A Model Driven Tool for Automated System Level Testing of Middleware

Turker Keskinpala, Abhishek Dubey, Steve Nordstrom, Ted Bapty, Sandeep Neema

Institute for Software Integrated Systems, Vanderbilt University,
2015 Terrace Place, Nashville, TN 37203
Phone: 615-343-7472 Fax: 615-343-7440
{tkeskinpala, dabhishe, steve-o, bapty, sandeep}@isis.vanderbilt.edu

Abstract
This paper presents a contribution to the challenges of manually creating test configurations and deployments for high performance distributed middleware frameworks. We present our testing tool based on the Model Integrated Computing (MIC) paradigm and describe and discuss its generative abilities that can be used to generate many test configurations and deployment scenarios from high-level system specifications through model replication.

1. Introduction

Software systems are increasingly becoming larger, more complex and distributed. Advances in hardware systems make it possible to run more complex software on less costly platforms. However, testing processes do not seem to be keeping pace with the advances in hardware platforms and increase in software size. It is apparent that manually written, handcrafted system tests are not effective [1].

Software that resides between applications and run-time infrastructure is called *middleware* [2]. Middleware is increasingly being used to cope with the complexity of designing efficient and scalable distributed software systems.

Model-based design [3], [4] provides a scalable methodology for system design and analysis based on sound system theory and abstraction by integrating the efforts in system specification, design, synthesis, validation, verification and design evolution. In this approach, the use of models enables encapsulation of only relevant system properties and sifts out irrelevant details, leading to mitigation of design complexity. Augmented with model

transformation techniques, the system models can be automatically refined/abstracted for design evolution and can serve as input to tools that perform validation and verification.

One such model based design methodology is Model Integrated Computing (MIC) [4]. MIC brings in key concepts of domain modeling to the paradigm of model driven system development. One capability supported by MIC is the definition and implementation of *domain-specific modeling languages* (DSMLs). Crucial to the success of DSMLs is *metamodeling* and *auto-generation*. A *metamodel* defines the elements of a DSML, which is tailored to a particular domain. The modeling language which is used to construct metamodels is known as a metamodeling language. Auto-generation involves automatically synthesizing useful artifacts from models, thereby relieving DSML users from the specifics of the artifacts themselves, including their format, syntax, or semantics.

The approach described in this paper mitigates the complexity of manually creating system test suites by applying these model based techniques to create a platform for representing system specifications with varying levels of detail in order to automatically generate system tests and configurations. The paper is organized as follows: in Section 2 we will give the motivation and challenges. In Section 3 we will go over some related work and in Section 4 we will describe our proposed solution. Section 5 will describe the partial implementation of our testing tool. We will conclude and mention our future plans in Section 6.

2. Motivation and Challenges

Our research is motivated in the context of testing a high performance distributed middleware framework, XDAQ [5], designed and implemented at the European Organization for Nuclear Research (CERN) for the Compact Muon Solenoid (CMS) experiment's data acquisition system [5]. CMS experiment's data rates will exceed the data rates of currently operating high energy physics experiments. The XDAQ application framework provides run-time support for clustered applications by mapping the Intelligent I/O (I2O) concept [6] to a distributed computing environment and encapsulating specification details into an application-programming framework. In [5], overall requirements for middleware such as the XDAQ are stated as high efficiency and flexibility in configuration. The XDAQ follows the I2O specification for configuration and control of the cluster and the applications of its executives and adapts XML as the data format for serializing the I2O functions. Predefined functions are used to configure applications, access and set their operational parameters.

One aspect of testing a high performance distributed middleware framework and its components is to be able to create many configurations that would configure functional operation of components as well as their deployment in the cluster. It is cumbersome and

inefficient for test engineers to write XML test configurations by hand as the tester would be making many copy-paste operations which can introduce errors into the process. Moreover, the configuration space of the control and deployment parameters of applications within the framework is sufficiently large; there is no way for the tester to manually create configurations for all possible combinations of parameters. Last but not the least, it would be very time consuming to scale up and modify a manually written XML test configuration in response to changes in hardware resources or other test criteria.

3. Related Work

Model driven testing is not a new approach for generating suite of test cases. Behavioral modeling to generate test cases has long been used for software testing [7], [8]. In [9], Dalal et al., use data models and a generation infrastructure for automatic generation of test cases. UML is increasingly being used for system testing. A UML based tool supported test methodology is described in [10]. In [11], derivation of functional system test requirements from UML is explained. [12] and [13] present an architecture and set of tools for model based testing using UML for distributed systems.

