2002-12

Dynamic Assembly for System Adaptability, Dependability, and Assurance

Luqi

Naval Postgraduate School

Luqi, "Dynamic Assembly for System Adaptability, Dependability, and Assurance, (DASASA) Project", Technical Report, NPS-SW-013, December 2002.
https://hdl.handle.net/10945/65208

NPS-SW-02-013

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Dynamic Assembly for System Adaptability, Dependability, and Assurance (DASADA) Project

Progress Report (10/01/2001–9/30/2002)

By

Luqi

September 2002

Prepared for: Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington, VA. 22203-1714

REPORT DOCUMENTATION PAGE

Form Approved
OMB NO. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	9/30/2002	Progress Report 10/01/2001 – 9/30/2002

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Dynamic Assembly for System Adaptability, Dependability, and Assurance (DASADA) Project – Progress Report (10/01/2001 – 9/30/2002)	01-K962
6. AUTHOR(S) Professor Luqi	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Automation Center, Naval Postgraduate School, Monterey, CA 93943	8. PERFORMING ORGANIZATION REPORT NUMBER NPS-SW-02-013
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency 3701 North Fairfax Drive Arlington, VA. 22203-1714	10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.

12 a. DISTRIBUTION / AVAILABILITY STATEMENT	12 b. DISTRIBUTION CODE
Approved for public release; distribution unlimited.	

13. ABSTRACT (Maximum 200 words)

The objective of our effort is to analyze the research results of DASADA program, to provide recommendations to the Program Manager on the merits of new software engineering technologies and their possible integration with respect to future Department of Defense (DoD) systems, and to facilitate the transfer of DASADA technologies to DoD users. Our work focuses on applying DASADA technologies to the areas including rapid reconfigurable weapon software architecture, software module interoperability, COTS interaction and collecting reliable measures for predicting software development time by probes. For this purpose, we conducted research on MetaH avionics architecture description language and developed models and methods for solving the integration and interoperability problems in component-based distributed heterogeneous systems. Moreover, we developed a modified model to predicate software development time and assess the risk on the basis of reliable measures collected in early phases of software development. We also educated DoD engineers and military officers on DASADA technologies via distance learning and integrated concepts and technologies of DASADA into some courses such as SW4599 and SW4582.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Architecture Description Language, Interoperability, COTS integration, Risk Assessment	77
	16. PRICE CODE

17. SECURITY CLASSIFICATION OR REPORT	18. SECURITY CLASSIFICATION ON THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev.2-89)
Prescribed by ANSI Std. 239-18
298-102

Table of Contents

NAVAL POSTGRADUATE SCHOOL
Monterey, California 93943-5000

RADM David R. Ellison
Superintendent

Richard S. Elster
Provost

This report was prepared for and funded in part by the Defense Advanced Research Projects Agency.

This report was prepared by:

Luqi
Professor, Computer Science

Reviewed by:

Luqi
Director, Software Engineering
Automation Center

Released by:

D. W. Netzer
Associate Provost and
Dean of Research

Progress Report

Dynamic Assembly for System Adaptability, Dependability, and Assurance (DASADA) Project

10/01/2001-9/30/2002

1. Statement of the Problem Studied

The DoD is aware that as software becomes more complex, it will become extremely critical to have the ability for components to change themselves by swapping or modifying components, changing interaction protocols, or changing its topology. The Defense Advanced Research Programs Agency formed the Dynamic Assembly for Systems Adaptability, Dependability, and Assurance (DASADA) program in order to task academia and industry to develop dynamic gauges that can determine run-time composition, allow for the continual monitoring of software for adaptation, and ensure that all user defined properties remain stable before and after composition and deployment. The objectives of this project are to analyze the research results of DASADA program, to provide recommendations to the Program Manager on the merits of new software engineering technologies and their possible integration with respect to future Department of Defense (DoD) systems, and to facilitate the transfer of DASADA technologies to DoD users.

In FY01, we have accomplished following tasks:

- Conducted critical study and review of the 19 DASADA projects,
- Educated DoD engineers and military officers on DASADA technologies via distance learning,
- Conducted in-depth case study of one the EDP programs,
- Developed checklist and template for DASADA technology evaluation,
- Developed a guide to help DoD managers to select software metrics in acquiring new technologies for weapon systems software.

2. Summary of tasks accomplished in FY02

- Educated DoD engineers and military officers on DASADA technologies via distance learning.

- Integrated concepts and technologies of DASADA into some SW courses
  - SW4599 – Automated Hardware/Software Integration in DOD
  - SW4582 – Weapon Systems Software Safety
- Conducted research on MetaH avionics architecture description language.
- Developed models and methods for solving the integration and interoperability problems in component-based distributed heterogeneous systems.
- Developed a modified model to predicate software development time and assess the risk on the basis of reliable measures collected in early phases of SW development.

3. Highlights of important results

In FY02, we have made lots of efforts on educational programs to propagate the DASADA technologies to the military students and direct doctoral and master students to apply DASADA technologies to DoD projects. We offer two courses (SW4599 and SW4582) to integrate the concepts and technologies of DASADA.

SW 4599--- Automated Software/Hardware Integration in DoD

Automated software/hardware integration is a key problem for current software development in DoD. This course covers some important aspects of this field, including software prototyping, interface integration, data integration, and control integration. Automatable decision support methods for software/hardware integration are also discussed.

SW 4582--- Weapon Systems Software Safety

This course treats software safety from a systems perspective. It contributes to introduce the system safety technologies to the DoD officers. This course addresses the topic of safety along three dimensions: systematic assessment and management of risk, designing safety into a system starting with system conceptualization, and applying system safety theory, principles, techniques, and tools to build safety cases for the purposes of certifying and accrediting software for use in safety-critical applications and infrastructures.

We conducted in-depth study of DASADA technologies include follows:

- MetaH (modeling, timing analysis),
- UNCLE (constraint consistency gauges),
- QRAM (resource allocation gauges),
- IMPACT (system load tracking and visualization),
- SIM-TABASSCO (Semantic Interoperability Measures: Template-Based Assurance of Semantic Interoperability in Software Composition),
- Veridian Pacific-Sierra Research (terrain-reasoning software being reconfigured via the Venice tool),
- Proteus (run time and design time gauges for alternate architecture deployment).

On the basis of the study of these technologies, we directed our doctoral and master students to conduct a series of research efforts on the areas including rapid reconfigurable weapon software architecture, software module interoperability, COTS integration and SW development time predicting.

Research on MetaH avionics architecture description language

We conducted the study to identify ways in which the emerging standard SAE Avionics Architecture Description Language (AADL) and associated tools and methods can beneficially support avionics safety and airworthiness certification.

We gave preliminary recommendations and guidelines for extension and use of the AADL and supporting tools and methods for avionics system safety assurance and avionics system design assurance.

We also conducted a comparative survey of FAA and Army avionics airworthiness certification processes, guidelines, standards and methods. Acquisition reform and the MilSpec Reform initiative encourage the use of commercial standards, processes and products wherever suitable. An improved ability to reuse civil avionics can reduce military acquisition costs. An AADL toolset suitable for both civil and military applications can have its cost shared across a larger market. Consequently, this study evaluated both civil and Army airworthiness certification regulations, guidelines, standards and methods.

Research on software module interoperability and COTS integration

Meeting future system requirements by integrating existing stand-alone systems is attracting renewed interest. Computer communications advances, functional similarities in related systems, and enhanced information description mechanisms suggest that improved capabilities may be possible; but full realization of this potential can only be achieved if stand-alone systems are fully interoperable. Interoperability among independently developed heterogeneous systems is difficult to achieve: systems often have different architectures, different hardware platforms, different operating systems, different host languages and different data models.

We have developed the Object-Oriented Model for Interoperability (OOMI) to capture the information required for resolving modeling differences in a federation of independently developed heterogeneous systems. In this study, a model of the information and operations shared among systems, termed a Federation Interoperability Object Model (FIOM), is defined so that defining the interoperation between systems in terms of an object model provides a foundation for easy extension as new systems are added to the existing federation of systems. Construction of the FIOM is done prior to run-time with the assistance of a specialized toolset, the OOMI Integrated Development Environment (OOMI IDE). Then at runtime, OOMI translators utilize the FIOM to automatically resolve differences in exchanged information and in inter-system operation signatures. This study provides an efficient way to integrate existing stand-alone systems and enable the software module interoperability for many DoD large-scale applications.

Legacy software systems in the Department of Defense (DoD) have been evolving and are becoming increasingly complex while providing more functionality. The shortage of original software designs, lack of corporate knowledge and software design documentation, unsupported programming languages, and obsolete real-time operating system and development tools have become critical issues for the acquisition community. Consequently, these systems are now very costly to maintain and upgrade in order to meet

current and future functional and nonfunctional requirements.

We addressed the issue of interoperability in DoD legacy system databases and evaluated XML as a tool for transferring message data between varied systems.

With the demands for increased communication, the dire requirement for a common mode of information transfer is greatly realized. Many legacy systems have developed their own unique interfaces. XML is one solution which can help ease the transition to a common interface. A software program was developed to generate select messages in their native and XML formats.

We proposed a new interoperability model for re-engineering of old procedural software of the Multifunctional Information Distributed System Low Volume Terminal (MIDS-LVT) to a modern object-oriented architecture. In the MIDS-LVT modernization acquisition strategy, only one Computer Software Configuration Item (CSCI) at a time will be redesigned into an object-oriented program while interoperability with other unmodified CSCIs in the MIDS-LVT distributed environment must be maintained. Using this model, each legacy CSCI component can be redesigned independently without affecting the others.

Research on methods for predicting SW development time and assessing the risk

Risk management is most effective in impacting the project's success if project risks are identified and mitigated early in the software lifecycle. We developed a set of metrics for risk assessment and development efforts predicting. These metrics can be automatically collected from early phases of the life cycle of software development. Based on the metrics, the Modified Risk Model was developed. Additionally, the Modified Risk Model is versatile enough to be adapted to any software development activity.

The Modified Risk Model is a macro model developed to aid program managers in effectively planning the required effort to deliver software products. The model projects the probability of completing a software project, subject to the available resources supplied by management. Inverse, when given the probability of completing a software

project, this model can predicate the required efforts of SW development. This approach to software project risk management is unique because the model's input parameters are derived. Different program managers would derive the same projections on the same software project.

4. Deliverables

- Dissertations and Theses
  1) Michael R. Murrah, Enhancements and Extensions of Formal Models for Risk Assessment in Software Projects, Ph.D. dissertation, Naval Postgraduate School, Monterey, CA, September 2002.
  2) Paul E. Young, Heterogeneous Software System Interoperability Through Computer-Aided Resolution of Modeling Differences, Ph.D. dissertation, Naval Postgraduate School, Monterey, CA, July 2002.
  3) Shong Cheng Lee, Class Translator for the Federation Interoperability Object Model (FIOM), Master's thesis, Naval Postgraduate School, Monterey, CA, March 2002.
  4) Kris Pradeep, XML As A Data Exchange Medium For DoD Legacy Databases, Master's thesis, Naval Postgraduate School, Monterey, CA, March 2002.
  5) Theng C. Moua, Application Programmer's Interface (API) for Heterogeneous Language Environment and upgrading the Legacy Embedded Software, Master's thesis, Naval Postgraduate School, Monterey, CA, March 2002.

- Papers
  1) Luqi, Ying Qiao, Lin Zhang, "Computational Model for High-confidence Embedded System Development", Monterey Workshop 2002--- Radical Innovations of Software and Systems Engineering in the Future, Venice, Italy, October, 7-11, 2002.
  2) J. Drummond, Luqi, W. Kemple, M. Auguston, and N. Chaki. "Quality of Service Behavioral Model from Event Trace Analysis." *Proceedings of the 7th international Command and Control Research and Technology Symposium (CCRTS 2002)*, Quebec City, Quebec, 16-20 September 2002.
  3) W. Ray, Luqi, and V. Berzins. "Optimizing Systems by Work Schedules (a Stochastic Approach)." *Proceedings of the Workshop on Software Performance (WOSP 2002)*, Rome, Italy, 23-26 July 2002.

4) P. Young, V. Berzins, J. Ge, and Luqi. "Using an Object Oriented Model for Resolving Representational Differences between Heterogeneous Systems." *Proceedings of 17th ACM Symposium on Applied Computing*, Madrid, Spain, 10-14 March 2002.

5) W. Ray, Realizing Adaptive Systems, 17th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications, Washington State Convention & Trade Center, Seattle, Washington, USA, November 4-8, 2002.

Heterogeneous Software System Interoperability Through Computer-Aided Resolution of Modeling Differences

Paul E. Young (Ph. D)

Meeting future system requirements by integrating existing stand-alone systems is attracting renewed interest. Computer communications advances, functional similarities in related systems, and enhanced information description mechanisms suggest that improved capabilities may be possible; but full realization of this potential can only be achieved if stand-alone systems are fully interoperable. Interoperability among independently developed heterogeneous systems is difficult to achieve: systems often have different architectures, different hardware platforms, different operating systems, different host languages and different data models.

The Object-Oriented Method for Interoperability (OOMI) introduced in this dissertation resolves modeling differences in a federation of independently developed heterogeneous systems, thus enabling system interoperation. First a model of the information and operations shared among systems, termed a Federation Interoperability Object Model (FIOM), is defined. Construction of the FIOM is done prior to run-time with the assistance of a specialized toolset, the OOMI Integrated Development Environment (OOMI IDE). Then at runtime OOMI translators utilize the FIOM to automatically resolve differences in exchanged information and in inter-system operation signatures.

Enhancements and Extensions of Formal Models for Risk Assessment in Software Projects

Michael R. Murrah (Ph. D)

The Modified Risk Model is a macro model developed to aid program managers in effectively planning the required effort to deliver software products. The model projects the probability of completing a software project, subject to the available resources supplied by management. This approach to software project risk management is unique because the model's input parameters are derived. Subjective variables are not part of the model. Different program managers would derive the same projections on the same software project.

Risk management is most effective in impacting the project's success if project risks are identified and mitigated early in the software lifecycle. The Modified Risk Model was developed specifically for this purpose. Additionally, the Modified Risk Model is versatile enough to be adapted to any software development activity.

Validation of the model occurs in approximately 2,000 software projects. During these preliminary experiments, the Modified Risk Model out performed the macro models of Basic COCOMO and the Simplified Software Equation. However, to date, operational tests have not been conducted on the model.

The Modified Risk Model requires four parametric inputs, all of which are automatically collectable and derived extremely early in the software lifecycle:

- **Organization.** The MRM implements a measure to capture the efficiency of a software development organization.
- **Complexity.** The MRM architecture accommodates interface with the Computer Aided Prototyping System developed at the Naval Postgraduate School. (Dupo02) and this research are capable of deriving key complexity measures from the machine generated specification code. The MRM is capable of using different complexity measures as a "plug- ins"; thus, allowing the model to interface with organizations not equipped with CAPS.
- **Requirements.** A software project can be viewed as a finite set of issues that require resolution prior to project completion. These issues are not fully revealed in the beginning of the process. The MRM captures the stability of the known issues and adjusts projections based on the introduction or deletion of additional issues. As with the other model parameters, requirements volatility is completely adaptable to unique software development situations. A risk analyst can choose to monitor the change in the project's risk or implement static projections.
- **Management Trade-Offs.** To successfully develop software, a balance must exist between the organization *(efficiency)*, product attributes *(complexity)*, and project stability *(requirements volatility)*. In reality, this is not always the case. It becomes the responsibility of management to balance the equation. Management applies resources (time and people) to achieve a successful balance.

The Modified Risk Model lets management know how well balanced is the software development. The risk analyst also has the ability to derive the *Management Trade-Offs* within a confidence interval. With this information, management can implement any suitable staffing profile to achieve the model's projection.

Class Translator for the Federation Interoperability Object Model (FIOM)

Lee, Shong Cheng (Master)

There is a growing need for systems to inter-operate in order to facilitate information sharing and to achieve objectives through joint task executions. The differences in data representation between the systems greatly complicate the task of achieving interoperability between them. Young's Object Oriented Method for Interoperability (OOMI) defines an architecture and suite of tools to resolve representational differences between systems. The OOMI architecture and tool suite will reduce the labor-intensity and complexity of the integration of disparate systems into a cooperative system of systems (federation of systems) and their subsequent deployment. At the heart of this architecture is the definition of translations between any two different classes of objects and a run-time component (the *Translator*) that will execute such translations.

This thesis describes a prototype framework that implements the OOMI, a prototype class translation code generator that assists an Interoperability Engineer in the definition of the translations and a prototype *Translator* that executes these translations.

XML As A Data Exchange Medium For DoD Legacy Databases

Kris Pradeep (Master)

This thesis addresses the issue of interoperability in DoD legacy system databases and evaluates XML as a tool for transferring message data between varied systems.

With the demands for increased communication, the dire requirement for a common mode of information transfer is greatly realized. Many legacy systems have developed their own unique interfaces. XML is one solution which can help ease the transition to a common interface.

This thesis is a part of a larger team effort. In contributing to this larger effort, a software program was developed to generate select messages in their native and XML formats.

Application Programmer's Interface (API) for Heterogeneous Language Environment and upgrading the Legacy Embedded Software

Theng C. Moua (Master)

Legacy software systems in the Department of Defense (DoD) have been evolving and are becoming increasingly complex while providing more functionality. The shortage of original software designs, lack of corporate knowledge and software design documentation, unsupported programming languages, and obsolete real-time operating system and development tools have become critical issues for the acquisition community. Consequently, these systems are now very costly to maintain and upgrade in order to meet current and future functional and nonfunctional requirements.

This thesis proposes a new interoperability model for re-engineering of old procedural software of the Multifunctional Information Distributed System Low Volume Terminal (MIDS-LVT) to a modern object-oriented architecture. In the MIDS-LVT modernization acquisition strategy, only one Computer Software Configuration Item (CSCI) at a time will be redesigned into an object-oriented program while interoperability with other unmodified CSCIs in the MIDS-LVT distributed environment must be maintained. Using this model, each legacy CSCI component can be redesigned independently without affecting the others.

Formal Specification of Generative Component Assembly Using Two-Level Grammar *

Barrett R. Bryant
Computer and Information Sciences
University of Alabama at Birmingham
Birmingham, AL 35294-1170, U. S. A.
bryant@cis.uab.edu

Mikhail Auguston
Computer Science
New Mexico State University
Las Cruces, NM 88003, U. S. A.
mikau@cs.nmsu.edu

Rajeev R. Raje Andrew M. Olson
Computer and Information Science
Indiana University Purdue University Indianapolis
Indianapolis, IN 46202, U. S. A.
{rraje, aolson}@cs.iupui.edu

Carol C. Burt
Computer and Information Sciences
University of Alabama at Birmingham
Birmingham, AL 35294-1170, U. S. A.
cburt@cis.uab.edu

Abstract

Two-Level Grammar (TLG) is proposed as a formal specification language for generative assembly of components. Both generative domain models and generative rules may be expressed in TLG and these specifications may be automatically translated into an implementation which realizes an integration of components according to the principles of the Unified Metacomponent Model (UMM) and Unified Approach (UA) to component integration. Furthermore, this implementation realizes Quality of Service (QoS) guarantees by means of static QoS verification at the time of system assembly, and dynamic QoS validation on a set of test cases.

1. Introduction

The recent shift in the focus of OMG (Object Management Group) to "Model Driven Architecture" (MDA) [17] is a recognition that to create mechanized software and bridging of component architectures requires standardization not only of infrastructure but also Business and Component Meta-Models. This emphasizes the fact that a comprehensive meta-model, that seamlessly encompasses heterogeneous components by capturing their necessary aspects including Quality of Service (QoS) and associated guarantees, is needed for creating future generation of distributed systems.

The UniFrame project proposes a unified metacomponent model (UMM) [18] for distributed component-based systems, and a Unified Approach (UA) [19] for integrating these components. The core parts of the UMM are: *components, service and service guarantees*, and *infrastructure*. UMM provides an opportunity to bridge gaps that currently exist in the standards arena. The creation of a software solution for a distributed computing system (DCS), using UA, has two levels: a) component level – developers create components, test and validate the appropriate QoS and deploy the components on the network, and b) system level – a collection of components, each with a specific functionality and QoS, and a semi-automatic generation of a software solution for a particular DCS is achieved.

