1999-05-12

Experiences Using Semi-Formal Methods During Development of Distributed, Research-Oriented, System-Level Software

NPS-CS-99-004

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Experiences Using Semi-Formal Methods During Development of Distributed, Research-Oriented, System-Level Software

by

David St. John
Taylor Kidd
Debra Hensgen
Mantak Shing
Shirley Kidd

May 12, 1999

INITIAL DISTRIBUTION LIST

1.	Professor Debra Hensgen Naval Postgraduate School, Code CS, Monterey, CA 93943	10
2.	Dudley Knox Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5100	2
3.	Research Office, Code 09 Naval Postgraduate School Monterey, CA 93943-5000	1

REPORT DOCUMENTATION PAGE

Form approved

OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	12 May, 1999	TECHNICAL REPORT

4. TITLE AND SUBTITLE
EXPERIENCES USING SEMI-FORMAL METHODS DURING DEVELOPMENT OF DISTRIBUTED, RESEARCH-ORIENTED, SYSTEM-LEVEL SOFTWARE

5. FUNDING
MSHN Project

6. AUTHOR(S)
David St. John, Taylor Kidd, Debra Hensgen,Mantak Shing, Shirley Kidd

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School, 833 Dyer Road, Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER
NPS-CS-99-004

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
DARPA / ITO
3701 North Fairfax Drive
Arlington, VA 22203-1714

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words.)

The Management System for Heterogeneous Networks (MSHN) is a large, distributed research software system project that began over 18 months ago. The primary goal of MSHN is to develop a framework within which next-generation resource management system (RMS) issues can be investigated. The initial MSHN design was developed using basic object-oriented design principles and the initial prototype was built with object-oriented technology. After building an initial proof-of-concept prototype, we looked to the standardized terms, symbols and diagrams of the Unified Modeling Language to explain the functionality of MSHN to new students and staff members of the development group, as well as interested colleagues outside of the group. As we learned more about the UML and applied it to MSHN in further detail, we found that this semi-formal method not only (I) helped us to communicate MSHN's functionality to others, but also (ii) improved our design, helping us identify bloated packages and opportunities for object re-use, and (iii) enabled us to more easily settle some open questions that had previously been sources of contention among the members of the MSHN team. Additionally, we found the Unified Process to be quite useful as a framework for building research-level, distributed systems software.

14. SUBJECT TERMS

software engineering, distributed computing, semi-formal methods, Unified Modeling Language, software development, MSHN, resource management system

15. NUMBER OF PAGES
08

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassifed	Unclassified	Unclassified	

NSN 7540-01-280-5800

Standard Form 298 (Rev. 2-89)
Prescribed by

ANSI Std 239-18

Enclosure (6)

NAVAL POSTGRADUATE SCHOOL
Monterey, California 93943-5000

RADM Robert C. Chaplin
Superintendent

R. Elster
Provost

This report was prepared for Naval Postgraduate School and funded by DARPA.

This report was prepared by:
David St. John, Taylor Kidd, Debra Hensgen, Mantak Shing, Shirley Kidd

David St. John
Staff
Computer Science Department

Taylor Kidd
Associate Professor
Computer Science Department

Debra Hensgen
Associate Professor
Computer Science Department

Mantak Shing
Associate Professor
Computer Science Department

Shirley Kidd
Staff
Computer Science Department

Reviewed by:

Dean D. Boger, Chair
Department of Computer Science

Released by:

D. W. Netzer
Associate Provost and
Dean of Research

Experiences Using Semi-Formal Methods During Development of Distributed, Research-Oriented, System-Level Software

David St. John[§], Taylor Kidd[†], Debra Hensgen[†], Mantak Shing[†], Shirley Kidd[§]

[†]Department of Computer Science
833 Dyer Road, Code/CS
Naval Postgraduate School
Monterey, CA 93943 USA
{kidd, hensgen}@cs.nps.navy.mil

[§]Anteon Corporation
2600 Garden Road
Suite 220A
Monterey, CA 93940 USA
{stjohn, kidds}@cs.nps.navy.mil