Model driven development is also used for development, configuration and deployment of distributed real time middleware. In [14], Schmidt et al., address the issues of configuring and deploying middleware frameworks and introduce a MIC tool that can be used configure and assemble middleware.

Our approach differs from the approaches that only enable automatic generation of test cases from behavioral system models. We are not only dealing with generating test cases from a behavioral system model but we also take into account the resources, applications, application dataflow at a higher-level to generate many test directives at once. In essence, we are combining test generation from behavioral modeling approaches with an approach similar to [14].

4. Proposed Approach

This paper describes an automated system-level test configuration tool tackling the aforementioned problems using the principles of Model Integrated Computing (MIC) [4], [15], [16]. At the core of this approach is the Generic Modeling Environment (GME) which as demonstrated in [16] is used to instantiate multiple Domain-Specific Modeling Languages (DSMLs) and translators that provide the means to transform domain models into low-level programming and simulation artifacts.

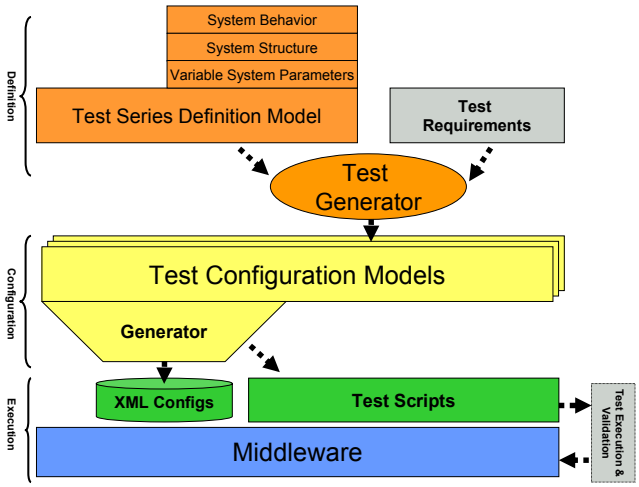


Figure 1: Testing tool structure

The definitive goal of our automated testing tool is to automatically generate series of test cases/configurations from a higher-level system model along with a deployment model and resources. Figure 1 shows the overall structure of our approach.

At the top of this structure is the definition layer that is used to define the system behavior and structure along with *test requirements*. The goal is to create multiple test configurations from this definition layer. It is possible to achieve this by creating a DSML called *Test Series Definition Language*. The model, *Test Series Definition Model (TSDM),* designed in this language captures the configuration schema of the domain, details of the application types of the domain, hardware resources, and hardware resource configurations. Moreover, the TSDM enables a test designer to define the variability range of behavioral and structural system parameters.

Test Generator is responsible for generating multiple concrete test configuration models by binding the variabilities in TSDM to specific values based on the test requirements. Test Generator will be a heuristic based tool which will adhere to the crucial issues of necessity, sufficiency, and coverage.

Test Configuration Models (TCM's) constitute the configuration layer of our approach. These models are not meant to be seen or modified by the test designer. TCM's capture the middleware's configuration schema and are transformed into XML configuration files that will configure the middleware. A single XML file is generated for each test configuration model so that there will be as many XML files as there are test configuration models. In addition to the XML configuration files, test scripts to run the series of test configurations can be generated.

The lowest level of the structure is the execution layer. In this layer, the test scripts generated by TCM's can be run manually or they can be supplied to the *Test Execution and Validation* engine which drives the test cases. The Test Execution and Validation engine uses the expected results that are embedded in the test scripts to decide on the success or failure of the tests.

5. Model Driven Testing of Middleware

We are applying the model-driven testing tool to the testing of the XDAQ middleware. For the implementation we chose a bottom-up approach. We have started by creating a modeling language to manually design test configurations instead of generating them from the test series definition models. The main reason for doing so was to get a better understanding of XDAQ configuration and possible abstractions. So far, we have implemented the Test Configuration Modeling Language (TCML) and XML generation from TCML. In this section, we will also describe a prototype for defining parameterized test series definition models that will be used to generate several test configuration models.

5.1. Test Configuration Modeling Language (TCML)

The TCML is created using the GME metamodeling environment. Figure 2 shows the GME editor and the metamodel for a portion of the TCML.

