figure 7. Simple Example- Generated User Interface from the Models showing the Bar Graph and the Line Graph.

Inherently, changes to system models require regeneration of any domain artifacts produced from the models which are affected by the change. We see this effect in traditional source code compilation schema, namely, objects are built or linked only when the source files are modified. Accordingly, objects can be linked against new libraries to provide better functionality without recompilation of the executable (only a link is performed). Similar methods are currently not employed for the process of model translation. Source artifacts are typically either 1) generated all at once from an integrated system model or 2) individual artifacts or groups of artifacts are generated by the modeler by hand in one-off fashion and the new artifacts are incorporated into the system.

This behavior leads to a problem in that many of the generated artifacts may be quite similar (the same, in fact) to the artifacts they are replacing. Integration of these newly time-stamped artifacts may therefore trigger a recompilation of the generated code regardless of whether any changes are present or not. Graphical models are often not intermixed with other types of source code in Code Management Systems (CMS) due in part between the tight coupling between model formats, model languages, and the tools which understand them. Automatic translation, or "compilation," of these models require tools that support the automation needed for things such as nightly builds, automated testing, etc. Typically these modeling tools are tied to full graphical applications and require manual model importation and translation using a graphical user interface (GME, Ptolemy, etc). This lack of automotive functionality makes integration with existing CMS's extremely difficult. It is desirable, therefore, to have a code management system which is capable of managing models in a similar fashion as source code which employs many of the same time saving features of traditional CMS's. The Unified Data Model (UDM) [7]

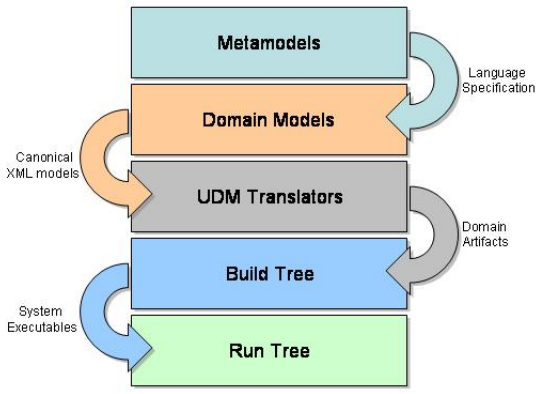

Figure 8. Additional levels of processing must be added when incorporating model translators into heterogeneous automatic build systems. The textual annotations describe what each level provides for subsequent levels..

toolset developed at Vanderbilt was used to allow the interpretation of models within a UNIX environment. One feature of the UDM toolset is the ability to create a standalone command line executable that is tailored for model translation in the context of a given language. Both the source code for these command line model translators as well as the language specification files is stored in the CMS. However, this requires a sequence of events be enforced that will first build the model translators before invoking these translators on their associated models. The process involves a strategy to govern the sequence of events necessary to ensure that the proper artifacts are generated at each level so that the next level can execute. The process is the following:

1. Build the set of language specifications from the set of metamodels using the UDM metamodel translator. The generated language specifications are needed by the UDM tools to generate model translators that understand the languages specified by the metamodels.

2. Build the UDM domain model translators using the modeling language specifications generated in stage 1

3. Use the newly built UDM translators to interpret the domain (system) models. These translators may produce source code artifacts (.cc, .h, makefiles, etc) which fill in remaining portions of the source tree.

4. Execute a build of the Build Tree

5. Populate the Run Tree for packaging and distribution to remote nodes.

This approach to integrating model based tools with a traditional source based CMS involves creating additional processes to govern the way in which modeling languages, model translators, domain models, and artifacts are maintained. Since the generation of modeling languages from a meta-level specification is itself a translation process performed by a UDM model translator, both domain model translation code and the meta-level translation code can be placed under version control.

9. Conclusion and Future Work

This paper presents an approach to large scale autonomic system design and development. The tools support automation of the process, keeping in consideration key factors such as easy maintenance and the demands of ever evolving systems. The approach uses concepts of model integrated computing by providing a set of loosely coupled modeling languages that allow the specification of different components of the system to facilitate easier integration of components while preserving the components' visual interdependencies. The concepts presented in the paper are motivated by the example of a large scale real-time physics system called the BTeV system.

The development of these tools are ongoing as part of the Fermilab BTeV project and the NSF ITR program. We continue to refine the modeling paradigm and the interdependencies between these modeling aspects. In addition, we continue to scale up the system, with the eventual goal of supporting the full-scale, 2000-5000 processor BTeV system. We also continue to streamline the code generation process and optimize the generated code. We also are studying the potential for tools to verify target systems prior to implementation. These include formal analysis of mitigation state machines, and the generation of software for monte-carlo simulations.

10. Acknowledgements

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, Pittsburg, and Syracuse Universities.

References

[1] Sztipanovits J., "Engineering of Computer-Based Systems: An Emerging Discipline", Proceedings of the IEEE ECBS'98 Conference, 1998.