The basis for the automatic generation of software is a Generative Domain Model (GDM) as employed in Generative Programming [9]. This model consists of two parts: a problem space and a solution space. The former is the collection of concepts and features that occur in an application domain, such as a particular kind of business, and that determine the nature of problems in the domain. These may be expressed in various ways, but common ones are UML, ER and feature diagrams. A corresponding solution

*This material is based upon work supported by, or in part by, the U. S. Army Research Laboratory and the U. S. Army Research Office under contract/grant numbers DAAD19-00-1-0350 and 40473-MA, and by the U. S. Office of Naval Research under award number N00014-01-1-0746.

space is a collection of specifications of software systems that present solutions to the problems in the problem space. For a software system constructed out of components, as envisioned here, such a specification will be expressed in terms of a collection of specifications for standardized components, which are supplied by vendors. These specifications must also include configuration knowledge, which describes how components may be combined or depend upon one another and how a system is constructed from its constituent components.

Component development and deployment starts with a UMM requirements specification of a component from a particular domain. This specification is natural language-like and indicates the functional (i.e., computational) and non-functional (i.e., QoS parameters) features of the component. This specification is then refined into a formal specification, based upon the theory of Two-Level Grammar (TLG) [7]. Both generative domain models and generative rules may be expressed in TLG and these specifications may be automatically translated into an implementation which realizes an integration of components.

This paper is organized as follows. Section 2 describes the Two-Level Grammar specification language and section 3 describes the principles of the Unified Approach to system assembly from components. In section 4, a case study is presented illustrating how these principles are realized. Finally we conclude in section 5.

2. Two-Level Grammar

Two-Level Grammar (TLG) was originally developed as a specification language for programming language syntax and semantics [8], and later used as an executable specification language and as the basis for conversion from requirements expressed in natural language into a formal specification [6]. TLG is a formal notation based upon natural language and the functional, logic, and object-oriented programming paradigms. TLG allows queries over the knowledge base, such as a problem or solution domain, to be expressed in a natural language-like manner which is consistent with the way in which UMM is expressed. TLG is then a framework under which a natural language may be used to both describe and inquire about the nature of components and systems, while maintaining the formalism of formal specification languages. The combination of natural language and formalization is unique to TLG and also fits UMM well.

The name "two-level" in Two-Level Grammar comes from the fact that TLG consists of two context-free grammars defining the set of type domains and the set of function definitions operating on those domains, respectively. These grammars may be defined in the context of a class in which case type domains define instance variables of the class and function definitions define methods of the class. The syntax of TLG class declarations is:

class *Identifier-1* [extends *Identifier-2*, ... *Identifier-n*].
  instance variable and method declarations
end class [*Identifier-1*].

Identifier-1 is declared to be a class which inherits from classes *Identifier-2*, ..., *Identifier-n*. In the above syntax, square brackets are used to indicate the `extends` clause is optional so a class need not inherit from any other class. The instance variables comprising the class definition are declared using domain declarations of the following form:

Identifier-1, ..., Identifier-m ::
  `data-object-1; ...; data-object-n.`

where each `data-object-i` is a combination of domain identifiers, singleton data objects, and lists of data objects, which taken together as a union form the type of *Identifier-1, ..., Identifier-m*. Syntactically, domain identifiers are capitalized, and singleton data objects are finite lists of natural language words written entirely in lower case letters. For improved readability, domain identifiers are italicized and data objects are represented in typewriter font. Predefined types include *Integer*, *Boolean*, *Character*, *String*, lists, sets, bags, and mappings. Postfix operators * and + may also be used to define lists of zero or more and one or more elements, respectively.

Function definitions comprise the operational part of a TLG specification. Their syntax allows for the semantics of the function to be expressed using a structured form of natural language. Function definitions take the forms:

function signature.
function signature :
  function-call-1, ..., function-call-n.

where n≥1. Function signatures are a combination of natural language words and domain identifiers. For improved readability, we will use boldface type to represent the function keywords. Domain identifiers in the context of a function typically correspond to variables in a conventional logic program. Some of these variables will typically be input variables and some will be output variables, whose values are instantiated at the conclusion of the function call. Therefore, func-

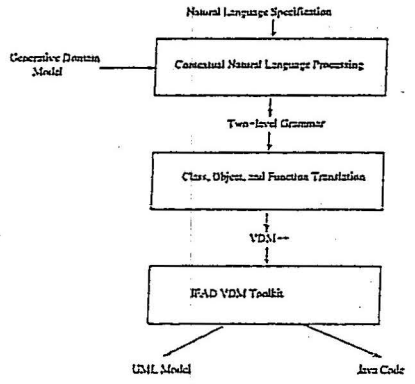

Figure 1. Two-Level Grammar Implementation

tions usually return values through the output variables rather than directly, in which case the direct return value is considered as a Boolean `true` or `false`. `true` means that control may pass to the next function call, while `false` means the rule has failed and an alternative rule should be tried if possible. Besides Boolean values, functions may return regular values, usually the result of arithmetic calculations. In this case, only the last function call in a series should return such a value.

Methods of class objects are called by writing a sentence or phrase containing the object. The result of the method call is to instantiate the logical variables occurring in the method definition. In any class for every instance variable of simple type there are get and set methods to access or modify that variable.

TLG is implemented as part of a specification development environment which facilitates the construction of TLG specifications from natural language using a domain knowledge base structured as a Generative Domain Model (GDM) expressed in XML (eXtensible Markup Language) [5], and then translates TLG specifications into executable code. The natural language requirements are translated into a contextual knowledge representation [15] which may then be expressed using TLG. The TLG is then translated into VDM++ [10], the object-oriented extension of the Vienna Development Method (VDM) specification language [13]. The *IFAD VDM Toolbox*™ [11] may then be used to generate code in an object-oriented programming language such as Java or C++, as well as a UML (Unified Modeling Language) model. The overall description of this process is described in Figure 1. Further details of the implementation are given in [14].

3. Unified Meta-Component Model (UMM) and Unified Approach (UA)

In general, different developers will provide on the Internet a variety of possibly heterogeneous components oriented towards a specific problem domain. Once all the components necessary for implementing a specified distributed system are available and a specific problem is formulated, then the task is to assemble them into a solution. The UA assumes that the generation environment is built around a generative domain-specific model supporting component-based system assembly. The distinctive features of the approach are as follows:

- The developer of the desired distributed system presents to this process a system query, in a structured form of natural language, that describes the required characteristics of the distributed system. The query is processed using the domain knowledge (such as key concepts from a domain) and a knowledge-base containing the UMM description (in the form of a TLG) of the components for that domain. From this query a set of search parameters is generated which guides "head-hunter" agents for a component search in the distributed environment. Head-hunters serve to locate the components which are needed to complete the requested system [22].
- The framework, with the help of the infrastructure, collects a set of potential components for that domain, each of which meets the Quality of Service (QoS) requirements specified by the developer. QoS requirements are expressed in terms of a catalog of parameters established for this purpose [4]. After the components are fetched, the system is assembled according to the generation rules embedded in the generative domain model. Essentially, the generated code constitutes the glue/wrapper interface between the components. Since TLG may be used to provide for attribute evaluation and transformation, syntax and semantics processing of languages, parsing, and code generation, the TLG formalism is used to specify the generative rules and the output of the TLG will provide the desired target code (e.g., glue and wrappers for components and necessary infrastructure for distributed run-time architecture). All of this is implemented according to the process for translating TLG specifications into executable code as described earlier.
- Along with the generated system will be a formal UMM specification of the generated system so that

it may be used in subsequent assemblies. This *formal UMM specification* will also be a basis for generating a set of test cases to determine whether or not an assembly satisfies the desired QoS.

- Static QoS parameters (e.g. dependability of the component) are processed during generation time and hence will be processed by the TLG directly. Dynamic QoS parameters (e.g. response time ofthe component) result in instrumentation of generated target code based on event grammars [1, 2], which at run time will produce the corresponding QoS dynamic metrics which may be measured and validated.

To summarize, the inputs for the system assembly and generation step are: the query for the system build, UMM descriptions of the components found by headhunters, and the QoS parameters for the system build. The outputs are the generated code instrumented for the dynamic QoS metric evaluation and auxiliary code needed to compile, assemble and run the system, and UMM description of the generated system which makes it possible to add the new component to the component database. TLG is the formalism for representing UMM's, GDM's, QoS parameters, supporting queries, and generation rules. Only the queries that have counterparts in the GDM are processed. The GDM contains generation rules for system assembly from the components. The query language is an essential part of the approach since the query provides the input for component search via the headhunter mechanism and following glue and wrapper generation. The query supplies the initial parameters for the headhunters to search in the distributed environment and gives the input for the generation step itself.

QoS parameters given in the query provide yet another aspect for the generated code - the instrumentation necessary for the run-time QoS metrics evaluation. Based on the query or informal requirements, the user has to come up with a representative set of test cases. Next the implementation is tested using the set of test cases to verify that it meets the desired QoS criteria. If it does not, it is discarded. After that, another implementation is chosen from the component collection. This process is repeated until an optimal (with respect to the QoS) implementation is found, or until the collection is exhausted. In the latter case, the process may request additional components or it may attempt to refine the query by adding more information about the desired solution from the problem domain. If a satisfactory implementation is found, it is ready for deployment. The complete view of this system is shown in Figure 2.

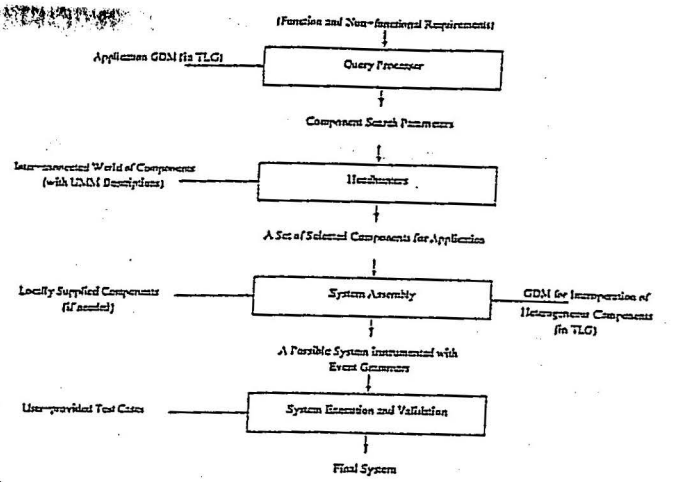

Figure 2. System Assembly in UniFrame

A few attempts have been made to incorporate QoS (quality of service) into component-based software systems. The Aster project [12] uses architectural descriptions of components and their interactions, including non-functional properties, to customize middleware. Quality Objects (QuO) [3] is a framework for providing QoS to software applications composed of objects distributed over wide area networks. QuO bridges the gap between socket-level QoS and distributed object level QoS, emphasizing specification, measuring, controlling, and adapting to changes in QoS. RAPIDware [16] is an approach to component-based development of adaptable and dependable middleware. It uses rigorous software development methods to support interactive applications executed across heterogeneous networked environments. Process^{NFL} [21] is a language for describing non-functional properties of software, which may include QoS properties. The Unified Approach is concerned not only with specifying QoS properties of components, but also to assure satisfaction of these properties in an implementation resulting from assembling the components. It should be noted that the assurance of QoS (as described above) indicates that a component can guarantee appropriate values for its QoS parameters in an 'ideal' situation. This does not guarantee that a component will be able to either provide this QoS under failure circumstances or will automatically adjust its QoS to hide the failures. For the failure situations, the ideas provided by Aster, QuO, or RAPIDware can be incorporated into UMM and UA.

4. A Case Study

This section describes a simple example of a bank account management system in order to illustrate some

of representation features of the UA. The specification of bank accounts should include its attributes and the operations it should perform, such as `check balance`, `deposit`, or `withdraw`. This information may be expressed by the following feature model in Two-Level Grammar:

class *BankAccount*.
  AccountNumber, *PIN* :: *String*.
  Balance :: `float`.
  check balance.
  deposit *Float*.
  withdraw *Float*.
end class.

Assume that the GDM in this example contains a rule for system assembly, as mentioned in Section 3, that specifies that a Bank Account Management System consists of one of each of the two component types, *AccountServer* and *AccountClient*, each of which has an attribute of type *BankAccount*. For this example, let there be two instances of *AccountServer* and one instance of *AccountClient*. Server components are heterogeneous – `JavaAccountServer` adheres to the Java-RMI model; while `CorbaAccountServer` uses the CORBA model. The client, `JavaAccountClient` is developed by using the Java-RMI model and is implemented as an applet. The goal is to assemble a bank account management system from these available components. The UMM descriptions of these components would indicate their relevant properties, including: 1) the interface of the `JavaAccountServer`:

```
void javaDeposit  (float ip);
void javaWithdraw (float ip)
  throws overDrawException;
float javaBalance ();
```

and QoS parameters *Availability* $\geq$ 85% and *Response Delay* $<$ 30*ms*, 2) the interface of the `CorbaAccountServer`:

```
void corbaDeposit  (float ip);
void corbaWithdraw (float ip)
  throws overDrawException;
float corbaBalance ();
```

and QoS parameters *Availability* $\geq$ 90% and *Response Delay* $<$ 10*ms*, and 3) the interface of the `JavaAccountClient`:

```
void depositMoney  (float ip);
void withdrawMoney (float ip);
float checkBalance ();
```

and QoS parameters *Availability* $\geq$ 90% and *Response Delay* $<$ 50*ms*. The complete UMM specifications for these components are described in [20].

Queries are stated in a structured form of natural language and then processed into TLG. The general form of a query is to request creation of a system that has certain QoS parameters. The name of the system is important in identifying the application domain and the QoS parameters should also follow the catalog standards. A sample query for the above example can be informally stated as: *Create a bank account management system that has availability* $\geq$ *50% and response delay* $<$ *100 ms*. This query requires the satisfaction of one static and one dynamic QoS parameter. From the query and the available knowledge in the GDM associated with the bank account management systems, a query will be formulated for a headhunter in the UMM. In response, the headhunter will discover the three components and their QoS properties. Note that the availability QoS parameter is used to screen potential components at the time they are retrieved. The catalog specification for this parameter suggests that the availability criteria should be multiplied, so the availability of the Java-Java system is 76.5% and for the Java-CORBA system 81%, both meeting the stated criteria. The process of locating components through the head-hunter mechanism is described further in [22].

Two-Level Grammar is used as the formalism for both the UMM and generative rules. The UMM formalization establishes the context for which the generative rules may be applied. The TLG functions include generative rules for construction of the wrapper/glue code and the event grammar instrumentation to assure the QoS of the bank account record management system. The GDM for bank account management systems will be described according to this template, including both generation rules and QoS parameter processing.

A sampling of TLG rules which may be used to generate the appropriate glue/wrapper code to connect the components of the bank account management system is presented below. These rules are based on selecting from the GDM of the bank account management systems the appropriate system model for this two-component DCS. We first give type domain definitions.

ClientUMM, *ServerUMM* :: *UMM*.
ClientOperations, *ServerOperations* :: {*Interface*}*.

The *ClientUMM* would be the UMM specification of `JavaAccountClient` presented previously and the *ServerUMM* would be the UMM specification of `JavaAccountServer` or `CorbaAccountServer`. *ClientOperations* and *ServerOperations* are defined as a list of *Interface*'s. We assume that types *UMM* and *Interface* have already been defined.

The generation rule to produce Java code for two UMM models representing a client and server, respectively, is shown below. This rule is expressed using a TLG function which has a signature followed by a set of rules (or subfunctions) to be executed when the function is called. Function keywords are indicated in bold font.

```
generate system from ClientUMM and ServerUMM :
    ClientOperations := ClientUMM get operations,
    ServerOperations := ServerUMM get operations,
    OperationMapping :=
        map ClientOperations into ServerOperations,
    ComponentModel :=
        ServerUMM get component model,
    generate java code for OperationMapping
        using ComponentModel.
```

The main tasks are to map client operations onto server operations, e.g., `depositMoney` in `JavaAccountClient` maps to `corbaDeposit` in `CorbaAccountServer` or to `javaDeposit` in `JavaAccountServer`, and then generate the code to implement this mapping. The generated code will be in Java since the client code is in Java and must seamlessly interface with it. If the client is in C++ or other language, similar rules will be defined and many rules will be language independent.

The actual mapping to be defined will be based upon a natural language analysis of the names of operations. The closer the names match the vocabulary in the feature model, the more easily the system can establish the correct mapping. This depends upon both the care and style with which the user has written the interface method names and so may vary widely. For this example, it can be seen that the correspondence between names, while not exact, is relatively close.

The next set of rules describes the specifics of generating CORBA code in Java to implement the mapping that arises by integrating the `JavaAccountClient` with the `CorbaAccountServer`. The generated code is distinguished from types (variables) and function keywords by using a typewriter font.

```
CorbaPackageName :: String
CorbaObjectClass, CorbaObjectName :: String.
ClassName, JavaClassName :: String.
generate java code OperationMapping
    using corba :
  CorbaPackageName :=
    OperationMapping get corba package name
  CorbaObjectClass :=
    OperationMapping get corba object type,
  ClassName := OperationMapping get class name,
  JavaClassName := Java || ClassName,
  CorbaObjectName := object || ClassName,
  SetUpCode :=
    ComponentModel generate java code,
  Operations :=
    generate java code for OperationMapping,
return
  import CorbaPackageName . *;
  public class JavaClassName {
    private CorbaObjectClass CorbaObjectName ;
    // initialize CORBA client module
    public void init () {
      SetUpCode
    }
    Operations
}.
```

This rule generates the class structure required by the Java implementation, which consists of a function `init` to set up the CORBA ORB and the operations needed in the server. This includes the code to initialize the CORBA object so that future operations can refer to it. It is necessary to first extract the names of the CORBA package, class of the CORBA object to be referenced within the package, and the name of the class itself. These are all stored in the *OperationMapping*. The name of the Java class generated is simply the string "Java" concatenated [1] with the name of the server class, i.e., `JavaCorbaAccountServer`. The name of the CORBA object is generated in a similar way.

The rule below describes the mechanism for generating individual methods in `JavaCorbaAccountServer`. For simplicity, only the case where the class is to contain a single method is shown. Multiple methods are handled similarly.

```
generate java code for
      OperationName1 ArgumentList1 ReturnType
      maps to
      OperationName2 ArgumentList2 ReturnType :
  JavaReturnType := java type of ReturnType,
  JavaArgumentList :=
      list all Argument from ArgumentList1
          mapped to JavaArgument
              by function java argument of
                  Argument is JavaArgument,
  JavaArgumentListDefinition :=
      separate JavaArgumentList by , ,
  OperationCall := generate java code for
      OperationName2 ArgumentList1 ReturnType,
return
    public JavaReturnType OperationName1
          ( JavaArgumentListDefinition ) {
```

[1] The TLG concatenation operation (||) differs from juxtaposition in that it does not produce a space between the operands.

```
    EventTrace . setBeginTime ();
    OperationCall
    EventTrace . setEndTime ();
    EventTrace .
      calculateResponseTime ();
}.
```

This generation assumes that the methods have the same return type and so the main task is to express the arguments of the first operation in terms of Java syntax, generate the appropriate method call, and instrument the code with the event grammar mechanism to measure the response time. The former is accomplished by using a TLG list comprehension to map the arguments in *ArgumentList1* into corresponding Java arguments represented by *JavaArgumentList*. Each *Argument* from *ArgumentList1* is mapped into a *JavaArgument* using the function java argument of *Argument* is *JavaArgument*. There is a subtlety here in that *JavaArgumentList* is an abstract syntax representation of the desired argument list and so this must be made into concrete syntax using the separate operation which adds the appropriate commas in between the argument declarations. The appropriate method call is handled by the rule below.

generate java code for
  OperationName ArgumentList ReturnType :
IdentifierList :=
  list all *Argument* from *ArgumentList*
    mapped to *Identifier* by
      function argument id of *Argument*
        is *Identifier*,
IdentifierListInCall :=
  separate *IdentifierList* by , ,
return
  CorbaObjectName . *OperationName*
    (*IdentifierListInCall*);.

Again a list comprehension is used to extract the arguments from the argument list, this time only the identifier part (achieved by function argument id of *Argument* is *Identifier*). Likewise, the abstract syntax representation must be made concrete by comma separators.