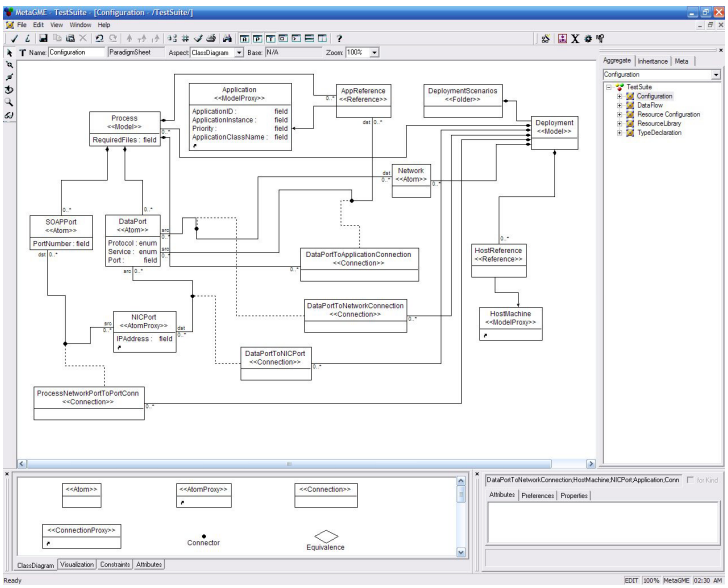


Figure 2: Metamodel for TCML as seen in GME

The TCML metamodel captures the configuration schema for middleware applications, types (e.g. applications), resources, resource configurations, and dataflow among applications. The TCML also provides a type folder, hardware resources folder, resource configuration folder, dataflow scenarios folder and deployment scenarios folder.

Using TCML, several types can be created inside the type folder. Figure 3a shows an example application type that is created in GME using TCML. Application parameters are characterized as *configurable* and/or *observable* or neither. Configurable parameter means that the value of the parameter will play a role in the configuration. If a parameter is observable, it means that its value or state can be monitored for test validation.

Figure 3b shows definition of a resource. In Figure 3b, a host machine with 2 network interface cards is shown. One can create as many resources as there are inside the resource folder. Figure 3c shows a resource configuration. Host machines that are used in the resource configurations are instantiations of the hosts that are available in the hardware resources folder as shown in Figure 3b. Figure 3c depicts the configuration of two hosts on a network. Finally, Figure 3d shows a dataflow scenario for selected applications. The applications that are used in the dataflow scenarios are the instantiations of applications that are in the type folder. Instantiating applications in GME as subtypes enable assignment of different values to application parameters and create several subtypes from a common type. In Figure 3d, four Roundtrip applications are used and the dataflow among them denote the

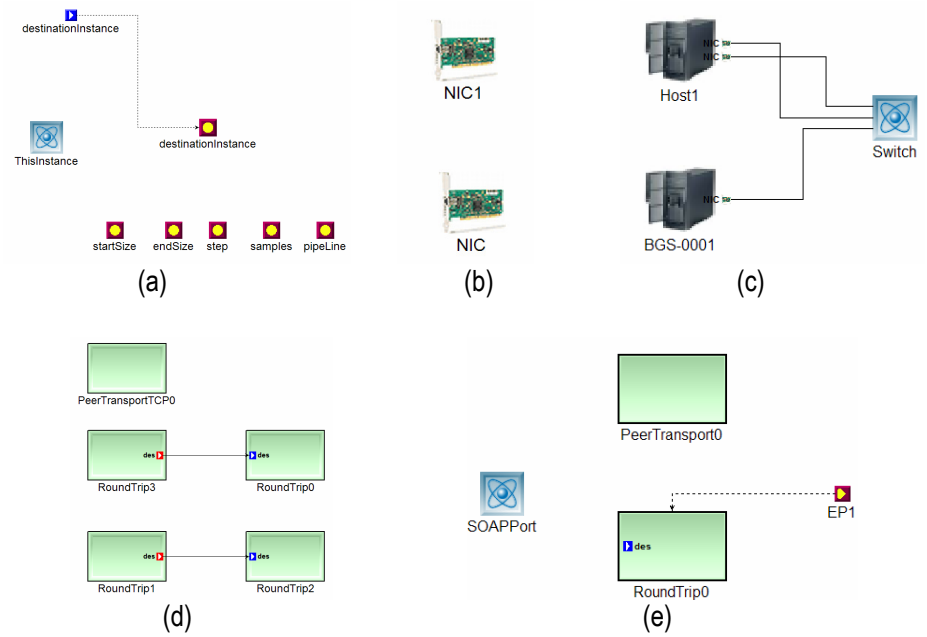


Figure 3: (a) Type Definition (b) Resource Definition (c) Resource Configuration (d) Dataflow (e) Applications deployed in a XDAQ context