Abstract

The Management System for Heterogeneous Networks (MSHN[1]) is a large, distributed research software system project that began over 18 months ago. The primary goal of MSHN is to develop a framework within which next-generation resource management system (RMS) issues can be investigated. The initial MSHN design was developed using basic object-oriented design principles and the initial prototype was built with object-oriented technology (IDL, C++, Java, and CORBA). After building an initial proof-of-concept prototype, we looked to the standardized terms, symbols and diagrams of the Unified Modeling Language (UML) to explain the functionality of MSHN to new students and staff members of the development group, as well as interested colleagues outside of the group. As we learned more about the UML and applied it to MSHN in further detail, we found that this semi-formal method not only (i) helped us to communicate MSHN's functionality to others, but also (ii) improved our design, helping us identify bloated packages and opportunities for object re-use, and (iii) enabled us to more easily settle some open questions that had previously been sources of contention among the members of the MSHN team. Additionally, we found the Unified Process to be quite useful as a framework for building research-level, distributed systems software.

1 Introduction

The Management System for Heterogeneous Networks (MSHN) is a large, distributed, system-level research project. An object-oriented design approach was used in the design and implementation of the first version of MSHN. Because we are developing *research*, system-level software, as opposed to *production*-quality software, we did not initially find it necessary to use semi-formal or formal methods and processes. However, when the MSHN project was selected for integration with other projects, we were specifically requested to describe the high-level functionality and public interfaces of the MSHN components using the Unified Modeling Language (UML) for the benefit of the integration team. After completing this high-level description, and finding much to like about the UML, we began applying it to the lower-level design of the second version of the MSHN components. This use of the UML allowed us to identify several cases of bloated packages and opportunities for

[1] Pronounced "mission"

object re-use that we had not previously identified. We also adopted the Unified Process for MSHN development [4].

Although the Unified Process was very useful, it required some re-configuration. We, as researchers, have different types of requirements from those implementing production software. In the course of describing our experiences with semi-formal methods, we attempt to point out some of these differences and generalize them in a way that we hope will help other system researchers see the value in semi-formal software engineering methods and processes.

In this paper, we describe (1) the status of the MSHN project prior to our application of the Unified Process, and (2) the immediate improvements we were able to make in MSHN following the introduction of a semi-formal method.

2 State of the System Design Before Applying a Semi-Formal Method

The Management System for Heterogeneous Networks (MSHN) is a resource management system for heterogeneous, distributed computing systems that grew out of a previous research project called SmartNet. SmartNet is primarily a scheduling framework designed to assist resource management systems (RMS) in determining the placement of compute-intensive applications in networked systems containing a heterogeneous collection of high-performance computers [2].

MSHN's basic goal is to continuously determine the mapping of resources to applications in a dynamic, heterogeneous, distributed environment that provides the best quality of service to a dynamically changing set of applications. The MSHN project is expanding upon SmartNet by addressing the following issues: (i) how to handle shared resources such as networks and file servers, (ii) how to transparently monitor resource usage and availability, (iii) how to support adaptive applications, and (iv) how to deliver good quality of service to all applications. MSHN requires a flexible, component-based architecture that supports experimentation. MSHN's initial architecture is shown in Figure 1.