Finally, the event grammar instrumentation is added to measure the time at the beginning of the server method call and again at the end so that the actual response time can be evaluated against the required QoS ($< 100ms$). The QoS metrics for "response delay" mean execution time for each method call within the server or client, and require the instrumentation of each generated wrapper for the client/server method call with auxiliary functions able to check the clock at the beginning and at the end of method call, calculate the duration, and submit it to the execution monitor (also generated as a part of instrumentation). We assume that these are taken care of by a class called `EventTrace`. Each of the two example systems will be implemented with the code for carrying out event trace computations according to test cases which must be supplied by the user. These test cases will be executed to verify that the bank account management system satisfies the QoS specified in the query. If the system is not verified, it is discarded. This verification process is carried out for each of the generated bank account management system (two in the above example). Then the one with the best QoS is chosen, in the above example the `CorbaAccountServer` and `JavaAccountClient` combination.

For the example UMM specification, the following code for the `depositMoney` function would be produced.

```
public void depositMoney (float ip) {
  EventTrace . setBeginTime ();
  objectCorbaAccountServer . deposit (ip);
  EventTrace . setEndTime ();
  EventTrace . calculateResponseTime ();
}
```

5. Summary and Conclusions

The UMM provides a framework for constructing systems that involve interoperation of heterogeneous and distributed software components. It is based on: a) a meta-component model, b) interpretation by a Two-Level Grammar of queries requesting distributed systems, c) formal specification, based on Two-Level Grammar of components and systems, d) generative rules, along with their formal specifications, for assembling an ensemble of components from available choices, and e) validation and assurances of QoS using event grammars.

In the future, the efficient generation and update of a distributed computing system will require at least a semi-automatic integration of software components, based on their advertised QoS, in such a way that it meets the QoS constraints specified by the user. The result of using UMM, with the tools and techniques it embodies, is semi-automatic construction of such a system. A simple case study is provided in this paper for illustration, but the principles of the proposed approach can be applied to larger applications.

Acknowledgements. The authors would like to thank IFAD for providing an academic license to the IFAD VDM Toolbox in order to conduct this research.

References

[1] Auguston, M. Program Behavior Model Based on Event Grammar and its Application for Debugging Automation. In *Proc. 2nd Int. Workshop on Automated and Algorithmic Debugging*, pages 277–291, 1995.

[2] Auguston, M., Gates, A., Lujan, M. Defining a Program Behavior Model for Dynamic Analyzers. In *Proc. SEKE '97, 9th Int. Conf. Software Eng. and Knowledge Eng.*, pages 257–262, 1997.

[3] BBN Corporation. *Quality Objects (Quo)*, `http://www.dist-systems.bbn.com/tech/QuO`, 2001.

[4] Brahnmath, G. J., Raje, R. R., Olson, A. M., Auguston, M., Bryant, B. R., Burt, C. C. A Quality of Service Catalog for Software Components. In *Proc. (SE)^2 2002, Southeastern Software Engineering Conf. (to appear)*, 2002.

[5] Bray, T., Paoli, J., SperbergMcQueen, C. M., Maler, E. Extensible Markup Language (XML) 1.0. Technical report, W3C, October 2000. `http://www.w3c.org/xml`.

[6] Bryant, B. R. Object-Oriented Natural Language Requirements Specification. In *Proc. ACSC 2000, 23rd Australasian Computer Science Conf.*, pages 24–30, 2000.

[7] Bryant, B. R., Lee, B.-S. Two-Level Grammar as an Object-Oriented Requirements Specification Language. In *Proc. 35th Hawaii Int. Conf. System Sciences*, 2002.

[8] Cleaveland, J. C., Uzgalis, R. C. *Grammars for Programming Languages*. Elsevier North-Holland Inc., 1977.

[9] Czarnecki, K., Eisenecker, U. W. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.

[10] Dürr, E., van Katwijk, J. VDM++, A Formal Specification Language for Object Oriented Designs. In *COMP EURO 92*, pages 214–219, 1992.

[11] IFAD. The VDM++ Toolbox User Manual. Technical report, IFAD, 2000.

[12] INRIA-Rocquencourt. *ASTER: Software Architectures for Distributed Systems*, `http://www-rocq.inria.fr/solidor/work/aster.html`, 2001.

[13] Larsen, P. G., et al. Vienna Development Method - Specification Language - Part I: Base Language. Report, ISO/IEC 13817-1, December 1996.

[14] Lee, B.-S., Bryant, B. R. Automated Conversion from Requirements Documentation to an Object-Oriented Formal Specification Language. In *Proc. ACM Symp. Applied Computing (to appear)*, 2002.

[15] Lee, B.-S., Bryant, B. R. Contextual Knowledge Representation for Requirements Documents in Natural Language. In *Proc. FLAIRS 2002, 15th Int. Florida AI Research Symp. (to appear)*, 2002.

[16] Michigan State University. *RAPIDware: Component-Based Development of Adaptable and Dependable Middleware*, `http://www.cse.msu.edu/rapidware`, 2001.

[17] Object Management Group (OMG). Model Driven Architecture: A Technical Perspective. Technical report, OMG Document No. ab/2001-02-01/04, February 2001.

[18] Raje, R. R. UMM: Unified Meta-object Model for Open Distributed Systems. In *Proc. ICA3PP 2000, 4th IEEE Int. Conf. Algorithms and Architecture for Parallel Processing*, 2000.

[19] Raje, R. R., Auguston, M., Bryant, B. R., Olson, A. M., Burt, C. C. A Unified Approach for the Integration of Distributed Heterogeneous Software Components. In *Proc. Monterey Workshop Engineering Automation for Software Intensive Systems*, pages 109–119, 2001.

[20] Raje, R. R., Auguston, M., Bryant, B. R., Olson, A. M., Burt, C. C. A Quality of Service-based Framework for Creating Distributed Heterogeneous Software Components. *Submitted for publication*, 2002.

[21] Rosa, N. S., Cunha, P. R. F., Justo, G. R. R. Process^{NFL}: A Language for Describing Non-Functional Properties. In *Proc. 35th Hawaii Int. Conf. System Sciences*, 2002.

[22] Siram, N. N., Raje, R. R., Bryant, B. R., Olson, A. M., Auguston, M., Burt, C. C. An Architecture for the UniFrame Resource Discovery Service. *Submitted for publication*, 2002.

Quality of Service Behavioral Model From Event Trace Analysis

John Drummond*
SPAWARSYSCEN
San Diego Ca 92152-5001
(619)553-4131, drummond@spawar.navy.mil

Valdis Berzins, Luqi, William Kemple, Mikhail Auguston, Nabendu Chaki
Naval Postgraduate School
555 Dyer Road, Monterey, CA 93943-5118 USA
(831)656-2610, berzins@cs.nps.navy.mil
(831)656-2735 luqi@cs.nps.navy.mil,
(831)656-2249, kemple@cs.nps.navy.mil
(831)656-2509, auguston@cs.nps.navy.mil
(831)656-2509, nchaki@nps.navy.mil

Abstract

The distributed command & control environment includes limited computer resources and numerous mission critical applications competing for these scarce resources. Additionally the stringent constraints and considerable complexity of distributed command & control systems can create a condition that places extreme demands upon the allocated resources and invites a potential for program errors. Consistent quality of service distribution can be a critical element in ensuring effective overall program completion while avoiding potential errors and process failures. The potential for errors and process failures can be understood and addressed by performing a practical analysis of the resource deployment procedures utilized within this environment. However, analyzing resource-based quality of service within a distributed command & control environment is a demanding endeavor. This difficult task can be simplified

* This work sponsored by the Defense Advanced Research Projects Agency, Information Technology Office (DARPA-ITO)

by directly examining specific quality of service actions that take place during program execution. Therefore, to pragmatically isolate these actions and develop a practical quality of service behavioral model, the research discussed in this paper has implemented an event trace approach to examine the exact quality of service execution path during program operation.

Introduction

The command & control environment is especially complex and may certainly exhibit dynamically changing attributes during its operation. The processing of command & control elements can exhibit perplexing difficulties such as abrupt mission changes, and dynamic tactical surprises[Harn 99]. Many critical applications within this environment could benefit from distributing the processing load. Distribution of the processing load does however also increase the overall complexity of the environment. Despite the expanded complexity, the augmentation to command & control environments of distributed processing is desired. The distributed processing environment can provide added benefits over the non-distributed approach due to the capacity for improvement in program accessibility, overall performance, additional sharing of limited resources, and the increased fault tolerance capabilities. However, this distributed command & control environment does present an expanded assemblage of requirements and constraints for effective computer resource control. The efficient allocation of computer resources can be considered a major element of these requisites. The direct implementation of resource control features into the distributed command & control infrastructure can be extremely advantageous for software programs that contain mission critical requirements. Implementing quality of service features into the distributed command & control infrastructure is not a trivial task. Additionally, subsequent to implementing the quality of service features, an examination must be performed upon the effectiveness of the implementation.

To properly ascertain that the essential quality of service based system resources are being reasonably utilized and efficiently shared among these programs some evaluation of the resource deployment method should be conducted. However, current analysis techniques for evaluation of resource deployment and control are somewhat lacking in that there is no exacting method to focus exclusively upon specific quality of service actions that take place during actual program execution. Therefore, the analysis of proper employment and dispersion of available resources is the focus of this research in the area of distributed command & control processing. This direct analysis can be carried out though the use of quality of service based behavioral models. The development characteristics of the behavior model are described in the next section.

Approach

The foundation for the approach to developing the quality of service behavioral model considers a centralized resource provisioning mechanism for control of all computer resources that can be found within the end-to-end pathway. Typically communication based quality of service analysis approaches have focused upon the network resource provisioning implementations that utilize a

decentralized resource management technique to disperse the required communication bandwidth resources at numerous locations (e.g. ATM switches, etc). The scope of the research being pursued in this paper expands this quality of service analysis of resource deployment to include computer resources that can consist of CPU, Network, Disk, I/O, and Memory. These resources can be considered critical elements within a true end-to-end distributed environment and have a direct bearing upon any quality of service capabilities.

This focus of efficient quality of service within the total end-to-end pathway is a much needed element for current DoD systems as stated by the Defense Advanced Research Projects Agency Quorum program manager [Koob 99] "While emerging network-level QoS mechanisms (such as RSVP) are an essential enabling technology for Quorum, they are insufficient in that they are limited to communications QoS. Quorum defines "end-to-end" as being the quality-of-service seen by the application, which calls for coordinated QoS management across middleware, operating systems, and networks."

For the purpose of this research investigation the mechanism for controlling and coordinating these critical quality of service resources will be centralized within a singular system as can be found within the Linux/RK resource kernel[Rajkumar 98]. This research does not examine the multiple controller communication/network mechanism mentioned earlier. This investigation is accomplished by an in-depth look at various quality of service and resource deployment characterizations as well as the application of high level modeling and quality of service analysis. The detailed analysis is attained through the utilization of the SPAWAR System Center DARPA Quorum Integration Test & Exploitation project (Quite) testbed environment located at the SPAWAR System Center. Other specific logical conditions and constraints for this work include distributed systems, heterogeneous environment, multiple diverse quality of service levels essential for program execution (i.e. application requirements vary high/low needs), and available resources can include network bandwidth, CPU, memory, etc.

To achieve a precise analysis of quality of service procedures an approach has been implemented to examine the exact quality of service execution path during program operation. The evaluation approach utilized in this research is based upon an event trace concept employed by [Auguston 00] originally as an analysis tool for focusing upon correctness in C language programs. This event trace concept discusses the idea that testing and debugging are mostly concerned with the program run-time behavior, and states that developing a precise model of program behavior becomes the first step towards any dynamic analysis.

The *method* of performing the quality of service event trace analysis begins with an event trace of a targeted application. This event trace is utilized as the basis for developing the quality of service behavior that characterizes the targeted program. The quality of service based event trace approach allows for a detailed quality of service parameter examination. This event trace is utilized as the tool for collecting the predefined quality of service metrics and allows for in-depth analysis based upon these previously developed metrics. The specific events to be isolated within

these parameters for this research are based upon actions that may have temporal properties (e.g. Event Start, Event Stop) or simply be atomic in nature (e.g. Initialization). This follows the event trace work of [Auguston 98]: "Every event defines a time interval which has a beginning and end. For atomic events, the beginning and end points of the time interval will be the same."

The procedures for performing the quality of service event trace include:

- Develop operative models of execution pathways (quality of service & resource control specific statement execution) within the target application based upon identifiable details such as resources requested, resources utilized, resources available, etc.

- Initiate the development of a working model of program behavior based upon quality of service factors. This is accomplished by producing abstractions of events that are fundamental to specific quality of service actions performed during program execution, which include:

    - Quality of service request statement execution that requests resource reservation.
    - Procedure execution that focuses upon the evaluation and negotiation of available resources to be applied to the originating resource request.
    - Software statement execution of procedures for proper utilization of the assigned resources.
    - Execution of statements responsible for the detection of any resource needs change within the application software.
    - Execution of procedures focusing upon the re-negotiation based on increase or decrease of available and previously assigned resources.
    - Execution of reallocation statements for specific resources by the resource controller.
    - Sending and receiving of quality of service related messages by both the application and resource controller software.

- Identify quality of service specific application program points that directly relate to appropriate resource deployment as illustrated in Figure 1. Such elements have direct consequences upon quality of service behavior and include:

    - QoS specific message passing
    - Application QoS violations
    - QoS negotiations
    - QoS resources and control of resources
    - QoS re-negotiations
    - QoS level

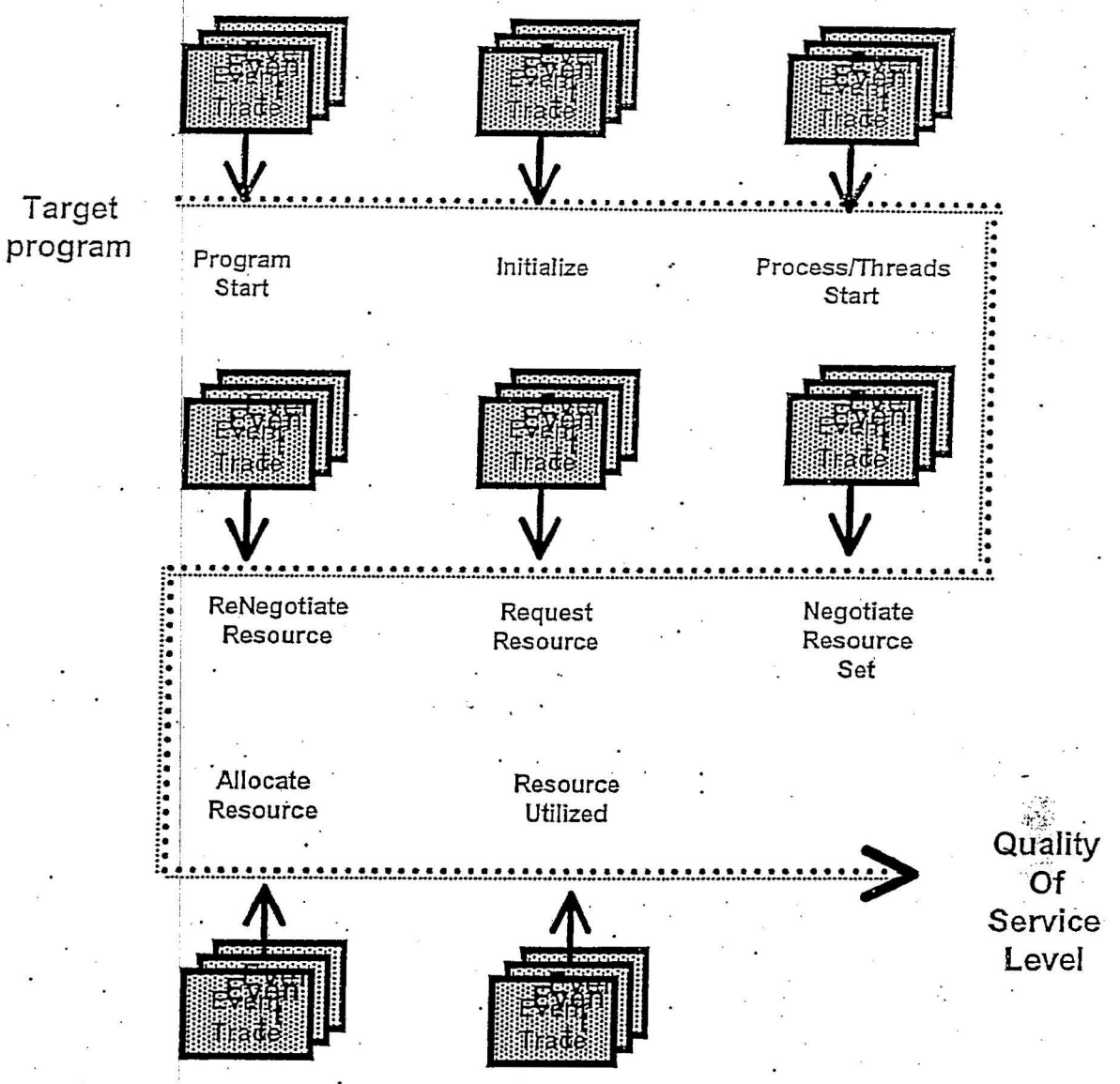

Figure 1. Quality of Service Program Points.

- Instrument the targeted program based upon these previously identified specific quality of service program points. This direct code instrumentation will allow for effective event trace recording at the precise location of the quality of service actions of interest.

At this point it is necessary to further expand upon the events of interest for quality of service analysis and the development of the behavior model. An event is a detectable action that influences the overall achievement of the desired quality of service level. The event is the smallest element of the quality of service behavior model. The discovery of this action is noted by the embedded instrumentation within the targeted program sources at the pre-defined program

points as previously shown in Figure 1. The event attributes describe the event and include the process or thread within which this event has occurred, and a boolean attribute denoting the associated quality of service action of success/failure.

The event model is constructed from a specific quality of service based action and all the attributes relevant to this action. The event model is applied to the event trace for the purpose of constructing the quality of service behavior. There are eight events of interest and their respective attributes that form the composition of the behavioral model.

The resource request event is critical to the development of the behavioral model because it represents the applications mechanism for acquiring the proper resources for successful program execution. Within the quality of service behavioral model every resource request event and subsequent failure/success attribute is indicative of the applications behavior. The resource request event model is composed of the action of requesting resources and the set of event attributes that include: depth level 'DP, process type 'TP, location 'LC, path 'PA, resource type 'RT. The request resource event RQ = {DP, TP, LC, PA, RT}, this event model is illustrated in the next figure.

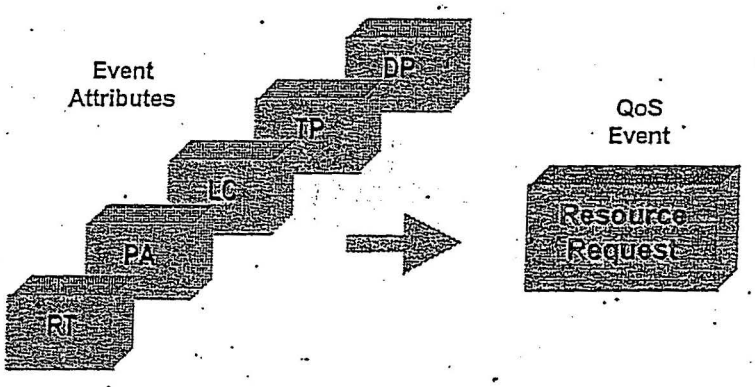

Figure 2. Resource Request Event Model.

The quality of service violation event is an important element of the behavioral model as it is representative of a quality of service fault. This failure event has a causal relation to the preceding quality of service associated attempt actions that include resource negotiation, resource request, resource re-negotiation, and resource assign. Within the composition of the quality of service behavioral model failure is indicative of the applications behavior. The quality of service violation event model consists of the resource request failure, or resource negotiation failure, or resource re-negotiation failure, or resource assigned failure actions and the set of event attributes: depth level 'DP, process type 'TP, location 'LC, path 'PA, resource type 'RT. The quality of service violation event QV = {RT, DP, TP, LC, PA} and this event model is illustrated in the next figure.