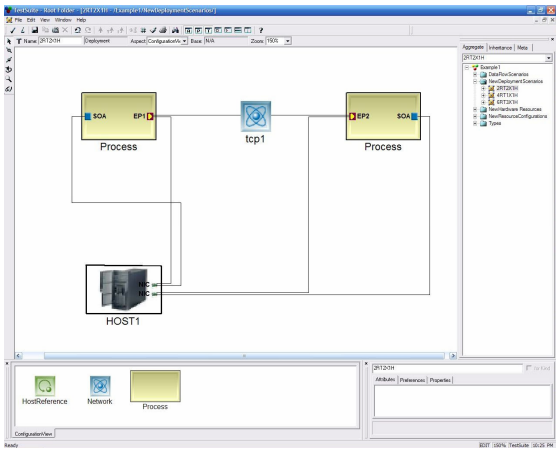


Figure 4: The overall deployment scenario/test configuration

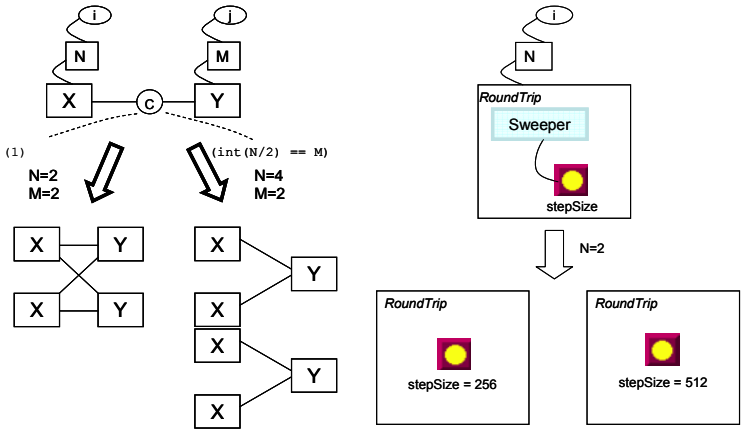
which Roundtrip is a sender and which is a receiver. Figure 3e shows the deployment of the applications on a XDAQ context which is a host that is running the XDAQ executive on a port that is accessible from outside. The applications seen in Figure 3e are instantiations of the applications that are defined in a dataflow scenario. It is also possible to connect applications to a data port, so called an *endpoint*, denoted as EP1 in figure.

There can be many applications that are deployed in a XDAQ context and there can be many contexts in one configuration. Figure 4 shows a deployment scenario where there are two contexts on one host with endpoints residing on a logical network named “tcp1”. The host machine seen in this deployment scenario is an instantiation of the host that is used in the resource configuration shown in Figure 3c. Many such deployment scenarios can be generated inside the deployment scenarios folder. All hardware resources, types, dataflow scenarios in their respective folders are available when creating deployment scenarios.

It is important to observe that using MIC concepts enabled us to decouple, types, resources, dataflow and deployment aspects of the design from each other. This is desired because a change in one aspect can be propagated through the model without requiring a major modification to the rest of the model. For example, if a host machine is replaced by a new one, it would be enough to replace the instance of the host in a deployment scenario with the new host. Furthermore, an entirely new configuration can be generated by changing some parameters of an application subtype without changing the deployment or vice versa.

5.2. Generating Artifacts

MIC approach uses generators (model interpreters) to translate models into other artifacts that are used in analysis or at run-time in the application [17]. In this implementation, input to our model interpreter was each TCM in the deployment scenarios folder, and the generated



(a) (b)

Figure 5: (a) Varying structure (b) Varying application parameters

artifact is an XML configuration file for each TCM input. After the tool implementation is complete, TSDM will be the input and many TCM's will be the artifacts of the model interpretation.

5.3. Test Validation

As mentioned in Section 4, test result validation is handled by the Test Execution and Validation engine as it compares the values of observable parameters with the expected values generated from the TCM's.

5.4. Test Series Definition Language Prototype

Defining series of tests can be achieved if we parameterize the models. For parameterization it is necessary to determine possible variations in configurations. For XDAQ configuration, it is possible to change the number of XDAQ contexts, the number of applications deployed in each context, and the parameter values of each deployed application. In addition, the mapping between applications and XDAQ contexts as well as the mapping between XDAQ contexts and hardware resources is variable. The problem may be viewed as creating a variable structure. We can use three concepts for creating variable structures:

- *Iterators*: Definition of a series, e.g. i,j,k = {start:step:stop}
- *Replicators*: Functions of iterators, e.g. N =i, M=j
- *Connectors*: Connection conditions as functions of replicators, e.g. C = (N/2==M)

Figure 5 shows how variation in structure and in application parameters may be achieved. Figure 5a shows two possibilities for variation. The left hand side replicates the applications X and Y by two and connects all the applications to each other since the connection condition is (1). The right hand side replicates X by four and Y by 2 and connects each pair of X to a Y.