[2] Nordstrom G., "Metamodeling - Rapid Design and Evolution of Domain-Specific Modeling Environments", Proceedings of the IEEE ECBS '99 Conference, 1999.

[3] J.Sprinkle, "Model Integrated Computing," IEEE Potentials, vol 23, no 1, pp.28-30, Feb 2004.

[4] Bapty T., Neema S., Scott J., Sztipanovits J., Asaad S., "Model-Integrated Tools for the Design of Dynamically Reconfigurable Systems", VLSI Design, 10, 3, pp. 281-306, 2000.

[5] Agrawal A., Bakshi A., Davis J., Eames B., Ledeczi A., Mohanty S., Mathur V., Neema S., Nordstrom G., Prasanna V., Raghavendra, C., Singh M., "MILAN: A Model Based Integrated Simulation Framework for Design of Embedded Systems", Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES), Snowbird, UT, June, 2001.

[6] Ledeczi A., Maroti M., Bakay A., Nordstrom G., Garrett J., Thomason IV C., Sprinkle J., Volgyesi P., "GME 2000 Users Manual (v2.0)", Institute For Software Integrated Systems, Vanderbilt University, December 18, 2001.

[7] E. Magyari, A.Bakay, A. Lang, T. Paka, A. Vizhanyo, A. Agrawal, G. Karsai: "UDM: An Infrastructure for Implementing Domain-Specific Modeling Languages", The 3rd OOPSLA Workshop on Domain-Specific Modeling, OOPSLA 2003, Anahiem, California, October 26, 2003.

[8] Buttler J.N., et. al, "Fault Tolerant Issues in the BTeV Trigger", FERMILAB-Conf-01/427, December 2002.

[9] Kwan S., "The BTeV Pixel Detector and Trigger System", FERMILAB-Conf-02/313-E, December 2002.

[10] Avizienis A., Avizienis R., "An immune system paradigm for the design of fault-tolerant systems", Presented at Workshop 3: Evaluating and Architecting Systems for Dependability (EASY), in conjunction with DSN 201 and ISCA 2001, 2001.

[11] Avizienis A., "Toward Systematic Design of Fault-Tolerant Systems", IEEE Computer, 30(4):5158, April 1997.

[12] Harel D., "Statecharts: A Visual Formalism for Complex Systems," Science of Computer Programming, vol. 8, pp. 231-274, June 1987.

[13] Z.T.Kalbarczyk, R.K.Iyer, S.Bagchi, K.Whisnant, "Chameleon: A software infrastructure for adaptive fault tolerance," IEEE Transactions on Parallel and Distributed Systems, vol. 10, no. 6, pp.560-579, June 1999.

[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] S. Nordstrom, A Runtime Environment to Support Fault Mitigative Large Scale Real-Time Embedded Systems Research, Master’s Thesis, Graduate School of Vanderbilt University, May 2003.

[17] 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.

[18] S. Shetty, S. Neema, T. Bapty. ”Model Based Self Adaptive Behavior Language for Large Scale Real-Time Embedded Systems”. IEEE Conference on Engineering of Computer Based Systems (ECBS), Brno, Czech Republic, (May 2004).

[19] S. Shetty, ”Towards Developing Tools and Technologies for Modeling Faults in Large Scale Real-Time Embedded Systems”, Master’s Thesis, Graduate School of Vanderbilt University, May 2004 .

[20] Ledeczi A., Bapty T., Karsai G.: Synthesis of Self-Adaptive Software, IEEE Aerospace 2000 , CD-ROM Reference 10.0304, Big Sky, MT, March, 2000.

[21] IBM Autonomic Research webpage http://www.research.ibm.com/autonomic

[22] Flavin Cristian,”Understanding Fault Tolerant distributed systems, ”Communications of ACM, vol. 34, pp. 56-78, February, 1991.

[23] D. Vashishtha, S. Neema and T. Bapty, ”Simulation Based Analysis of Real Time, Fault Tolerant, Large Scale Embedded System,” Proceedings of the Huntsville Simulation Conference 2003, October 2003.

[24] Jos A. Macas 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 2004.

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

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

[27] BTeV RTES Fermi lab Webpage http://www-btev.fnal.gov/public/hep/detector/rtes/index.shtml

[28] Oriezy, P., Gorlic,M.M., Taylor,R.N., Architecture Based Runtime Software Evolution. Proc,International Conference on Software Engineering 1998 (ICSE’98), Kyoto, Japan, Apr. 1998

[29] G. Lanfranchi, P. Della Peruta, A. Perrone, and D. Calvanese. Toward a new landscape of systems management in an autonomic computing environment. IBM Systems Journal, 42(1):119128, 2003.

[30] S-W. Cheng, D. Garlan, B. R. Schmerl, J. P. Sousa, B. Spitznagel, P. Steenkiste, and N. Hu. Software architecture-based adaptation for pervasive systems. In Proceedings of the International Conference on Architecture of Computing Systems, pages 6782. Springer-Verlag, 2002.