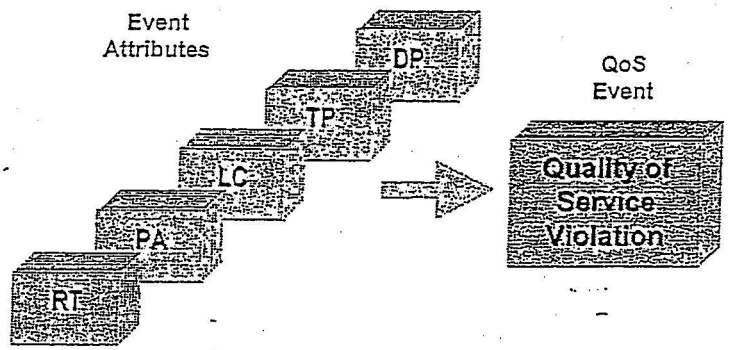

Figure 3. Quality Of Service Violation Event Model.

The quality of service level event supports the behavior model as it represents the action of the resource controller appropriating the requested resource. Within the quality of service behavioral model the appropriation of resources is a significant action in the attainment of proper quality of service and consequently characterizing the applications behavior. The quality of service level event model is composed of the action of resource reserve creation success action through the resource controller and the set of event attributes that include: depth level 'DP, process type 'TP, location 'LC, path 'PA, resource type 'RT, resource size 'SZ, resource period 'RP, resource deadline 'RD, and resource used 'RU. The quality of service event QL= {DP, TP, LC, PA, RT, SZ, RP, RD, RU}, this event model is illustrated in the next figure.

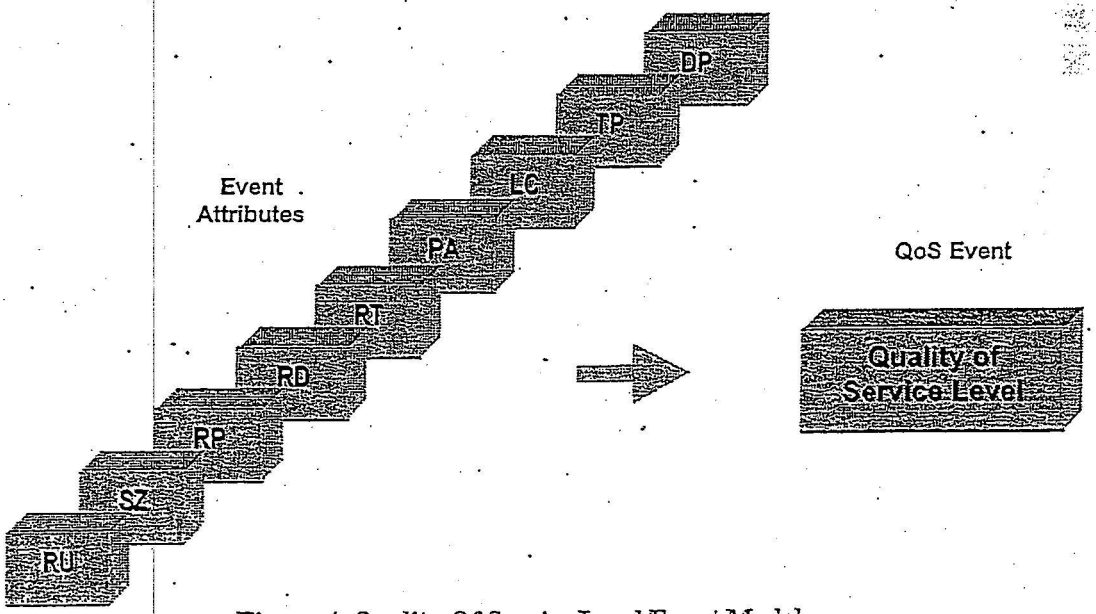

Figure 4. Quality Of Service Level Event Model.

The resource negotiation event is an important part of the behavior model because it represents the transaction of establishing a resource set with the resource controller. This event is significant in the acquisition of resources and therefore an important in the development of the applications quality of service behavior. The resource negotiation event model is comprised of the action of setting up this resource set and the event attributes that include: depth level 'DP, process type

'TP, location 'LC, path 'PA, resource type 'RT. The resource negotiation event RN= {DP, TP, LC, PA, RT}, this event model is illustrated in the next figure.

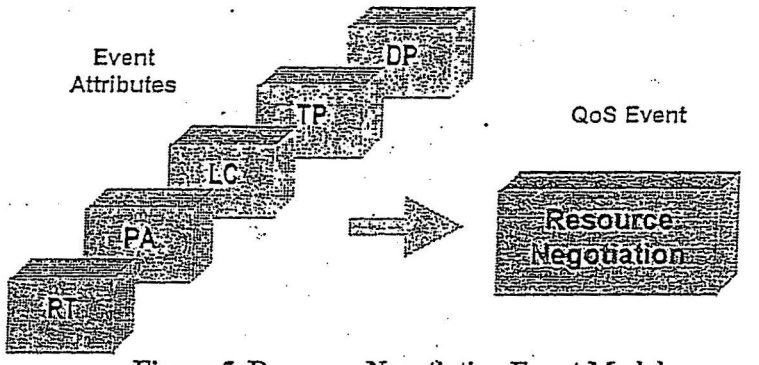

Figure 5. Resource Negotiation Event Model.

The system reservation event is a component in the behavior model because it represents the action by the system, and other applications not currently being targeted by the event trace, of requesting resources from the resource controller. When the focus is directed only at the target program for evaluation the system resource event simply represents a competing load application. This event is critical to the quality of service behavior model because it enables an evaluation of the target program under resource competition load. The system reservation event model consist of this resource reservation action by these competing users through the resource controller and the set of event attributes that include: process type 'TP, location 'LC, path 'PA, resource type 'RT, resource size 'SZ, resource period 'RP, resource deadline 'RD, and resource used 'RU. The system reservation event SR = {TP, LC, PA, RT, SZ, RP, RD, RU}, this event model is illustrated below.

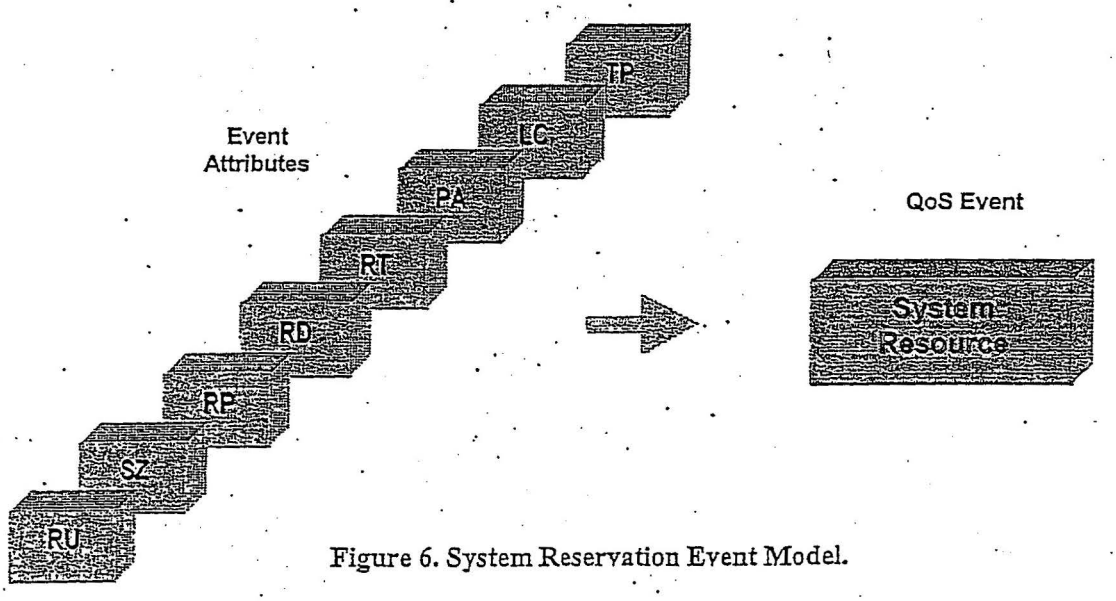

Figure 6. System Reservation Event Model.

The resource assignment event is a critical element in the quality of service behavior model because it describes the action of assigning resources by the resource controller to the requesting

thread/process. The resource assignment event model is composed of the action of attaching the resource set to the specific process/thread through the resource controller and the set of event attributes that include: depth level 'DP, process type 'TP, location 'LC, path 'PA, resource type 'RT. The resource assignment event SR= {DP, TP, LC, PA, RT}, this event model is illustrated in the figure below.

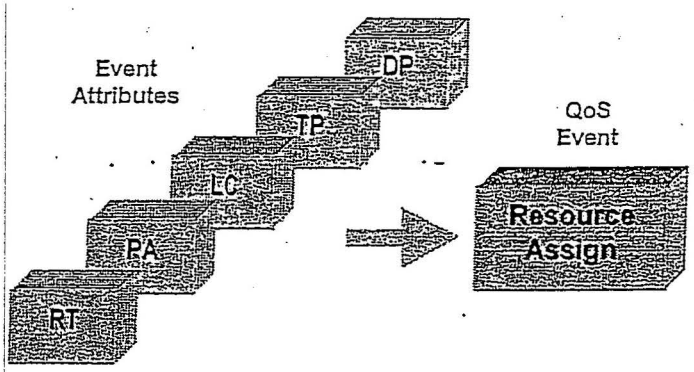

Figure 7. Resource Assign Event Model.

The path length event is part of the quality of service behavior model because it represents the action of traversing the quality of service path to a succeeding program point level within the event trace. The path length event model consists of the action of proceeding through program points and the set of event attributes that include: depth level 'DP, process type 'TP, location 'LC, path 'PA. The path length event PL = {DP, TP, LC, PA} and this event model is illustrated below.

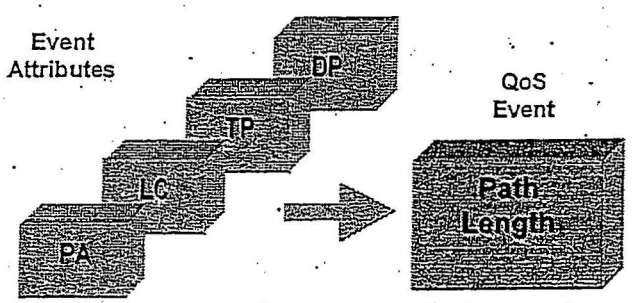

Figure 8. Path Length Event Model.

The resource re-negotiation event is critical to the quality of service behavior model because it is representative of the process/thread actions to correct a preceding quality of service violation. The resource re-negotiation event model is comprised of the action of re-negotiating the amount of resource requested through the resource controller and the set of event attributes that include: depth level 'DP, process type 'TP, location 'LC, path 'PA, resource type 'RT, resource size 'SZ, resource period 'RP, and resource deadline 'RD. The resource re-negotiation event RR = {DP, TP, LC, PA, RT, SZ, RP, RD}, this event model is illustrated in the next figure.

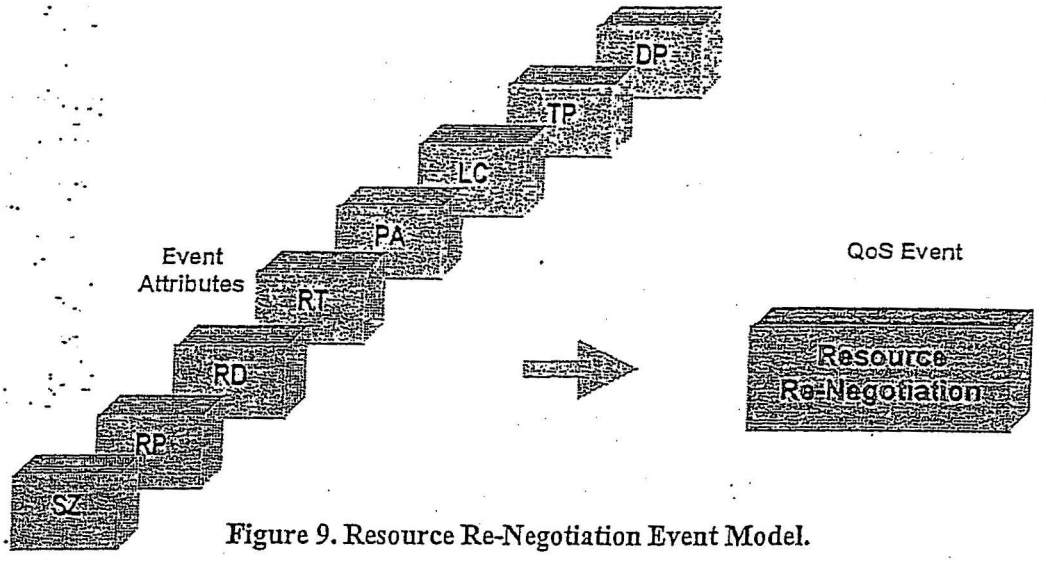

Figure 9. Resource Re-Negotiation Event Model.

These specific event trace quality of service actions which comprise the event models are informally structured with no imposed overall ordering structure as they occur asynchronously. However, there is a partial ordering within a specific thread/process execution as denoted by the thread/process depth level attribute, and a causal ordering between request events(resource negotiation, request resource, resource re-negotiation, resource assign) and fault event(quality of service violation). After the event occurs the event trace notes the specific attributes of the quality of service action such as type(quality of service resource), level(path depth), path(aggregate progression of the event trace), ptype(process or thread), and loc(within the process/thread). This data is noted and a boolean evaluation process determines its success/failure attribute.

The behavioral model is composed of resource request events 'RQ, resource negotiation events 'RN, resource assign events 'RA, quality of service events 'QV, resource re-negotiation events 'RR, quality of service level events 'QL, system reservation events 'SR, and path length events 'PL. Potential failure behaviors are comprised of the following sets of events: {RN, QV}, {RQ, QV}, {RQ, QV, RR, RR, RR, QV}, {RR, QV}, and {RA, QV}. Typical success behaviors are composed of sets of events that include: {RN}, {RQ, QL, RA}, {RR, QL, RA}, and {RA}. An example of a quality of service behavior model that characterizes quality of service failure is illustrated in the next figure.

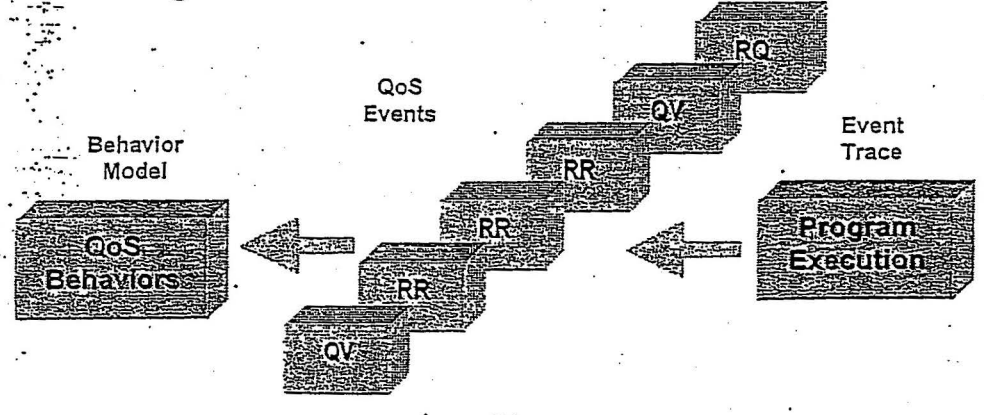

Figure 10. Quality Of Service Behavior Model.

The behavioral model can be utilized to isolate specific quality of service behaviors. For example in the failure occurrence illustrated in above with the events {RQ, QV, RR, RR, RR, QV} the program point of failure can be isolated through examination of the event trace results. This search can be achieved through a retrace of the execution path to the specified depth level of the distinct(named) thread/process, and by examination of the event attributes associated with the failure event quality of service violation QV = {DP, TP, LC, PA, RT}.

The quality of service event trace on the targeted application collects these events and based upon this information the quality of service behavior can be constructed. The behavior model is partially ordered set of event types (e.g. resource request) and event attributes (success/fail boolean). The quality of service metrics of the event trace are calculated based on quality of service actions. These calculations include elements such as the number of resource re-negotiation events that have occurred, and the number of application quality of service violations. The results of these event trace metrics include the lists of specific processes/threads identities and their resource specific events. This information provides the necessary data to construct the quality of service behavior.

Applying The Event Trace

This discussion has thusfar focused upon the composition of the quality of service event trace. The model of this generalized (general case) approach can be applied to the specific case implementation of a pre-selected application. The employment of the quality of service event trace analysis is based upon the event models applied to a target application program that has been instrumented for accurate feedback. This analysis is then utilized to develop an overall quality of service behavior for characterization of the command & control systems application that can be applied to mitigation of discovered quality of service related efficiency problems. Through direct examination of the event trace results, specific potential failure regions (QV events) can be isolated to specific path locations and thread/process depth levels that denote the distinct program point within the targeted application. The potential error region can then be adjusted to improve the quality of service level achievement probability.

The selection of the target program/environment for this work has been subjectively inspired through the research of the DARPA Quorum program. The specific emphasis of the DARPA-ITO sponsored Quorum program is the development of computing environments with quality of service attributes, controls, and guarantees on local to global scales[Koob 99]. This Quorum program is a multi-million dollar collaboration of fifty top research groups representing universities, industries, and SPAWAR System Center. The quality of service event trace research

reported in this paper leverages from this DARPA project focus and this association has provided a high degree of valuable input.

A failure detection program was isolated as a candidate target application based upon the logical conditions and constraints discussed earlier. This fast failure detection program has as its primary objective the efficient and prompt detection of node failures within group communication software as utilized within distributed mission critical systems of the AEGIS environment. The fast failure detection program was designed and developed under the DARPA-ITO Quorum Integration, Testbed and Exploitation (Quite) project efforts.

As noted in [Drummond 02] this program can be set up to take advantage of a resource management system based upon quality of service procedures or operate as a simple non-quality of service application "The Fast Failure Detector can be built and executed on its own or it can be executed while taking advantage of facilities like Linux/RK [Oikawa 98] and Ensemble[Birman 00] group communication." For the purpose of this event trace analysis research the Fast Failure Dector program has been implemented using both the Linux/RK kernel and the Ensemble group communication software.

The specific area of concentration within this targeted environment is based upon the following problem domain characteristics: distributed computing environment, multiple heterogeneous systems, network medium connections, software applications with specific requirements (quality of service resource needs), centralized resource control software (with quality of service awareness), metrics data gathering instrumentation software. This specifically isolated computing environment can be readily encountered within the AEGIS system, as well as in many other DOD and commercial systems. AEGIS is a combat system architecture that contains a computer-based command & decision component. The core element of the AEGIS architecture provides simultaneous operations capability. These operations include measures against multi-mission threats including anti-air, anti-surface, and anti-submarine. The problem space of this domain requires specific levels of performance to operate correctly.

Explicit quality of service program points that directly relate to resource utilization have been isolated within the target program. Based upon these program points the target application has been instrumented. This has been accomplished by direct source code instrumentation that allows for effective event trace recording at the actual location where the quality of service specific actions take place. The analysis results can be utilized to develop an overall characterization of the fast failure detection program for any mitigation of discovered quality of service related efficiency problems. The specific event trace analysis has examined the quality of service events and related quality of service characteristics of this fast failure detection program within a distributed environment.

For this application of the quality of service event trace analysis approach to the specific case of the selected targeted application the following distinct events and attributes have been recorded

within each quality of service event trace execution. These events, their actions and attributes follow the event models described earlier.

EVENT	ACTION	ATTRIBUTE
RES_NEG	Resource Set Negotiation	RES_TYP, PATH, LOC, LEVEL, PTYPE
REQ_RES	Resource Request	RES_TYP, PATH, LOC, LEVEL, PTYPE
RES_ASG	Resource Assignment	RES_TYP, PATH, LOC, LEVEL, PTYPE
PATH_LN	Quality of Service Program Point Traversal	PATH, LOC, LEVEL, PTYPE
QOS_VIO	Quality of Service Violation	RES_TYP, PATH, LOC, LEVEL, PTYPE
RES_RNG	Resource Re-Negotiation	RES_TYP, PATH, LOC, LEVEL, PTYPE, SIZE, PERIOD, DEADLINE.
QOS_LEV	Resource Appropriation	RES_TYP, PATH, LOC, LEVEL, PTYPE, SIZE, PERIOD, DEADLINE, USED
SYS_RES	System/Application Resource Reservation	RES_TYP, PATH, LOC, PTYPE, SIZE, PERIOD, DEADLINE, USED

Table 1. Quality of Service Events.