Figure 5b shows how application parameters can be changed. A *Sweeper* (which is a specialized iterator) can be attached to a parameter whose value would be parameterized for each replication of the application.

6. Conclusion and Future Work

In this paper, we described our ongoing work of using MIC concepts to create a model-driven automated system level testing tool. We have given the overall tool structure which aims to generate test configurations from a high-level language which uses a system model along with a deployment model. We have described our partial implementation of the tool for the system level testing of the high performance distributed XDAQ middleware. We have also given a prototype of how we are planning to create variable structures as part of the test series definition language.

We have already started to focus on determining the specific aspects of the system to test to be able to create a TSDM. Then, we will work on the heuristic search algorithms for the implementation of the test generator which will be a critical component of our approach.

7. 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 University and European Organization for Nuclear Research (CERN).

8. References

[1] Stobei K., Microsoft, “Too Darned Big To Test”, Quality Assurance, Vol. 3, No. 1, February 2005

[2] Schantz R.E., Schmidt, D.C. “Middleware for Distributed Systems: Evolving the Common Structure for Network-centric Applications”, In Marciniak, J., Telecki, G., eds.: Encyclopedia of Software Engineering. Wiley & Sons, New York (2002)

[3] Greenfield J., Short K., Cook S. and Kent S., “Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools”, John Wiley & Sons, New York, 2004
[4] Sztipanovits J. and Karsai G., “Model-integrated computing”, IEEE Computer, Volume 30(4):110-111, 1997
[5] Gutleber J., Antchev G. et al. “Clustered Data Acquisition for CMS”, Proceedings of the International Conference on Computing in High Energy and Nuclear Physics, CHEP 2001, Beijing, China, September 3-7, 2001, Ed. H.S. Chen, pp. 601-605, Science Press, ISBN 1-880132-77-XX
[6] I2O Special Interest Group, Intelligent IO Architecture Specification 2.0, 1999 (www.i2osig.org)
[7] Clarke M. J., “Automated test generation from a behavioral model”, Proceedings of the Eleventh International Software Quality Week, San Francisco, CA, May 1998
[8] Cheng K. T., Krishnakumar A. S. “Automatic functional test generation using the extended finite state machine model“, Proceedings of the 30th International Conference on Design Automation, p.86-91, June 14-18, 1993, Dallas, Texas, United States
[9] Dalal S. R., Jain A., Karunanithi N., Leaton J. M., Lott C. M., Patton G. C., Horowitz B. M.,“Model-based testing in practice“, Proceedings of the 21st International Conference on Software Engineering, p.285-294, May 16-22, 1999, Los Angeles, California, United States
[10] Basanieri, F., Bertolino, A., “A Practical Approach to UML Based Derivation of Integration Tests”, Proc. 4th International Software Quality Week Europe (QWE’2000), Brussels (Belgium), November 20–24, 2000
[11] Briand, L. and Y. Labiche, “A UML-based approach to system testing”, in M. Gogolla and C. Kobry, editors, UML 2001, LNCS 2185 (2001)
[12] Cavarra A., Davis J., Jeron T., Mounier L., Hartman A., and Olvovsky S., “Using UML for automatic test generation”, In International Symposium on Software Testing and Analysis ISSTA, 2002
[13] Hartman A. and Nagin K., “The AGEDIS Tools for Model Based Testing”, In Proceedings of ISSTA’2004, 2004
[14] Schmidt D., Gokhale A., Natarajan B., Neema S., Bapty T., Parsons J., Gray J., Nechypurenko A., Wan N., “CoSMIC: An MDA Generative Tool for Distributed Real-time and Embedded Component Middleware and Applications”, 2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture, 2002
[15] Nordstrom G., ”Metamodeling - Rapid Design and Evolution of Domain-Specific Modeling Environments”, Proceedings of the IEEE ECBS ’99 Conference, 1999
[16] 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
[17] Karsai, G.; Sztipanovits, J.; Ledeczi, A.; Bapty, T., "Model-integrated development of embedded software," *Proceedings of the IEEE* , vol.91, no.1pp. 145- 164, Jan 2003