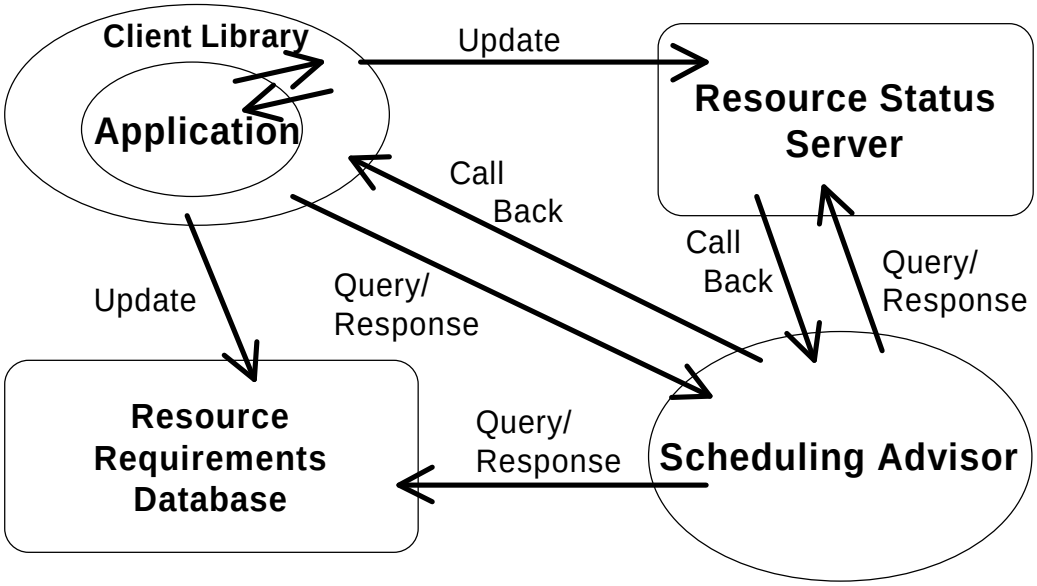

Figure 1. Initial MSHN Architecture.

The original MSHN architecture included four components: the Client Library, the Resource Status Server, the Resource Requirements Database, and the Scheduling Advisor. Each serves a specific function.

The Resource Status Server maintains a quickly changing repository of information pertaining to the current status of the resources that MSHN may assign to processes. The Resource Requirements Database is a database of specific applications' resource requirements, indexed by compute characteristics, a concept pioneered in SmartNet. The Client Library enables an application to transparently interact with the other MSHN components.

The Client Library transparently detects the status of the resources on which an application is running as well as the resource requirements of that application. It reports this information to the Resource Status Server and the Resource Requirements Database, respectively. The Client Library also intercepts requests to start a new application and queries the Scheduling Advisor, which decides where to run the application. It is the Scheduling Advisor's job to determine the best mapping of resources to applications, based upon the current status of the resources, the expected resource requirements of the application, and the requested application's quality of service requirements.

Once the Scheduling Advisor determines where a particular application should be started, the Client Library is responsible for starting the application. Additionally, the Scheduling Advisor sets up callbacks with the Resource Status Server and the Resource Requirements Database so that if any significant changes occur, the Scheduling Advisor will be notified. If the Scheduling Advisor re-computes a new schedule, it contacts the Client Libraries associated with the affected applications. The applications may then adapt to the new schedule in various ways, via their Client Library.

Several teams of people are working on the design and implementation of this project. Two groups, each consisting of a faculty member and several students, tackled the problem of proposing, designing and implementing heuristics that would determine schedules to meet different classes of quality of service requirements. Another group refined the design of the Scheduling Advisor and implemented a prototype of it, working closely with the first two groups. A fourth group examined the question of security as a quality of service attribute. The bulk of the remainder of MSHN's design was to be refined by the authors, along with several staff members and six graduate students. We decided early on that we would describe the interfaces to MSHN's components in IDL. We also decided at an early stage to use CORBA to facilitate interaction among the various components, which, by their very nature, were replicated and distributed. Many students proceeded independently researching individual issues within the MSHN system.

As this research progressed, a new component, the MSHN Daemon, was added, initially with the sole purpose of starting up new applications. Several investigators thought that there was an additional need to determine the status of resources on which no current MSHN tasks were executing, so that functionality was also considered for addition to the MSHN Daemon. Finally, some investigators recognized that the MSHN Daemon might also be responsible for sensing the status of resources that did have MSHN-assigned tasks executing on them.

Additionally, two students worked to develop a generalized application emulator in order to allow us to emulate real applications, on various platforms, without having to install software, purchase licenses, learn the systems, and, in some cases, obtain security clearances.

About halfway through the project, the first MSHN prototype was successfully demonstrated, along with many of the other projects that were also part of the same DARPA program. Based upon this demonstration, and subsequent delivery of documentation, MSHN and several other projects were identified as the basis for components to be combined into a system of systems. At this point, the integration group requested that the MSHN investigators supply a UML description of MSHN [1].

While "retrofitting" a UML description onto the existing MSHN architecture, we decided to also use UML to help us explain MSHN's components to others, including new students, staff members, as well as colleagues working on other systems. Although each of us thought that we understood the interactions between the MSHN components well, before we started documenting them in UML, we found that the UML provided an excellent framework for clarifying points to one another as well as improving our design.