These specific event types (shown in table 1) that have been included within the quality of service event trace of the target failure detection program were chosen because they represent distinct actions during program execution that have a direct influence resource utilization. These event types include attributes that are closely associated with and help describe these actions. The quality of service events occur asynchronously within the quality of service event trace and are informally structured with no overall strictly imposed ordering, however there is a partial ordering within each executing thread.

The RES_TYP notation represents the event trace attribute that denotes the type of resource(RT) reservation that is requested. This event attribute is not utilized during the analysis of this target failure detection program, as the sole resource that has been reserved by this program is the CPU resource. When it is used, the other possible resources that this attribute can represent include Disk, Network, and Memory. The PATH attribute references the total event trace quality of service path(PA) length that has been recorded during the application execution. This path element represents a simple integer value that is dynamically updated and recorded as the event trace proceeds. This integer value indicates the aggregate progression of the event trace. The quality of service event trace attribute labeled LEVEL is similar to the Path element. However, this element reflects the specific process or thread execution path depth(DP) as it proceeds though the operations necessary to attain a specific level of quality of service. This element is also a simple integer that is dynamically updated and recorded as the process or thread executes. The LOC attribute references the specific processing location(LC) that the quality of service

event trace is recording from within the specific process or thread. This attribute is a simple char type and includes FFDMAIN, FFDINIT, KSYSTEM, THREAD1, THREAD2, and THREAD3. The next attribute in the quality of service event trace output data is titled PTYPE. This attribute indicates the specific task type(TP) that has been recorded. Two of the possible task types include process, and thread which are directly related to the failure detection application. The SIZE attribute reflects the resource size(SZ) being requested. The SIZE attribute is measured in resource units. The PERIOD attribute indicates the period(RP) that the resource is utilized within. The DEADLINE attribute relates to specific information(RD) utilized by Deadline Monotonic and Earliest deadline First scheduling policies. The USED attribute reflects the event of resources being allocated(RU). The TOTAL element indicates the additive figure of all resources that have been allocated. The AVAL element is representative of the total resource available as indicated by the resource kernel. This element is also measured in resource units.

For this case study additional fabricated competing application tasks RS1, RS2, and system processes such as DISK were executed during the event trace. The competing application is a simple CPU resource load program that requests large amounts(400.0 & 200.0 units) of this resource. Its sole purpose is to present the target program with a resource competitor for evaluation under load. The system process is produced by the resource kernel for continuos disk access and requests a nominal amount of CPU resource(0.299 units). The resource kernel also indicates a setback of minimum 90 resource units that cannot be allocated.

Conclusion

The concluding results of this examination have produced high-level quality of service behavior representations of the fast failure detection program. This quality of service event trace analysis has shown the capacity to specifically reveal various failure points, potential resource re-negotiation inefficiencies, and lengthily quality of service path calls. All of these elements have a direct bearing upon the quality of service based resource deployment efficiency for the distributed command & control fast failure detection application program and environment.

The events of interest from the resulting case study examination demonstrate a typical success behavior composed of {RN}. Also discovered were potential failure behaviors illustrated as a progressive pattern of potential for failure that concludes with a quality of service violation and final failure. This progression can be see in patterns composed of {RQ, QV, RR, QL, RA}, {RQ, QV, RR, RR, QL, RA}, {RQ, QV, RR, RR, RR, QV}. For this final failure the program point of failure can be isolated through examination of the event trace results and the event attributes as shown in Table 2 below. This examination includes a retrace of the execution path to the specified depth level of the distinct(named) thread/process, and by isolation of the event attributes associated with the failure event quality of service violation = {DP, TP, LC, PA, RT}. Where DP= 7-11, TP=THREAD, LC=THREAD2, PA=34-38, RT=CPU.

ETTYPE	PATH	LEVEL	LOC	PTYPE	SIZE	PERIOD	DEADLINE	TOTAL	AVAIL
QOS_VIO	34	7	THREAD2	THREAD				610.299	90

RES_RNG	35	8	THREAD2	THREAD	6.000	50.000	50.000	610.299	90
RES_RNG	36	9	THREAD2	THREAD	4.000	25.000	25.000	610.299	90
RES_RNG	37	10	THREAD2	THREAD	2.000	12.000	12.000	610.299	90
QOS_VIO	38	11	THREAD2	THREAD				610.299	90

Table 2. Event Trace Results.

It is interesting to note in performance analysis that this complete denial of resources could have a catastrophic consequence for the requesting thread or process. This result could also translate into an uncertain outcome for the total program execution resulting in total program failure. This instance of a potential for total program failure was evident in the case study of the fast failure detection program. During the quality of service event trace analysis execution that included competition for resources that produced the data found in Table 2, the fast failure detection program exhibited a total collapse of the THREAD2 task. This failure of the specific thread to reserve necessary resources from the resource kernel in turn resulted in an abort of the program.

Thus far this work has specifically been directed towards the area of developing quality of service behavior models for targeted distributed command & control programs. This utilization of the quality of service event trace approach to behavioral modeling can be further expanded for inclusion into a development/analysis framework.

References

[Auguston 00] Auguston, M., *Assertion Checker For The C Programming Language Based On Computations Over Event Traces*, Fourth International Workshop on Automated Debugging, AADEBUG2000, Munich, Germany, August 2000.

[Auguston 98] Auguston, M., *Building Program Behavior Models*, Proceedings of the European Conference on Artificial Intelligence ECAI-98, Workshop on Spatial and Temporal Reasoning, Brighton, England, August 23-28, 1998.

[Birman 00] Birman, K., et al., "*The Horus and Ensemble Projects: Accomplishments and Limitations*", Proceedings of the DARPA Information Survivability Conference & Exposition (DISCEX '00), Hilton Head, South Carolina, January 2000.

[Drummond 02] Drummond, J., Wells, D., Rahman, M., *Detecting Failure Within Distributed Environments*, SPAWAR Technical Paper TR1884, Space and Naval Warfare Systems Center, San Diego, Ca. 2002.

[Harn 99] Harn, M. Berzins, V. Luqi, Kemple, W., *Evolution of C4I Systems*, Command & Control Research and Technology Symposium, 1999.

[Koob, 1999], Koob, G., *Background for DARPA-ITO Quorum Mission Statement*, Defense Advanced Research Projects Agency Information Technology Office, 1999.

A Better XML Parser through Functional Programming

Oleg Kiselyov

Software Engineering, Naval Postgraduate School
Monterey, CA 93943
oleg@pobox.com
oleg@acm.org

Abstract. This paper demonstrates how a higher-level, declarative view of XML parsing as folding over XML documents has helped to design and implement a better XML parser. By better we mean a full-featured, algorithmically optimal, pure-functional parser, which can act as a stream processor. By better we mean an efficient SAX parser that is *easy to use*, a parser that does not burden an application with the maintenance of a global state across several callbacks, a parser that eliminates classes of possible application errors.
This paper describes such better XML parser, SSAX. We demonstrate that SSAX is a better parser by comparing it with several XML parsers written in various (functional) languages, as well as with the reference XML parser Expat. In the experience of the author the declarative approach has greatly helped in the development of SSAX. We argue that the more expressive, reliable and easier to use application interface is the outcome of implementing the parsing engine as an enhanced tree fold combinator, which fully captures the control pattern of the depth-first tree traversal.

Keywords: XML parsing, traversal, tree fold, Scheme, Haskell

1 Introduction

On the surface of it, parsing of XML presents no problems. We merely need to apply yacc/lex or a similar tool to the Extended BNF grammar in the XML Recommendation. XML parsing ought to be even easier in functional languages, thanks to the development of intuitive parsing combinator libraries.

It comes as a surprise then that all but two functional-style XML parsers barely comply even with a half of the XML Recommendation [13]. None of the pure or mostly functional-style XML parsers support XML Namespaces. With the exception of FXP [10], the existing functional-style parsers cannot process XML documents in a stream-wise fashion. These parsers thus exhibit significant processing latency and are limited to documents that can fit within the available memory. The application interface of the only one functional, full-featured,

S. Krishnamurthi, C. R. Ramakrishnan (Eds.): PADL 2002, LNCS 2257, pp. 209–224, 2002.

stream-oriented parser FXP mirrors the API of the reference XML parser Expat [4]. The latter is notorious for its difficult and error-prone application interface.

XML is markedly more difficult to parse than it is commonly thought. It is by no means sufficient for a parser merely to follow the Extended BNF grammar of XML. Besides the grammar, the XML Recommendation [13] specifies a great number of rules (e.g., whitespace handling, attribute value normalization, entity references expansion) as well as well-formedness and validity constraint checks, which a parser must implement. Whitespace handling rules in particular require an unusually tight coupling between tokenizing and parsing.

The second peculiar aspect of XML parsing is its strong emphasis on efficiency and the convenience of the application interface. The traditional view of parsing as a transformation of a source document into an abstract syntax tree is deficient for several classes of XML applications. We should note first that the traditional approach does apply to XML, where it is called a Document Object Model (DOM) parsing. The DOM approach is a necessity for applications that repeatedly traverse and search the abstract syntax tree of a document. Other applications however scan through the document tree entirely, and only once. Such applications can potentially process an XML document as it is being read. Loading the whole document into memory as an abstract syntax tree is then inefficient both in terms of time and memory. Such applications can benefit from a lower-level, event-based model of XML parsing called a Simple Application Programming Interface for XML (SAX). A SAX parser applies user-defined actions to elements, attributes and other XML entities as they are identified. The actions can transform received elements and character data on the fly, or can incorporate them into custom data structures, including the DOM tree. Therefore, a SAX parser can always act as a DOM parser. The converse is not true.

Although the SAX XML parsing model is more general, more memory efficient and faster, SAX parsers are regarded as difficult to use: "It feels like you are trapped inside an eternal loop when writing code. You find yourself using many global variables and conditional statements" [3].

Is it possible to implement an efficient, compliant, stream-oriented XML parser with a convenient user interface that minimizes the amount of user-application state? Furthermore, can functional programming help to design and to implement such a parser?

This paper proves by construction that the answer to both questions is yes. The contribution of this paper is a SSAX parser [7], a compliant SAX XML parser that is being used in several industrial applications. SSAX is not a toy parser: it fully supports XML Namespaces, character, internal and external parsed entities, `xml:space`, attribute value normalization, processing instructions and CDATA sections. At the same time, SSAX minimizes the amount of application-specific state that has to be shared among user-supplied event handlers. SSAX makes the maintenance of an application-specific element stack unnecessary, which eliminates several classes of common bugs. SSAX is written in a pure-functional subset of Scheme. Therefore, the event handlers are refer-

entially transparent, which makes them easier for a programmer to write and to reason about. The superior user application interface for the event-driven XML parsing is in itself a contribution of the paper. The paper demonstrates that this interface is not an accident but the outcome of a correctly chosen control abstraction, which captures the pattern on depth-first traversal of trees.

The key design principle of SSAX was a view of an XML document as an n-ary tree laid out in a depth-first order. XML parsing is then a tree traversal. We review the topic of functional-style tree traversals in Section 2. We will concentrate on efficiency and on capturing the pattern of such traversals in a higher-order combinator, `foldts`. In Section 3 we describe the SSAX parser, which is an implementation of `foldts` with the tree in question being an XML document. Section 4 demonstrates on several concrete examples that the SSAX parser is indeed efficient, easier to use and less error-prone, compared to other SAX parsers, in particular the reference XML parser Expat and its pure functional analogue FXP. We conclude in Section 5 that functional programming is intuitive and helpful not only for processing XML but for parsing it as well.

2 Depth-First Traversals of Trees

We start with a very simple example of a functional-style depth-first tree traversal and gradually extend it to improve efficiency and to abstract the pattern of the traversal. Although the SSAX parser has been implemented in Scheme, this section will use Haskell notation. The latter is more succinct; furthermore, it is more convenient for direct comparison with important papers on tree folding [5][6], which use Haskell notation.

Our trees are represented by the datatype

```
data Tree = Leaf String | Nd [Tree]
```

Given such a tree, we turn to our first problem of concatenating strings attached to all leaves, in their depth-first traversal order. If we view our trees as realizations of an XML information set [14], our first problem becomes that of computing a *string-value* for the root node of the information set.

The obvious solution to the problem

```
str_value1:: Tree -> String
str_value1 (Leaf str) = str
str_value1 (Nd kids) = foldr (++) "" (map str_value1 kids)
```

where

```
foldr:: (a->b->b) -> b -> [a] -> b
foldr f z [] = z
foldr f z (x:xs) = f x (foldr f z xs)
```

although elegant, is deficient. Indeed, let us apply str_value1 to a full binary tree of depth k whose leaves are one-character strings (2^k leaves total). Executing str_value1 then requires $k2^k$ character-moving operations and produces $(k-1)2^k$ garbage characters. The algorithm can be improved by noting that we do not have to concatenate the strings eagerly. Instead, we can accumulate strings in a list and join them after the traversal.

```
str_value2:: Tree -> String
str_value2 = concat . str_value2'

str_value2' (Leaf str) = [str]
str_value2' (Nd kids) = concat (map str_value2' kids)
```

This halves the amount of garbage and the number of character movements. However, appending two lists of size 2^i takes 2^i operations. The algorithm still has the time complexity of $O(k2^k)$; it still produces $k2^{k-1} + 2^{k+1}$ list cells of garbage. The best solution is to build a list of strings in the reverse order – with the reversal and concatenation at the very end:

```
str_value3:: Tree -> String
str_value3 = concat . reverse . (str_value3' [])

str_value3' seed (Leaf str) = str : seed
str_value3' seed (Nd kids) = foldl str_value3' seed kids
```

where

```
foldl:: (a->b->a) -> a -> [b] -> a
foldl f z [] = z
foldl f z (x:xs) = foldl f (f z x) xs
```

Some language systems offer a string-concatenate-reverse function, which halves the amount of the produced garbage. The running time of str_value3 is linear in the size of the tree. The amount of garbage – while unavoidable – grows only linearly with the size of the tree. The function str_value3 differs from str_value1 and str_value2 in another aspect. The actions at children nodes of the same node are no longer independent. The actions are threaded through the seed argument and must be performed in order. The independence of actions in str_value1 and str_value2 manifested itself in the presence of map, which is absent in str_value3.

We now turn to the next example – computing a digest of a tree. We want to traverse a tree depth-first and to compute an MD5 hash of all encountered nodes and leaf values. A hash function is generally non-associative. Therefore, we have no choice but to use a stateful traversal similar to that of str_value3.

```
md5Init:: MD5Context
md5Update:: String -> MD5Context -> MD5Context
md5Final:: MD5Context -> String
```

```
tree_digest:: Tree -> String

tree_digest = md5Final . (tree_digest' md5Init)
tree_digest' ctx (Leaf str) =
  md5Update "/leaf" $ md5Update str $ md5Update "leaf" ctx
tree_digest' ctx (Nd kids) =
  md5Update "/node" $ foldl tree_digest' (md5Update "node" ctx)
                                                          kids
```

Can we separate the task of tree traversal and recursion from the task of transformation of a node and a state? The benefits of encapsulating common patterns of computation as higher-order operators instead of using recursion directly are well-known [11][5]. For lists, the common pattern of traversal is captured by the familiar `foldl` and `foldr` operators, which can be generalized to trees [11][6]:

```
foldt:: (String -> a) -> ([a] -> a) -> Tree -> a
foldt f g (Leaf str) = f str
foldt f g (Nd kids) = g (map (foldt f g) kids)
```

Unlike the functions `str_value1` and `str_value2`, the efficient `str_value3'` cannot be expressed via `foldt` in a simple way because the actions at branches are dependent on the history of the traversal and cannot be simply mapped to children nodes. Such functions are often distinguished [11] by an extra parameter, which acts as a an accumulator or a continuation: (cf. `str_value2'` with `str_value3'` above). Such functions can be written as second-order folds [11], which return procedures as results. In our example:

```
str_value31 tree = concat $ reverse $ (str_value31' tree [])
   where
      str_value31' = foldt (\str seed -> str : seed)
         (\new_kids seed -> foldl (flip ($)) seed new_kids)
```

This representation requires higher-order features of the language and often not as efficient because (`str_value31' tree`) creates as many closures as there are nodes in the `tree`. The closures are then applied to `[]`, which generates the final result. In strict languages such as ML or Scheme (used in the following sections), closure creation is relatively expensive.

To make "mapping" of an accumulating function to a tree efficient, we introduce a more general control operator:

```
foldts :: (a->a) -> (a->a->a) -> (a->[Char]->a) -> a-> Tree-> a
foldts fdown fup fhere seed (Leaf str) = fhere seed str
foldts fdown fup fhere seed (Nd kids) =
   fup seed $ foldl (foldts fdown fup fhere) (fdown seed) kids
```

A user instantiates `foldts` with *three* actions; for comparison, `foldr` requires only one action and `foldt` needs two. The three `foldts` actions are threaded via

a seed parameter, which maintains the local state. An action accepts a seed as one of its arguments and returns a new seed as the result. The action fhere is applied to a leaf of the tree. The action fdown is invoked when a non-leaf node is entered and before any of the node's children are visited. The fdown action has to generate a seed to be passed to the first visited child of the node. The action fup is invoked after all children of a node have been seen. The action is a function of two seeds: the first seed is the local state at the moment the traversal process enters the branch rooted at the current node. The second seed is the result of visiting all child branches. The action fup is to produce a seed that is taken to be the state of the traversal after the process leaves the current branch.

The two previously considered examples – computation of a string value and of a digest for a tree – can easily be written with foldts:

```
str_value32 = concat . reverse . (str_value32' [])
  where
        str_value32' = foldts id (\_->id) (flip (:))
```

In this example, the seed is the list of leaf values accumulated in the reverse order. The fhere action prepends the value of the visited leaf to the list. The actions fdown and fup are trivial: they merely propagate the seed.

```
tree_digest2 = md5Final . (foldts fd fu fh md5Init)
  where fh ctx str = md5Update "/leaf" $ md5Update str $
                                         md5Update "leaf" ctx
        fd ctx = md5Update "node" ctx
        fu _ ctx = md5Update "/node" ctx
```

The computation of the tree digest is no more complex. The seed is the MD5 context. The fdown and fup actions mark the fact of entering and exiting a non-leaf node. This example clearly demonstrates that consuming node values and updating the local state are separated from the task of traversing the tree and recurring into its branches. This separation makes operations on tree nodes simpler to write and to comprehend.

3 XML Parsing as Tree Traversal

The enhanced tree fold, foldts, has more than theoretical interest. The foldts combinator is literally at the core of the pure functional XML parser SSAX. To see how foldts applies to XML parsing, we note that an XML document with familiar angular brackets is a concrete representation of a tree laid out in a depth-first order. Elements, processing instructions, CDATA sections and character data are the nodes of such a tree. The latter three are always the leaf nodes. Attributes are collections of named values attached to element nodes. Since element nodes can be non-terminal nodes, the moments the traversal enters and leaves an element node must be specifically marked, respectively as the start and

the end tags. XML parsing then is a depth-first traversal of an XML document regarded as a tree. XML parsing is a pre-post-order, down-and-up traversal as it invokes user actions when the traversal process enters a node and again when the process has visited all child branches and is about to leave the node.

Just like the `foldts`, the SSAX framework captures the pattern of the XML document traversal (i.e., parsing). To be more precise, the framework carries out such parsing chores as tokenizing, XML namespace resolution and the namespace context propagation, the whitespace mode propagation, the expansion of character and parsed entity references, attribute value normalization, maintaining the traversal order. The user can therefore concentrate on the meaningful work – what to do at encountered nodes.

At the heart of SSAX is a function `SSAX:make-parser`, which takes user-supplied node action procedures (also called content handlers) and instantiates the corresponding XML parser. Similarly to `foldts`, `SSAX:make-parser` requires three mandatory handlers: `new-level-seed`, `finish-element`, and `char-data-handler`. These handlers closely correspond to the procedural parameters `fdown`, `fup` and `fhere` passed to `foldts`. The output of `SSAX:make-parser` is a procedure `Port -> Seed -> Seed`. The first argument is a port from which to read the input XML document. The port is treated throughout the SSAX framework as if it were a "unique" parameter, using the terminology from the programming language Clean. The second argument to the parser is the initial value of the application state, the seed. The parser returns the final value of the seed, the result of a tree-induced composition of the user-supplied handlers.

`SSAX:make-parser` also accepts a number of optional handlers, which will be called when the parser encounters a processing instruction, a document type declaration, or the root element. If the optional handlers are omitted, the instantiated parser will be non-validating. `SSAX:make-parser` is actually a macro, which *integrates* the handlers into the generated parser code. We can regard `SSAX:make-parser` as a staged parser.

The semantics of `SSAX:make-parser` is the same as that of `foldts`. Both traverse a tree in a depth-first order and invoke handlers at "interesting points". Besides the traversal state `SSAX:make-parser` also maintains the list of active entities and the namespace context. The user handlers of `SSAX:make-parser` are also more complex, receiving as additional arguments the name of the current element and its attributes.

In the following section we consider several typical instantiations of `SSAX:make-parser`, with the goal of estimating SSAX complexity and comparing it with other XML parsers. The comparison will demonstrate the benefits of modeling the SSAX parser after `foldts`.

4 SSAX Examples and Comparisons

4.1 The Complexity of SSAX Parsing

The first example of using SSAX is untagging. This is a common "XML to text" translation that removes all markup from a well-formed XML document. We

should point out that this is the same example as the one discussed in Section 2. Indeed, untagging is precisely determining the string-value of an XML document tree. The example in Section 2 operated on trees represented as linked data structures in memory. In this section a tree is an XML document itself. In both cases, we traverse the tree and accumulate all character data as we encounter them. As Section 2 explained, it is beneficial to accumulate the character data in a list in reverse order and join them at the very end.

The procedure to remove markup from an XML document is shown below. This is an instantiation of the SSAX parser with three handlers: `new-level-seed`

```
(define (remove-markup xml-port)
  ; Accumulate the text values of leaves in a seed, in reverse order
  (let ((result
     ((SSAX:make-parser
       NEW-LEVEL-SEED
        (lambda (elem-gi attributes namespaces expected-content seed)
          seed)

       FINISH-ELEMENT
       (lambda
           (elem-gi attributes namespaces parent-seed seed) seed)

       CHAR-DATA-HANDLER
       (lambda (string1 string2 seed)
         (let* ((seed (cons string1 seed)))
           (if (string-null? string2) seed (cons string2 seed))))
      )
      xml-port '())))
    (string-concatenate-reverse result)
  ))
```

and `finish-element` merely propagate the seed, while `char-data-handler` adds the character data to the list. Since the optional document-type and root element handlers are omitted, the remove-markup parser is non-validating. The pieces of character data are passed to `char-data-handler` in two string arguments for efficiency. The similarity of the remove-markup code with `str_value32` of Section 2 is striking. We must note however that `str_value32` relied on `foldts`, which traversed a linked structure of type `Tree` in memory. The remove-markup procedure on the other hand parses an XML document, which it reads from a given input port. When this port contains a document such as the one on Fig. 1, the procedure yields a string "01234567". To verify that remove-markup, just as `str_value32`, runs in time and space that grows only linearly with the size of XML documents, we applied the procedure to documents such as the one on Fig. 1 of increasing depth. We ran all benchmarks on a Pentium III Xeon 500 MHz computer with 128 MB of main memory and FreeBSD 4.0-RELEASE operating system. The benchmark Scheme code was compiled by a Gambit-C 3.0

```
<node><node><node><leaf>0</leaf><leaf>1</leaf></node>
            <node><leaf>2</leaf><leaf>3</leaf></node></node>
      <node><node><leaf>4</leaf><leaf>5</leaf></node>
            <node><leaf>6</leaf><leaf>7</leaf></node></node></node>
```

Fig. 1. A full binary tree as an XML document

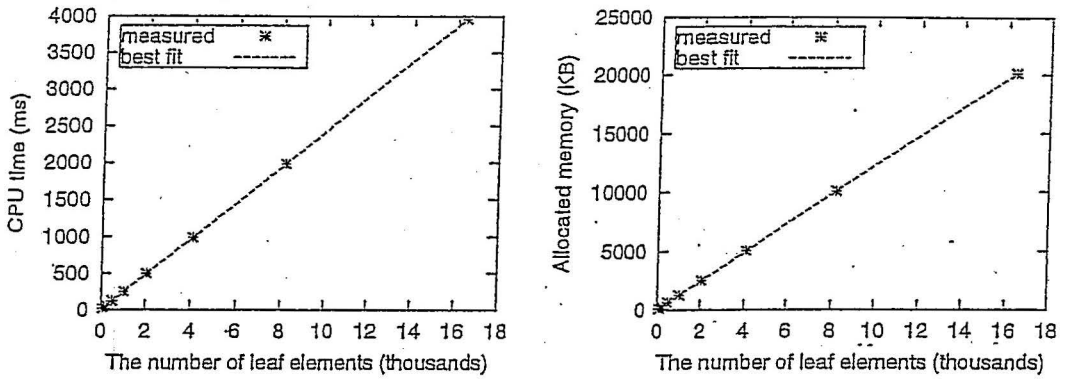

Fig. 2. Performance of the SSAX parser for documents of the form given on Fig. 1. The CPU time and the cumulative amount of allocated memory are plotted as functions of the number of leaf elements in the input XML document. Most of the allocated memory was garbage-collected

compiler. Figure 2 shows the result. The SSAX parser indeed has the linear space and time complexities. This is the *experimental* result, obtained by measuring the performance of the full-scale XML parser.

4.2 SSAX and Expat

No discussion of XML parsing can avoid Expat, which is *the* reference XML parser, written in C by James Clark [4]. Expat is a SAX, i.e., a stream-oriented parser. As the user passes it chunks of the input XML document, Expat identifies elements, character data or other entities and invokes the appropriate handler (if the user has registered one). The size of the chunks is controlled by the user; chunks can range from one byte to the whole XML document.

A tutorial article about Expat [2] explains well how Expat is supposed to be used. A user application most certainly has to have "a good stack mechanism in order to keep track of current context... The things you're likely to want to keep on a stack are the currently opened element and it's attributes. You push this information onto the stack in the start handler and you pop it off in the end handler." As an illustration, the Expat tutorial discusses a sample application `outline.c`, which prints an element outline for an XML document, indenting child elements to distinguish them from the parent element that contains them. In this case, the stack is represented by a global variable Depth, which controls

the amount of indenting white space to print before the element name. The variable is incremented in a user-supplied start-element handler, and is decremented in the end-element handler. A simplified code for the two user handlers is given on Fig. 3.

```
int Depth;
void start(void *data, const char *el, const char **attr) {
 int i;
 for (i = 0; i < Depth; i++)
      printf("  ");
 printf("%s\n", el);
 Depth++; }

void end(void *data, const char *el) { Depth--; }

int main(void) {
   XML_Parser p = XML_ParserCreate(NULL);
   XML_SetElementHandler(p, start, end); /* register the callbacks */
   /* invoke XML_Parse() passing the buffer with the XML document
     or a part of it */ }
```

Fig. 3. A simplified code for the `outline.c` application, using Expat

It is instructive to compare the Expat application `outline.c` with the corresponding SSAX application, whose complete code is given on Fig. 4.

In the Expat application, the maintenance of the application state, the Depth, is split across two separate handlers. This fact increases the possibility of an error. The `ssax-outline` application on the other hand has no global variables or other application-specific stack to maintain. Unlike the Expat handlers, SSAX handlers can be largely decoupled and thus easily written and understood.

The function `ssax-outline` also illustrates the benefit of the SAX XML parsing mode. The function prints element names as they are identified and accumulates no data. It can therefore process documents of arbitrary size – far bigger than the amount of available memory. The function `ssax-outline` is a true stream processor, with low memory requirements and low latency.

To compare the performance of SSAX and Expat, we ran several benchmarks. We need to discuss first the difference in input modes of the two parsers. An application that uses Expat is responsible for reading an XML stream by blocks and passing the blocks to Expat, specifically noting the last block. Expat requires the calling application be able to determine the end of the XML document stream *before* parsing the stream. If an application can do that, it can read the stream by large blocks. An application can potentially load the whole document into memory and pass this single block to Expat. Expat uses shared substrings extensively, and therefore is specifically optimized for such a scenario. If we take a document from a (tcp) pipe, it may be impossible to tell offhand when to stop

```
(define (ssax-outline xml-port)
  ((SSAX:make-parser
     NEW-LEVEL-SEED
     (lambda (elem-gi attributes namespaces expected-content seed)
       (display seed)                  ; indent the element name
       (display elem-gi) (newline)     ; print the name of the element
       (string-append "  " seed))     ; advance the indent level

     FINISH-ELEMENT                    ; restore the indent level
     (lambda
        (elem-gi attributes namespaces parent-seed seed) parent-seed)

     CHAR-DATA-HANDLER
     (lambda (string1 string2 seed) seed)
  )
  xml-port ""))
```

Fig. 4. The complete code for the outline application, using SSAX. The seed describes the depth of an element relative to the root of the tree. To be more precise, the seed is the string of space characters to output to indent the current element

reading. Furthermore, if we unwittingly try to read a character past the logical end of stream, we may become deadlocked. SSAX reads ahead by no more than one character, and only when the parser is positive the character to read ahead must be available. SSAX does not need to be told when the document is ended. On the contrary, SSAX will tell us when it has finished parsing a root (or other) element. SSAX can therefore safely read from pipes, can process sequences of XML documents without extra delimiters, and can handle selected parts of a document.

The performance benchmarks are based on the code to remove markup from an input XML document. This task, which was described in the previous section, simulates a typical Web service reply processing. Two input documents are XML encodings of full binary trees of depth 15 and 16. The documents are similar to the one on Fig. 1. The documents contain more markup than character data and, in addition, exhibit deeply nested elements. Overall the benchmark task is a good exercise of XML parsing engines. The first benchmark application, `string-value.c`, implements the most favorable to Expat scenario: it reads the whole document into memory, passes it to Expat and asks the parser to remove the markup. The second benchmark application, `string-value-by-one.c`, also uses Expat and also loads the whole document into memory first. The application however passes the content of that buffer to Expat one character at a time. This simulates the work of the SSAX parser. Finally, a SSAX benchmark `string-value-ssax.scm` likewise loads an XML document first, opens the memory buffer as a string port and passes the port to SSAX. The complete benchmark code is a part of the SSAX project [7]. The results are presented in Table 1.

Table 1. User/system times, in seconds, for running three benchmarks on two sample XML documents. The timing results were obtained from a precise virtual clock and reproduce within 3%. Platform: FreeBSD 4.0-RELEASE system, Pentium III Xeon 500 MHz, Bigloo 2.4a Scheme compiler. The numbers above reflect activities that occur entirely in memory. There was no i/o of any kind, there were no page faults

benchmark	XML-tree-depth-15	XML-tree-depth-16
string-value.c	0.105/0.016	0.213/0.022
string-value-by-one.c	0.747/0.014	1.494/0.012
string-value-ssax.scm	1.092/0.024	2.170/0.095
File size, bytes	884,723	1,769,459

The most notable result of the benchmarks is that a Scheme application is only 1.4 times slower than a comparable well-written C application, `string-value-by-one.c`. SSAX seems quite competitive in performance, especially keeping in mind that the parser and all of its handlers are referentially transparent. The ability to read from pipes and streams whose end is not known ahead of parsing costs performance. We do think however that the feature is worth the price. Shared substrings, present in some Scheme systems (alas not in the compiler used for benchmarking) will mitigate the trade-off.

4.3 FXP, the Functional Equivalent to Expat, and SSAX

The closest to SSAX XML parser is FXP [10], which is a purely functional, validating XML parser "shell" with a functional variant of the event-based interface. FXP is written in SML. Both SSAX and FXP invoke user-supplied handlers (called "hooks" in FXP) at "interesting" moments during XML parsing. The hooks receive an application state parameter and must return a possibly new state. The ways SSAX and FXP frameworks are instantiated to yield a specific XML processing application are also surprisingly similar, modulo static/dynamic typing. FXP "vitally relies" on SML's parameterized modules for customization while SSAX depends on Scheme's macros.

The most notable difference between SSAX and FXP is the interface between the parsing engine and the event handlers (hooks). SSAX is based on `foldts`, whereas the interface of FXP seems to be a pure functional analogue of Expat's application interface. The difference between the SSAX and FXP interfaces is important and instructive. A sample FXP application discussed at the end of the FXP API documentation [10] is a good example to illustrate that difference. The application converts an XML document to an abstract syntax tree form, which is not unlike the `Tree` datatype from Section 2. A SSAX distribution includes a similar function `SSAX:XML->SXML`. It is instructive to compare event handlers of the two applications. In both cases the event handlers are pure functional; they

receive from the parsing engine recognized pieces of markup or character data and accumulate them in a parse tree.

In the FXP application, the application data – the seed, in SSAX terminology – represent the partial document tree constructed so far. As the FXP documentation describes it, the seed has two components – a stack and the content. At any point the stack holds all currently open start-tags along with a list of their left siblings. The content component accumulates the children of the current element that are known so far. In the initial state, both components are empty. Character data event handlers add the identified character data to the content of the current element. The hook for a start-tag pushes that tag together with the content of the current element onto the stack. The element started by that tag becomes the current element. The end-tag hook reverses the content of the current element, pops the tag of the current element off the stack and combines it with its content. The constructed tree is then prepended to the content of the parent element which now becomes the current element.

The code for the comparable SSAX application is given on Figure 5. The code does correspond to the description of the FXP application to a certain extent. However, `simple-XML->SXML` is notably simpler. Whereas FXP application's state is comprised of a stack and the content, `simple-XML->SXML`'s state is a regular list. The list contains the preceding siblings of the current element or a piece of character data, in reverse document order. Maintenance of FXP's stack was split across two separate hooks: the handlers for the start and the end tags. The FXP handlers have to detect possible stack underflow errors. In contrast, the handlers of `simple-XML->SXML` are relieved of any stack maintenance and error handling responsibility: The function `simple-XML->SXML` does not have any stack. As Figure 5 shows, `simple-XML->SXML` handlers hardly do anything at all. The handler `new-level-seed` is particularly trivial; `finish-element` is not more complex either. The simpler the handlers are, the easier it is to write them and to reason about them.

We should point out that not only `simple-XML->SXML` lacks a stack, the SSAX parsing engine itself does not have an explicit stack of currently open XML elements. The traversal stack is implicit in activation frames of a recursive procedure `handle-start-tag` of the SSAX framework. If there is no explicit stack, there can be no stack underflow errors. Thus the comparison between FXP and SSAX indicates that the SSAX framework provides a higher level of abstraction for SAX XML parsing. This is the direct consequence of building SSAX around the `foldts` tree traversal combinator.

4.4 Other XML Parsers Written in Functional Languages

There are several other XML parsers implemented in functional languages: CL-XML (written in Common Lisp), XISO (written in Scheme), Tony (in OCaml), and HaXml (in Haskell). They are all DOM parsers. Neither of these parsers can process an XML document "on the fly," in a stream-like fashion.

Parser CL-XML [1] is the most thorough of the group. It checks all well-formedness and most of the validation constraints given in the XML Recom-

mendation. It is the only parser among those considered in this paper (besides SSAX and Expat) that supports XML namespaces, XML whitespace handling, general entity expansion, attribute value normalization, and the proper handling of CDATA sections. CL-XML is the least functional-style parser: it is written in imperative style, with extensive reliance on global and dynamic-scope variables.

XISO [9] is a mostly-functional parser implemented in Scheme. It is a pure DOM, non-validating parser. It is does not check many of XML well-formedness constraints either. Another parser of the similar quality is Tony [8], which is written in OCaml. It is not a pure functional parser: the parsing state and a character data accumulator are mutable. Like XISO, Tony does not detect or expand entity references, does not handle CDATA sections, does not support namespaces – it does not even handle newlines in attribute values.

One component of HaXml [12], a collection of utilities for using Haskell and XML together, is an XML parser. The parsing component includes a hand-written XML lexer, which produces a token stream for the parser proper. The latter is based on a slightly extended version of the Hutton/Meijer parser combinators. The HaXml parser does not do the normalization of attribute values and does not support XML namespaces. It does not detect many well-formedness let alone validation errors. The separation between the lexing and the proper parsing stages in HaXml is a principal weakness as tokenizing an XML document heavily depends on the parsing context. The weakness manifests itself, for example, in parser's failure to handle newlines and special characters within quoted strings.

The HaXml parser is a DOM parser. An XML document is first tokenized, then it is converted into a parse tree representation, which is handed over to

```
(define (simple-XML->SXML port)
  (reverse
    ((SSAX:make-parser
       NEW-LEVEL-SEED
       (lambda (elem-gi attributes namespaces expected-content seed)
         '())

       FINISH-ELEMENT
       (lambda (elem-gi attributes namespaces parent-seed seed)
         (cons (cons elem-gi (reverse seed)) parent-seed))

       CHAR-DATA-HANDLER
       (lambda (string1 string2 seed)
         (if (string-null? string2) (cons string1 seed)
             (cons* string2 string1 seed)))
    )
    port '())))
```

Fig. 5. A simplified SSAX:XML->SXML function from the SSAX distribution

a user application. Because Haskell is a non-strict language however, the lexer does not generate new tokens until they are required by the parser. The parser does not make a new node of the parse tree until this node is accessed in the user application code. Thus the HaXml framework *could* act similar to a SAX parser despite its multi-phase processing. This potential is not realized as HaXml eagerly loads the whole document into a string, to let the lexer backtrack or look-ahead by arbitrary amount. In contrast, SSAX never backtracks a character and never looks more than one character ahead. Therefore SSAX can handle (sequences of) documents from a TCP pipe or other stream.

5 Conclusions

In this paper we have shown an example of a principled construction of a SAX XML parser. The parser is based on a view of XML parsing as a depth-first traversing of an input document considered as a spread-out tree. We have considered the problem of efficient functional traversals of abstract trees and of capturing the pattern of recursion in a generic and expressive control structure. We have found such efficient and generic higher order operator: `foldts`. Unlike the regular tree fold, `foldts` permits space- and time-optimal accumulating tree traversals.

The `foldts` operator became the core of SSAX, a SAX parser that walks an XML document and invokes user-supplied handlers when it identified elements, processing instructions, character data and other entities. The SSAX parsing engine effectively abstracts the details of the XML document tree; the engine makes it unnecessary for user handlers to maintain their own stack of open elements; the engine reduces the amount of application state shared among the user handlers to the bare minimum. The comparison with other SAX parsers (the reference XML parser Expat and its functional analogue FXP) shows that SSAX provides a higher-level abstraction for SAX XML parsing. The user-handlers of SSAX are referentially transparent, are less error-prone to write and to reason about.

The SSAX parser is a full-featured, pure-functional, stream-oriented, algorithmically optimal SAX parser, which also makes user handlers easier to write and thus removes whole classes of possible bugs. The combination of these features distinguishes SSAX among other XML parsers. The features are the effect of the principled SSAX construction, in particularly, of the `foldts` traversal operator.

Acknowledgments

I would like to thank Shriram Krishnamurthi for valuable discussions, detailed comments and suggestions. This work has been supported in part by the National Research Council Research Associateship Program, Naval Postgraduate School, and the Army Research Office under contracts 38690-MA and 40473-MA-SP.

References

1. Anderson, J.: Common Lisp support for the 'Extensible Markup Language' (CL-XML). Version 0.906, June 2, 2001.
   http://homepage.mac.com/james_anderson/XML/documentation/cl-xml.html
   221
2. Cooper, C.: Using Expat. *xml.com*, September 1, 1999.
   http://www.xml.com/pub/a/1999/09/expat/index.html 217
3. Dunford, M.: DOM XML: An Alternative to Expat.
   http://www.phpbuilder.com/columns/matt20001228.php3 210
4. Expat XML Parser. Version 1.95.1, October 21, 2000.
   http://sourceforge.net/projects/expat 210, 217
5. Hutton, G.: A tutorial on the universality and expressiveness of fold. Journal of Functional Programming, 9(4):355-372, July 1999. 211, 213
6. Gibbons, J., Jones, G.: The Under-appreciated Unfold. Proc. Intl. Conf. Functional Programming, pp. 273-279, Baltimore, Maryland, September 27-29, 1998. 211, 213
7. Kiselyov, O.: Functional XML parsing framework: SAX/DOM and SXML parsers with support for XML Namespaces and validation. Version 4.9, September 5, 2001.
   http://pobox.com/~oleg/ftp/Scheme/xml.html#XML-parser 210, 219
8. Lindig, C.: Tony - a XML Parser and Pretty Printer. Version 0.8.
   http://www.gaertner.de/~lindig/software/tony.html 222
9. van Mourik, H.: XML parser in Scheme. Version: 0.9.23, June 2, 1998.
   http://student.twi.tudelft.nl/~tw585306/ 222
10. Neumann, A.: The Functional XML Parser. Version 1.4.4, October 30, 2000.
    http://WWW.Informatik.Uni-Trier.DE/~aberlea/Fxp/ 209, 220
11. Sheard T., Fegaras, L.: A fold for all seasons. Proc. Conf. on Functional Programming and Computer Architecture (FPCA'93), pp. 233-242, Copenhagen, Denmark, June 1993. 213
12. Wallace M., Runciman, C.: HaXml. Version 1.02 release, May 3, 2001.
    http://www.cs.york.ac.uk/fp/HaXml/ 222
13. World Wide Web Consortium. Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation October 6, 2000.
    http://www.w3.org/TR/REC-xml 209, 210
14. World Wide Web Consortium. XML Information Set. W3C Candidate Recommendation. May 14, 2001.
    http://www.w3.org/TR/xml-infoset 211

Holistic Framework for Establishing Interoperability of Heterogeneous Software Development Tools and Models

Joseph Puett
Naval Postgraduate School
833 Dyer Street
Monterey, CA 93943 USA
(831)-656-2361
jfpuett@nps.navy.mil

ABSTRACT

This research is an initial investigation into the development of a Holistic Framework for Software Engineering (HFSE) that establishes mechanisms by which existing software development tools and models will interoperate. The HFSE captures and uses dependency relationships among heterogeneous software development artifacts, the results of which can be used by software engineers to improve software processes and product integrity.