3 Applying a Semi-Formal Method to the System Design

Before we applied any semi-formal methods to MSHN, we had built a prototype of each of the MSHN components. Additionally, to aid in debugging as well as

demonstrating MSHN, we had implemented a visualizer that we could use to graphically examine the internals of each of these components. Finally, we had an incomplete design of a MSHN Daemon and some thoughts as to how we were going to increase the functionality of each of our existing prototype components.

Applying a UML description to our existing project not only provided a great framework for communication, but also helped us learn that (i) we were not in total agreement as to eventual functionality, and (ii) our initial design could be improved. This section documents some of our design modifications and identifies some of the improvements that we made.

As would be expected, some of the most striking improvements were made to the components that were added after the initial design, the MSHN Daemon and the MSHN Application Emulator. Due to space constraints, and because there were no substantial modifications made to them, the MSHN Scheduling Advisor, Resource Status Server, and Resource Requirements Database are not described below. The interactions between all MSHN components are, however, illustrated inFigure 2. A good overview of MSHN can be found elsewhere[3].

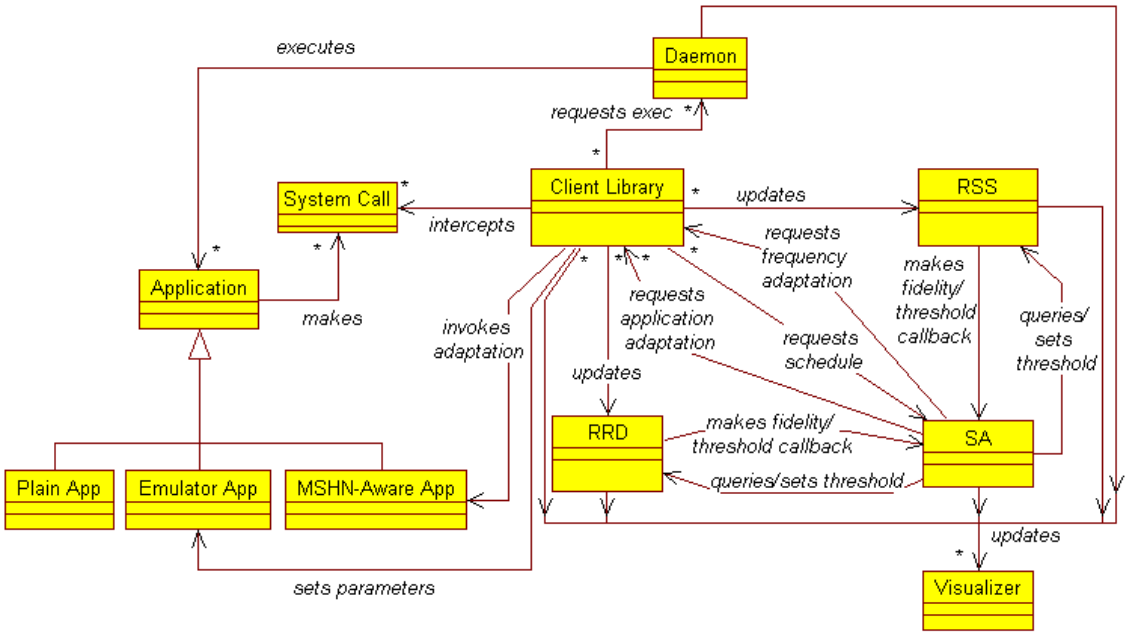

Figure 2. MSHN's Conceptual Model

In the previous section, we described the initial purpose for the MSHN Daemon as well as the functionality that several investigators believed should be embodied in it. Originally, a MSHN Daemon was to reside on each computing resource and was to be used to start up a new application when requested to do so by a Client Library. During design discussions, some members of the design team also wanted the Daemon to aid in determining the status of machines where no applications that were linked with the Client Library were running. These members also believed that the

Daemon may be better equipped to determine the status of some resources on machines that MSHN applications *were* using. The UML description provided a framework that enabled the designers to communicate with one another more clearly. UML descriptions of the MSHN Daemon, Client Library, and Application Emulator components, when simultaneously understood, made it easy to see that keeping the Daemon simple was the proper design decision.

The clarity, which the UML description provided, let us quickly see that the other required functionality, determining the status of resources not currently being used by applications that are linked with the MSHN Client Library, was better obtained by combining the Application Emulator, which had been originally designed for an altogether different purpose, with the Client Library. Interestingly, in these discussions, the Application Emulator became an integral part of the MSHN design, though it also continues to be important in its original role as well. The subsections below document the eventually agreed-upon requirements of the MSHN Daemon, the Client Library, and the Application Emulator and show why the minimally functional MSHN Daemon was the best decision.