Keywords

Software Development Tool Interoperability, Software Evolution, Software Quality Function Deployment (SQFD), Relational Hypergraph Modeling, Requirement and Specification Dependency Traceability, Software Environment

1. INTRODUCTION

A great deal of software engineering research has been conducted with the aim of developing or improving individual aspects of software development. Examples include research into software evolution models, requirements engineering, risk and cost estimation, software reuse, prototyping, testing, software integration, software maintenance, re-engineering, performance analysis, domain analysis, and architecture design. Typically, these individual aspects of software development require the software engineer to recognize that dependency relationships exist and to provide any bridging between different development models and tools (see Figure 1).

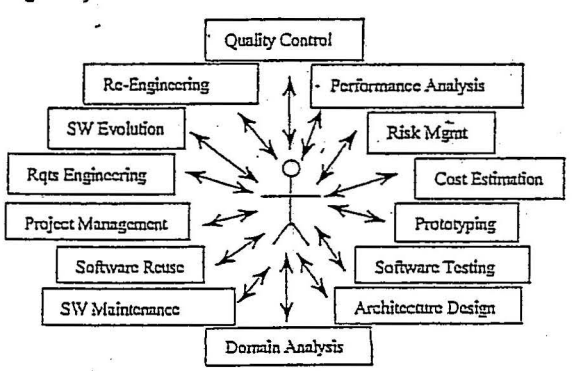

Figure 1. Typical SW Development Process Interaction

However, with limited exception [6][9], there has been little research into holistic models of how these various threads and processes could most efficiently and effectively interact. Currently, there is inadequate communication of risk and requirements across disjoint tools and models [8]. A holistic framework promises to provide seamless interoperability between these tools and models improving both process and product. The existence of such a framework enhances the discovery of dependencies among different aspects of the software engineering process and allows software engineers to implement process improvements that provide product integrity with respect to those dependencies. While the long-term goal of this research is to support all aspects of software engineering, the immediate goal is to demonstrate the theoretical feasibility of integrating a selected subset of models and tools using a Holistic Framework.

2. HOLISTIC FRAMEWORK

2.1 Overview

Central to a holistic view of software development is software evolution. A software-evolution system must provide strong version control of all artifacts produced during system development and track dependencies among artifacts. In today's distributed development environments, the evolution-control system must provide for collaboration between multiple users at multiple sites, provide mechanisms for notification when changes made by one developer affect the work of another, and when appropriate, provide blocking when on-going work of one developer would be counter-productive to attempted work by another. The artifacts to be controlled vary in both purpose and format. Examples include: organizational policy and vision documents, business case documents, development plans, evaluation criteria, release descriptions, deployment plans, status assessments, user's manuals, requirements and specifications, customer interviews, meeting minutes, code, software documentation, software architecture document, unit tests, test cases, and test results. The formats vary as well: data base entries, text documents, spreadsheets, images, drawings, audio files, and video clips. The long-term goal of the HFSE is to establish dynamic traceability, dependency tracking, and integration over this diverse set of information and formats.

By relating inputs and outputs of various software process models through an evolution interface that attaches and records the dependencies among evolution artifacts [4], information required by various processes can be automatically generated and obtained as needed. Such a model requires interaction between a GUI, an evolution control component, and an object model component.

The Evolution Model and Object Model interact with subordinate software development tools and processes (see Figure 2).

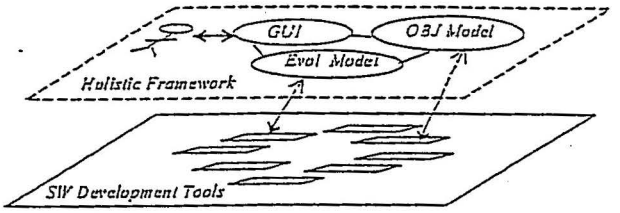

Figure 2. Holistic Model of SW Process Interaction

Considerations in establishing this higher level holistic framework include identifying the medium for representation of information (e.g., tree structure), establishing a communications medium (e.g., net, databases, publish and subscribe w/ CORBA, object mechanisms using XML), accounting for process order (e.g., sequential, parallel, hybrid), providing missing data, accounting for ambiguity of inputs and outputs, accounting for conflict resolution between models, and providing for extensibility.

2.2 The Ideal HFSE.

The ideal Holistic Framework for establishing interoperability of software development tools and models should include the following characteristics:

2.2.1 Non-proprietary

The HFSE should be generic and non-proprietary. The framework should allow any model or tool to be incorporated. The framework should not be established solely for use with a specific group of tools.

2.2.2 Support Real Tools

The HFSE should support real software development tools. The framework should not be established simply to support research/laboratory software development tools but must account for tools used to build real software.

2.2.3 Process Independent

The HFSE should be independent of the software development process. The framework should be of benefit regardless of whether a software development team uses the spiral development model, the evolutionary prototyping model, the waterfall model, the rapid application development model, the Win-Win model, etc.

2.2.4 Domain Independent

The HFSE should be able to integrate tools for any software domain. The framework can be used equally well to link together tools that are used to design and build command and control systems, or embedded real-time systems, or information system applications. This is not to suggest, though, that tools integrated for use with a particular domain could be successfully applied to develop software in a different domain.

2.2.5 Extensible

The HFSE should be extensible. Not only should it be possible to include new process models or tools by using the framework, but it should also be possible to modify or update the attributes of the framework based on new technologies or new attributes required by new process models.

2.2.6 Reduce Time to Market

Use of the HFSE should improve the speed in which software is produced.

2.2.7 Reduce Development Cost

Use of the HFSE should decrease the cost of developed software.

2.2.8 Improve Product Quality

Use of the HFSE should increase the quality of developed software.

2.2.9 Easy to Use

The HFSE should be intuitive and easy to use.

2.3 Research Scope.

Of these ideal characteristics, only some will be directly addressed and validated by this research. In particular, the characteristics described in paragraphs "2.2.1-2.2.5" above will be used to establish the framework. Mathematical relationships (principally from Graph Theory, Object Oriented Analysis & Design (OOAD), and Category Theory) will be used to demonstrate the validity of established constructs. The characteristics described in paragraphs "2.2.6-2.2.9" will be only lightly addressed in the dissertation, but there are plans for formal validation of these efficiencies to be undertaken as future research.

3. HYPOTHESIS.

The following is a statement of the research hypothesis:

It is theoretically feasible to integrate a selected set of software development tools and/or models through application of a Holistic Framework for Software Engineering (HFSE). Where:

- The HFSE consists of an extended Software Evolution model integrated with a Federation Interoperability Object Model of the subordinate software development tools/models.
- The integrated tool/model set provides additional interoperability (i.e. additional data exchange and joint task execution) beyond the interoperability available prior to the application of the HFSE to the set.

4. RESEARCH PLAN & METHODOLOGY.

Conducting this research consists of executing the following major tasks: 1) Identify and holistically define the essential characteristics of individual software development process models and tools, 2) Embed Quality Function Deployment in the Relational Hypergraph Software Evolution Model, 3) Apply the Object-Oriented Model for Interoperability for heterogeneous systems to establish an interoperability federation between software development process models, 4) Integrate the extended Evolution model and the object federation, 5) Prototype the HFSE, and 6) Apply the HFSE to a selected set of tools to establish evidence that the interoperability of the integrated tool set is improved.

4.1 Characterizing Tools & Models.

The first major step in this research is to identify and define the essential characteristics of software development process models and tools so that these characteristics can be used to appropriately extend the Relational Hypergraph Software Evolution Model and

be used to construct an Object Federation. The approach to this portion of the investigation is to analyze the structure, inputs, and outputs of a small collection of individual tools. To provide a breadth of coverage of software development (yet, manage the scope of the investigation) five tools will be analyzed. These will be drawn from the areas of requirements engineering [9], project risk [8], prototyping and implementation [7], reuse, and testing. A methodology that may prove useful here is to perform a domain analysis (of this subset of tools) and produce a feature model of that domain [2]. Next, these essential features will be compared and refined in the context of the objects defined in the Evolution Model; their common characteristics, structures, and relationships as described by Category Theory [5]; and the objects needed for establishing an Object Federation [10]. Such comparisons will provide a holistic descriptive model of the essential characteristics of software development tools.

4.2 Embedding QFD in the Evolution Model

The next major step in the research requires establishing dependency relationships between software development constructs within an evolution context. One way of developing this evolution interface is by extending an existing Software Evolution model [4] with Quality Function Deployment (QFD) [3] to introduce a continuum of dependencies between software artifacts. Existing models rely on predefined artifacts and limited dependency tracking. A QFD continuum separates relevant dependencies/priorities from noisy data and is an improvement over current models [4] that only provide primary and secondary dependencies with no articulation as to importance (and type) of the dependency to the rest of the design. The HFSE will provide semi-automated mechanisms for establishing the continuum of dependencies among software development artifacts. Such an extension also improves the vertical, horizontal, and temporal dependency graph between these artifacts.

4.3 Establishing the Object Model

Finally, it is necessary to develop an interaction framework between the subordinate process models and the extended evolution model. One promising approach is to use an Object-Oriented Model for Interoperability (OOMI) for resolving representational differences between heterogeneous systems [10]. This approach establishes a high-level Federation Interoperability Object Model (FIOM) that allows interaction between the objects of existing heterogeneous systems. By establishing such an object federation between existing process models (or their tools) and then integrating that federation with the extended evolution model, inputs and outputs between the subordinate models (or tools) will be available to each other while at the same time reporting that interaction to the extended evolution model. Our approach is similar to that of the High Level Architecture (HLA), but applied in a different context. The success of this research will help clarify the tradeoff between interoperability via conformance to a single global data standard versus the use of multiple representations, ontologies, and translations.

5. EVALUATION

Evaluation of the research will be initially undertaken by constructing a prototype HFSE integration tool, used for integrating subordinate software development tools. The HFSE prototype will be applied to a small representative subset of tools/models forming an integrated software development environment. The integrated tools will then be used in a software development scenario. Evaluation will be undertaken against a control group to provide evidence that the interoperability of the integrated tool suite is improved. Finally, the research will attempt by theoretical arguments to characterize the class of tools and models that could also be unified with additional effort.

5.1 Experimental Design

After the HFSE integration tool prototype is constructed, a static group comparison test [1] performed as a laboratory experiment will be used to provide confirming evidence of the research hypothesis. In this case, the HFSE prototype (the experimental variable) is applied to a selected subset of tools/models (the observation group). The performance of the integrated subset of tools/models (after the application of the HFSE) is then compared to the performance of the same tools/models operating without the benefit of integration by the HFSE when applied in a software development scenario. The comparison in this case will be to determine if there are any improvements in interoperability between the tools. Specifically, the research will be accumulating evidence of additional data exchange and additional joint task execution enabled by the application of the HFSE to the subset of tools/models. The evaluation will also be seeking counter-evidence that the HFSE reduces (or inhibits) data exchange and/or joint task execution.

5.2 Internal and External Validity.

[1] identifies the conditions for which scientifically sound experimentation should occur. In order for an experiment to be scientifically sound, the experiment must bound sources of internal and external invalidity. Internal validity deals with the question of whether or not the application of the process (the HFSE) was, in fact, the sole direct contributing cause of the measured result (improvements to tool interoperability). External validity deals with the question of whether the result can be generalized to external populations and sets outside the experiment. The static group comparison proposed controls some (but not all) of these sources of invalidity.

5.2.1. Sources of Internal Invalidity

5.2.1.1 History

This source of internal invalidity arises because of specific events occurring between measurements of the outcome that are in addition to the experimental variable. This source is not adequately controlled during the proposed experiment because once the tool set is integrated by the HFSE, its state may change between time periods that we search for evidence of improvements in interoperability of the observation group. The state may change because we will be seeking interoperability improvements during an active evolutionary software development effort. While unlikely to be the cause, these state changes cannot be ruled out as the effect of additional improvements in interoperability. The only way to control this source of invalidity would be to measure all changes in interoperability at each atomic time *step (perhaps every second)* of the software development effort. Given that the main interest is in establishing evidence in a complex evolutionary development effort (perhaps measured in days, weeks, or months), such an approach is impractical. Instead, we will attempt to mitigate this source of invalidity by attempting to determine and document the direct cause of each improvement to interoperability.

5.2.1.2 Maturation

This source of internal invalidity arises because of processes within the observation group change as a function of time, independent of any application of the HFSE. Again, this source of invalidity is not controlled and cannot be ruled out because software development process tools may have internal processes which are activated solely by time (e.g. automatic updating/rectifying of databases). Such processes will change the state of the observation group, meaning that it may not be possible to establish that the direct cause of differences in the observation group were a result of the HFSE. As in the case of "History," we will attempt to mitigate this source of invalidity by attempting to determine and document the direct cause of each improvement to interoperability.

5.2.1.3 Testing

This source of internal invalidity arises when the act of taking an observation changes the state of the observed item and thus influences future observations. This source of invalidity is adequately controlled since observing the evidence of improved interoperability within the tool set is unlikely to generate any changes to the state of the set.

5.2.1.4 Instrumentation

This source of internal invalidity arises because of changes in the observing instrument or changes in the observers create a bias between measurements. This source of invalidity could be a problem in the proposed experiment since we will be looking for "improvements in interoperability" -- perhaps influenced by subjective opinion. The key for controlling this source of invalidity is to carefully define what an "improvement" means, to define what "interoperability" means, and to uniformly apply these definitions to the comparison set. We will begin with the following definitions:

- Interoperability: data exchange and/or joint task execution between two separate tools/models
- Improvement: evidence of the existence of interoperability found in the integrated tool/model set, not found in the disjoint tool/model set.

If these definitions prove insufficient to explain or account for witnessed phenomena during the experiment, the definitions will be refined appropriately.

5.2.1.5 Statistical Regression

This source of internal invalidity arises when the observation sample group has been selected from the extremes of the potential observation population. Since the tools/models selected are more appropriately termed a "convenience" sample, this source of invalidity is not applicable to this experiment and therefore is controlled.

5.2.1.6 Selection Biases

This source of internal invalidity arises because of biases in the selection of the observation group. As mentioned in paragraph 5.2.1.5, our observation group of tools is a convenience sample, chosen primarily on the basis of the availability of the tool/model. Several of the tools/models were specifically developed with a view that they could be eventually integrated (the prototyping tool [7] and risk management model [8]). Because we have selected these tools/models for observation based on this bias, this source of invalidity is present in the experiment and therefore is not controlled. To mitigate this somewhat, an outside model was chosen as the requirements engineering model. This bias may be further mitigated by selecting representative, yet external, tools/models for the reuse and testing portions of the experiment.

5.2.1.7 Experimental Mortality

This source of internal invalidity arises when there is a loss of part of the observation group during the experiment. It is not expected that application of the HFSE will result in the loss of any tool/model; therefore, this source of invalidity should not occur and therefore is controlled.

5.2.1.8 Selection-Maturation Interaction

This source of internal invalidity arises in multi-observation group experiments when interaction between the observation groups is mistaken for the effect of the experimental variable. Since this experiment does not involve the interaction of the two observation groups, it cannot occur and therefore is controlled.

5.2.2 Sources of External Invalidity

5.2.2.1 Interaction of Testing and the Experimental Variable

This source of external invalidity arises when a pretest might change the observation groups' responsiveness to the experimental variable. In this case, no pretest is applied to the observation group, therefore there is no opportunity for bias. Thus, this source of invalidity is controlled.

5.2.2.2 Interaction of Selection and the Experimental Variable

This source of external invalidity arises because of interaction effects between the selected observation group and the experimental variable. In this case, the experimental variable is the application of the HFSE prototype, which has as its core the Hypergraph Evolution Model [4]. This model, the prototyping model [7], and the risk model [8] were originally designed to work together. Therefore, their interaction may unfairly bias the generality of the result. Thus, this source of external validity is not controlled and brings into question the application of the HFSE on other, randomly selected tools/models.

5.2.2.3 Reactive Arrangements

This source of external invalidity arises because of the reactive results of experimental arrangements. For instance, if we choose tools for the experiment with APIs (for experimental convenience and because we believe they will be easier to integrate with the HFSE prototype), we cannot then conclude that the result of the experiment is generally applicable to all tools (for instance, those without APIs). Such a situation does exist in this experiment because at least two of the tools/models (the prototyping tool [7] and the risk model [8]) have been chosen for their experimental convenience. Thus, this source of invalidity is not controlled. We are attempting to mitigate the degree of this external invalidity by selecting some commercial tools in addition to those tools.

5.2.2.4 Multiple Treatment of the Experimental Variable Interference

This source of external invalidity arises when there are multiple treatments of the experimental variable on the same observation group. Since the HFSE will only be applied once to a particular

set of tools, there is no opportunity that this source of invalidity will occur. Thus, it is controlled.

5.2.3 *Summary of Experimental Validity*

Table 1 (below) summarizes the sources of invalidity associated with the proposed experiment.

Table 1: Summary of Sources of Invalidity

Sources of Internal Invalidity	History	Not Controlled
	Maturation	Not Controlled
	Testing	Controlled
	Instrumentation	Partially Controlled
	Regression	Controlled
	Selection	Not Controlled
	Mortality	Controlled
	Interaction of Selection & Maturation	Controlled
Sources of External Invalidity	Interaction of Testing and Exp Variable	Controlled
	Interaction of Selection and Exp Variable	Not Controlled
	Reactive Arrangements	Not Controlled
	Multiple Treatment of Exp Var Interference	Controlled

It is evident, that even with mitigation measures that there are a number of sources of invalidity. Because of limitations in the scope of this research, these sources of invalidity will not be formally addressed (only discussed). However, there are plans for future research to address these shortcomings.

6. CONTRIBUTION

The most important original contribution to the field of Software Engineering that this Dissertation proposes is to establish the feasibility of a Holistic Framework that captures dependency relationships between software artifacts so that the relationships can be visualized and leveraged. Establishing an HFSE and applying it to a set of software development tools or models will improve the efficiency and effectiveness of software development in a number of ways. First, the entire process of software development will become more automatic. As long as model/tool inputs and outputs can be supplied through the holistic model, different tools will be able to interact automatically, with less involvement by the software engineer. Second, because all artifacts within the holistic model are tracked together as a large dependency graph, it is possible to extract select "slices" of the dependency graph for particular purposes, allowing more "focused" development. For example, since the holistic model interacts with existing process models such as software risk, reuse, and testing; it will then be possible to extract a "slice" of the entire dependency graph (a slice that represents the greatest risk) so that prototyping and analysis effort is not wasted on developing artifacts that are already well defined, understood, and/or successfully implemented in previous versions. Finally, such a framework will allow software engineers to identify and reason about previously unknown relationships between software development artifacts leading to both process and product improvements.

7. ACKNOWLEDGMENTS

This dissertation research is sponsored in part by the Space and Naval Warfare Systems Center, San Diego.

8. REFERENCES

[1] Campbell, D. T. and Stanley, J. C., Experimental and Quasi-Experimental Designs for Research, Houghton Mifflin Company, Boston, 1963.