Client Library functions. The Client Library provides a transparent interface between each application and the MSHN components. The Client Library transparently intercepts system calls. It collects resource usage and status information, which is forwarded to the Resource Requirements Database and Resource Status Server, respectively. The Client Library also intercepts calls that initiate new processes and consults the Scheduling Advisor for the best place to start each process. It requests (possibly remote) Daemons to execute applications based on the Scheduling Advisor's advice. The Client Library, when notified by the Scheduling Advisor via callbacks, invokes adaptation on MSHN-aware applications. Such adaptation is included under the special case of setting Application Emulator parameters.

Daemon functions. A copy of the MSHN Daemon runs on each compute resource available for use by the Scheduling Advisor. Its sole purpose is to start applications upon request from the Client Library.

Application Emulator functions. The MSHN Application Emulator emulates a running application by stressing particular resources in the same way that real applications do. The Application Emulator serves two purposes. The first (and originally-intended) purpose is as a way of running applications that statistically leave the same resource usage footprint as real applications without the overhead and uncertainty of actually installing, maintaining and running that particular application. The second purpose of the Application Emulator is to perform resource monitoring in the absence of any other MSHN-scheduled applications. Since resource monitoring can be viewed as a kind of application, the Daemon starts one instance of the Application Emulator on each machine, by default, at startup, to monitor resources. The Scheduling Advisor instructs the Application Emulator to monitor more or fewer resources, depending upon the resources that are currently being used and, hence, transparently monitored, by MSHN-scheduled applications.

As alluded to above, in the discussion that accompanies the requirements of the Application Emulator, we note one example of re-use that was discovered during this process, and of which we were completely unaware prior to this analysis. Another interesting point to note is the complexity of the Client Library. By comparing its complexity to that of other components, we were able to determine that, in order to complete that component, we would need to devote substantially more man-power than we had previously devoted to that component.

4 Conclusions

The Management System for Heterogeneous Networks (MSHN) is a large, ambitious, system-level research project. MSHN's major goal is to determine the best way to design and implement many features of a resource management system (RMS) that is able to deliver good quality of service to a mixture of applications with different requirements. Because MSHN is such a large, complex project, it requires the expertise and experience of many different investigators from distributed operating systems, resource management systems, computer architecture, theory of algorithms, control theory, stochastic processing, and dynamic programming.

Although MSHN's investigators had learned quite a lot from designing and implementing previous research resource management systems, their initial design of MSHN did not identify all of the components whose functionality deserved to be isolated and further investigated. One of those components, the MSHN Daemon, was identified because it was determined that implementation of a particular functionality, starting applications on new machines, would otherwise be too machine- and operating system-dependent. Another component, the Application Emulator, which later turned out to be integral to the overall functionality of MSHN, was designed initially for simply testing the system.

The Unified Modeling Language (UML) and a modified version of the Unified Process were applied halfway through the MSHN project, but still proved to be quite valuable. Several cases of object re-use and bloated packages were discovered. Perhaps, more importantly, UML and the Unified Process provided a framework that allowed the experts from the various fields to communicate more easily their experience to one another. We believe strongly that the MSHN project benefited from these semi-formal methods, and we intend to use them again in other projects, earlier and more often.

5 References

[1] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Addison Wesley, Reading, MA, 1999.

[2] R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Kieth, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, "Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet," *Proc. 7th IEEE Heterogeneous Computing Workshop*, March 1998, pp. 184-199.

[3] D. A. Hensgen, T. Kidd, D. St. John, M. C. Schnaidt, H. J. Siegel, T. D. Braun, M. Maheswaran, S. Ali, J. Kim, C. Irvine, T. Levin, R. F. Freund, M. Kussow, M. Godfrey, A. Duman, P. Carff, S. Kidd, V. Prasanna, P. Bhat, and A. Alhusaini, “An overview of MSHN: the Management System for Heterogeneous Networks,” *8th Heterogeneous Computing Workshop* (*HCW '99*), April 1999.

[4] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*, Addison Wesley, Reading, MA, 1999.