[2] Czarnecki, K. and Eisenecker, U.; Generative Programming: Methods, Tools, and Applications; Addison Wesley, 2000.

[3] Haag, S., Raja, M. K., and Schkade, L. L., "Quality Function Deployment Usage in Software Development," *Commun. ACM*, 39, 1, (1996), 41-49.

[4] Harn, M., Berzins, V., and Luqi, "Software Evolution Process via a Relational Hypergraph Model," Proc. IEEE/IEEJ/JSAI Int. Conf. on Intelligent Transportation Systems (Tokyo, Japan, Oct. 5-8, 1999), 599-604.

[5] Krishnan, V. S., An Introduction to Category Theory, North Holland, New York, 1981.

[6] Kruchten, P., "A Rational Development Process," *CrossTalk*, 9(7), July 1996, STSC, Hill AFB, UT, pp. 11-16.

[7] Luqi and Ketabchi M., "A Computer-Aided Prototyping System" *IEEE Software*, March 1988, pp. 66-72.

[8] Nogueira de Leon, J.C., "A Formal Model for Risk Assessment in Software Projects," *PhD Dissertation*, Naval Postgraduate School, Sept 2000.

[9] Rational Corporation, Rational Unified Process: Best Practices for Software Development Teams, Report TP-026A, rev. Nov 1998.

[10] Young, P., Ge Jun, Berzins, V., Luqi, "Using an Object Oriented Model for Resolving Representational Differences between Heterogeneous Systems," *Proc. Monterey Workshop* (Monterey, Calif., June 2001), 170-177.

Optimizing Systems by Work Schedules (A Stochastic Approach)

William J. Ray
Naval Postgraduate School
833 Dyer Road
Monterey, CA 93943
(831) 656-2509
wjray@nps.navy.mil

Luqi
Naval Postgraduate School
833 Dyer Road
Monterey, CA 93943
(831) 656-2735
luqi@nps.navy.mil

Valdis Berzins
Naval Postgraduate School
833 Dyer Road
Monterey, CA 93943
(831) 656-2610
berzins@nps.navy.mil

ABSTRACT

Many systems have very predictable points in time where the usage of a network changes. These systems are usually characterized by shift changes where the manning and functions performed change from shift to shift. We propose a pro-active optimization approach that uses predictable indicators like manning schedules, season, mission, and other foreseeable periodic events to configure distributed object servers. Object-Oriented computing is fast becoming the de-facto standard for software development and distributed object servers are becoming more common as transaction rates increase.

Optimal deployment strategies for object servers change due to variations in object servers, client applications, operational missions, hardware modifications, and various other changes to the environment.

As distributed object servers become more prevalent, there is more need to optimize the deployment of object servers to best serve the end user's changing needs. A system that automatically generates object server deployment strategies would allow users to take full advantage of their network of computers.

The proposed method profiles object servers, client applications, user inputs and network resources. These profiles determine an optimization model that is solved to produce an optimal deployment strategy for the predicted upcoming usage by the users of the system of computers and servers.

The validity of the model was tested by experimental measurement. A test bed was created and different manning schedules were simulated. The results of the experimentation showed that the average response time for a user could be improved by altering the deployment of the servers according to the scheduled manning of the system. The model was robust in the sense that the deployments that produced optimal response times in the model also produced optimal or near-optimal response times in the actual implementation of the test-bed.

WOSP '02, July 24-26, 2002, Rome, Italy.

Categories and Subject Descriptors

Primary Classification: D.1 [Programming Techniques]

Additional Classifications: D.1.3 [Programming Techniques]: Concurrent Programming - *distributed programming*, D.2.12 [Software Engineering] Interoperability – *distributed objects*, D.3.4 [Programming Languages]: Processors – *optimization*, D.3.3 [Programming Languages]: Language Constructs and Features – *concurrent programming structures.*

General Terms

General terms are as follows: Measurement, Performance, Design, and Experimentation.

Keywords

Keywords are as follows: Object-Oriented Programming, Stochastic Optimization, Distributed Computing, Load Balancing, and Performance Tuning.

1. INTRODUCTION

The future of computing is heading for a universe of distributed object servers. The evolution of object servers to distributed object servers will parallel the evolution of the relational databases. Over time, object servers will provide functionality to more client applications than their original applications, just as relational databases were used by more applications than the original application. In both cases, systems optimized for the original application may not perform well for the new applications. Tools that allow a programmer to model an object and easily create object servers with all the necessary infrastructure code needed to work as a distributed object server are available [15]. This will lead to an explosion in the number of object servers available to client applications.

A user's network of computers will change frequently. Object servers, applications, hardware and user preferences will be in a constant state of flux. No static deployment strategy can efficiently use the assets accessible on the network in such an environment.

No system can accurately predict user interaction with a system. Two separate users performing the same job will interact with a system differently. The same user may interact differently while performing the same job at different times. System usage patterns are often fusty and unstable so that the measured usage pattern for

a given time intervals is often a poor predictor of the pattern in the next time interval. For this reason, we propose an adaptive approach using a relatively simple approximate model.

Most deployment strategies today are statically dictated by the system engineer's view of how the systems will be utilized. Of course, the system engineer doesn't revisit these strategies every time hardware, software or user interactions change. Our goal is to allow the user to update usage, hardware and software profiles whenever necessary. Any time a profile is updated, the model would be run and an automated reconfiguration of the object server deployment could occur. In most cases, the frequency of change will be greatest in the hardware and usage pattern profiles. Since many of these changes can take place without the knowledge of a system engineer or the budget to employ one, a method that allows the users to update these profiles and initiate the reconfiguration is desired.

2. PREVIOUS WORK

There has been little work on deployment strategies for distributed object servers. The closest relevant research is in the fields of load balancing, client/server performance and distributed computing. Most state of the art load balancing techniques address scheduling of given set of tasks on a set of given machines. Some techniques only deal with tasks that are independent. Others deal with dependent tasks that are usually linked together by temporal logic and mutual exclusion constraints [6, 7, 11].

Other approaches to decreasing the average client response time include the use of replicas or clusters. These techniques usually involve making replicas of servers and distributing these replicas across machines. The optimizations then look at balancing requests across the replicas [6]. These techniques require additional hardware resources and add complexity to the architecture. Synchronization of replicas requires two-phase commits in order to guarantee consistency of data. The strategy works best when read-only queries predominated. As the update rate increases, the level of performance can deteriorate quickly.

Many vendors claim to address optimization within their products. Most of these involve the employment of replicas and clusters embedded in the logic of their EEJB, CORBA or DCOM enterprise tools, like Allaire's Jrun or Borland's Visibroker. These products work best if your system has just a few stateless object classes with numerous instances and plenty of available hardware. IBM's Distributed Application Partitioning (DAP) automatically determines how to place objects in a distributed program. DAP monitors the execution and records how often particular objects communicate with each other. Then it computes an ideal placement by determining the minimum cut set of a graph. While these products have value, they are limited to optimizing servers implemented within their tool. The ability to reason about performance over a mixed bag of object servers *regardless* of middleware (EEJB, CORBA, DCOM) implementation was not found in any product or previous research.

Research in optimization of distributed, real-time systems is also widely available. This research is aimed at real-time systems where the optimization is directed at the scheduling of tasks, similar to many load balancing techniques. In non-real time systems where user interactions dictate the majority of the tasks, deterministic scheduling of tasks is impractical. Conversely, moving object server locations around in a distributed, real-time system is often impractical. For these reasons, this work is directed at the non real-time arena.

Other approaches to improving the performance of servers include hardware improvements. These approaches usually involve shared-memory multiprocessor systems. While research focused on hardware, such as the Cache Coherent Non-Uniform Memory Access (CC-NUMA), does improve the performance of object servers, these solutions are not an option for most system engineers [5]. Much of the research involved in shared-memory multiprocessor systems relies on the existence of the fast, reliable shared-memory, which doesn't exist in a heterogeneous network of low cost computers. Multi-processor systems are orders of magnitude more expensive than single CPU systems. While these systems may be the only option for large monolithic servers, multi-server architectures can distribute their servers across much cheaper single CPU systems to gain needed performance.

Research in Grid Computing also has emerged as an important new field in distributed computing. Large-scale resource sharing across multiple organizations increases both the set of available network resources and the complexity of the underlying architecture. The need for authentication, authorization, resource access, resource discovery, and other challenges require applications to conform to "intergrid protocols" [3,4]. While these added complexities would be needed for environments like the Internet, they are not as useful in much smaller, single organization environments.

Most of the previous work relies on estimating future loads by measuring past loads. Our approach augments this with profiles of predicted usage patterns that can be chosen based on higher-level context information such as current mission.

3. CURRENT PRACTICES

Because of the difficulty in producing the infrastructure code necessary to support distributed object computing, many developers produce huge monolithic object servers [14]. A powerful machine is usually needed to adequately handle this server and successful applications that experience large increases in the number of users may outgrow the capabilities of the fastest available single machine.

With automated code-generation tools, servers will be much easier to produce and reconfigure [15]. This allows servers to be partitioned by allocating unrelated or loosely related objects types to different physical servers that can be deployed across the network to take advantage of the available assets. By taking advantage of all the assets on the network, faster response times can be achieved [14].

Loosely related object types are object types that contain associations to other object types. When these object types reside in different physical object servers, the result is an object server that calls on other object servers. A server that calls other servers is a complex server [1].

Many networks of computers are installed with a single purpose in mind. Over time, these networks support an evolving set of tasks. Even though the original role the network played can change dramatically, rarely does a single system engineer revisit the deployment strategy for the entire system. What a user ends up

with is usually the product of multiple application engineers' choices made based on the latest incremental changes without regard for the system as a whole and interactions among its roles. It is infeasible, because of cost, to hire a system engineer to re-assess the whole system every time a change occurs. In the end, the user is left with a system whose deployment strategy borders on randomness.

Any proposed solution to increasing performance can run into resistance if the cost is too great. System engineers regularly dismiss solutions that require re-implementation of the software. Re-implementation can occur either by changing the architecture or by conforming to a single implementation. The costs incurred by changing software can make even the most expensive hardware more attractive. Collection of the necessary information is also a concern. The more intensive and time consuming the collection becomes, the less likely the solution will be used. In the end, it becomes a delicate balance to increase model fidelity without increasing this burden on the system engineer.

4. OPTIMIZATION OF DISTRIBUTED OBJECT-ORIENTED SYSTEMS

The goal of this paper is to describe a method that can generate distributed object oriented server deployment architectures to take advantage of network resources for the purpose of reducing average client response time. A system that carries out this method must be able to reason about deployment strategies of loosely related objects. The proposed system maps all of these profiles into equations to minimize average client response time.

Average client response time was chosen as the optimization criteria over others. In this paper, the goal was to be user centric. Criteria that focused on maximizing machine utilization were not germane. Average client response time was chosen over minimizing the maximum response time of one call because the method takes into account the entire usage profile.

Our approach is to collect base system data by measurement to calibrate our model. The model is used to predict optimal deployment patterns for given usage patterns, hardware, etc. To validate the model presented in this paper, all possible deployment patterns were implemented and measured in a real test bed. The goal is to show that by minimizing the sum of the projected CPU load on the servers and their interactions, that we can lower average client response time.

4.1 Optimization Model

The optimization model will minimize the sum of all of the response times for a given call pattern over a given time interval. Since we want to allow the user the freedom to run client applications from anywhere on the network, we will ignore all processing on the client machines and all network delay between client machines and server machines. Queuing delay is addressed in a simplistic manner by limiting CPU utilization. Limiting this model to a local area network minimizes the importance of latency. The only factors we will consider for optimizing our server deployment are the processing on the object server and the network delay between complex object servers. Therefore, the objective function that we wish to minimize is:

$$Minimize \left[\sum_{n=0}^{N} \sum_{m=0}^{M} \frac{a_{nm} * R_n * S_{norm}}{S_m} + \sum_{i=0}^{N} \sum_{j=0}^{N} \frac{B_{ij}}{Q_{ij}} \right]$$

subject to the following four constraints:

1. Object Servers cannot be split across machines.

   a_{nm} = 1, iff server n is running on machine m

   a_{nm} = 0, otherwise

2. Each Server can run on only one machine (no multiple instances of the same server).

$$\forall n \left[\sum_{m=0}^{M} a_{nm} \equiv 1 \right]$$

3. RAM usage by the object servers cannot pass a set threshold on each machine.

$$\forall m \left[\sum_{n=0}^{N} a_{nm} * V_n \leq T_m * U \right]$$

4. CPU time on a given machine cannot surpass the corresponding real time interval.

$$\forall m \left[\sum_{n=0}^{N} \frac{a_{nm} * R_n * S_{norm}}{S_m} \leq C \right]$$

where:

N = Number of object servers

M = Number of physical machines

R_n = Normalized machine load of server n (seconds, s)

S_{norm} = Speed of the normalizing machine (MHz)

S_m = Speed of machine m (MHz)

B_{ij} = Data sent between server i to server j (bits, b)

Q_{ij} = Network Speed between server i to server j (bps)

T_m = Physical RAM on machine m (bits, b)

V_n = Memory allocated by server n (bits, b)

U = Multiple to limit RAM utilization [0.1,3.0]

C = Time Interval [seconds, s]

NOTE: The optimization process varies all a_{nm} and finds the minimum for the above objective function and constraints. Q_{ij} is dependent on a_{nm}. It is a function of the relative location of the two servers. Depending on this function, the system of equations may be linear or non-linear. For the examples in this paper, $Q_{ij} = \left(1 + \sum_{m=0}^{M} a_{im} \cdot a_{jm}\right) * L$, where L is the LAN speed. All other terms are fixed either by measurement or input.

4.2 Evolution

Over time, a collection of hardware, software and user requirements will change in a given environment. Common hardware changes consist of adding new computers, removing old computers, upgrading CPUs, modifying RAM and modifying network bandwidth capacity. Each of these hardware changes

will produce an event that would trigger the system to re-evaluate its deployment strategy.

Software can also be quite dynamic in nature. New object servers and applications can appear. Old ones can be removed. Existing object schemata and methods can be changed. Each of these changes would trigger an event to re-evaluate the deployment strategy.

4.3 Loosely Related Objects

Not all objects types that are related must necessarily be contained in a single object server. There is a point where the performance of the system would improve by moving the object type into a different server. This is usually the case when none of the application code exercises an inter-server method call or exercises it only very rarely. Large message sizes and slow network speeds will push for related object types to be co-located. Large queuing delays and increased swapping costs work to spread object types apart. The approach will be able to reason about not only deploying object servers, but also recommend the schema supported by these object servers.

4.4 Roles and Usage Patterns

User requirements can also be in a state of flux. Most computer systems are used to support multiple jobs. Business-hour requirements can differ greatly from after-hours computational requirements. A developer's network of computers can support multiple projects, but may need to be optimized for a single project for demonstrations. In the military, the operational mission being supported can change significantly. For example, a set of distributed object servers could be used to support many applications aboard a ship. These applications could handle such tasks as Anti-Submarine Warfare (ASW), Anti-Surface Warfare (ASUW), Anti-Air Warfare (AAW), Electronic Warfare (EW), humanitarian missions and rescue missions. The relative computational activity of these applications could differ significantly on different missions of the ship.

Optimizing a system of object servers for all possible roles would not be optimal when the system is only performing a couple of missions at a time. By profiling each role, the user could choose to re-optimize his deployment to decrease the response time when user chosen roles change. In this way, the user could tune his system to give peak performance for the task he is currently trying to perform.

4.5 Profiles

The tricky part is to figure out what elements are needed in the different profiles, how to map these profiles into equations and then model how these profiles interact with each other. The more complex the modeling of the hardware becomes the more computationally intensive the approach will become. Initially we demonstrate an approach with rather simplistic profiles to demonstrate its capabilities.

4.5.1 Hardware Profiles

The aspects being modeled in the hardware profiles include characteristics of each computer such as CPU speed and physical RAM size. The hardware profile also models the network speed between each computer. Current hardware profiles do not directly support multi-processor computers, but they could be modeled as groups of separate nodes with very high "network speeds" between them.

4.5.2 Object Server Profiles

Object servers need to be profiled for metrics associated with each method call in each object. The computational time of each method call should be captured and normalized to a specific hardware architecture. Since object servers ideally run continuously, the RAM of the object server must also be measured and summarized. The hardware profile and the object server profile are sufficient to optimize the server deployment for the case where all the functionality contained in all the objects is of equal value to the user. Metrics can be collected easily with a small client application that exercises each method call and records the data. Thus, actual implementation code for the application isn't needed to estimate the object server profiles.

4.5.3 Client Application Profiles

A client application profile characterizes frequencies of method calls to be processed by the object servers. Since exact frequencies of method calls are not algorithmically computable in the general case, measurement is necessary to reliably estimate frequencies of calls. The system must allow a user to create typical scenarios and record the method calls that occur in the scenario. This could be done by simulation or monitoring calls to the object servers when the system is in a training mode. The plus side to this method is that the user could represent more complex tasks involving many user interactions in a single profile. Numerous tools exist for complex event processing in a distributed system that could be used to facilitate this process [8, 9].

4.5.4 User Profiles

User profiles or roles indicate how a user interacts with the system over a given period of time. In simplistic terms, it is like keeping track of how many times each button is selected over a given time interval. Average button push rates can be expressed as number of events per second. The user can collect this data manually or via automatically collected audit trails. Multiple roles can exist for each user. The user could then select a set of roles and have the system come up with an optimal deployment strategy to meet these criteria.

4.6 Profile Mappings

In order to compute the optimal deployment strategy given a set of profiles, one needs to map these profiles into equations that can be solved for minimum response time. To illustrate the mappings, we present an example. The example consists of three machines, three object servers and three client applications. The method demonstrates the differences in deployment for a system tuned to a users-specific role. Table 1 shows the profile for the computer hardware available.

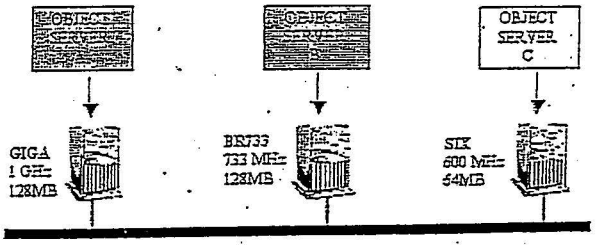

Figure 1: Server Deployment.

Table 1. Machine profile for example.

MACHINE	RAM (bits)	CPU Speed (MHz)
SIX	512,000,000 = 64MB	600
BR733	1,024,000,000 = 128MB	733
GIGA	1,024,000,000 = 128MB	1000

Table 2 shows the network bandwidth available to communicate from each machine to the other. In this example, the machines will have equal bandwidth between machines, as is the case when all servers are running on the same local LAN. The speed of communications between servers on the same machine is more difficult to predict. These speeds usually lie in the interval bounded by the speed of the machine's back plane and the speed of the network. It is dependent on the operating system, implementation of the middleware, and other factors. For this example, we assume that intra-machine communication is twice as fast as inter-machine communication. In the absence of measurements, the system can be run with best and worst-case scenarios by specifying the boundary values identified above.

Table 2. Network speed.

Machine to Machine Speed (bps)	SIX	BR733	GIGA
SIX	200,000,000	100,000,000	100,000,000
BR733	100,000,000	200,000,000	100,000,000
GIGA	100,000,000	100,000,000	200,000,000

Besides the hardware profiles, we need to have the server profiles. Table 3 lists each server's RAM requirements.

Table 3. Server RAM requirements.

SERVER	RAM Required (bits)
A	352,000,000 = 44MB
B	480,000,000 = 60MB
C	528,000,000 = 66MB

Additional parts of the object server are the timing of each individual method call available in each server and a list of complex method calls. All of these measurements were taken on a single machine to normalize the values. In this example, server A has one four methods, server B has two methods, and server C has three methods.

Table 4. Normalized Server Loads.

SERVER	Method	CPU time (s)	Average Size of Message (b)
A	1	0.5796	112000
A	2	2.6203	18400
A	3	1.18175	44800
A	4	2.0264	176000
B	1	1.76655	4000000
B	2	3.70085	2720000
C	1	3.0043	320000
C	2	4.8040	4000000
C	3	0.48815	400000

A complex method call is a method call that calls another object server. These method calls require special handling in measuring their load on the host server and in the objective function for optimizing the system. Table 5 lists the complex method calls in this example.

Table 5: Complex Method Calls.

Complex Method	Exterior Calls
B.2	C.1

The last information needed to optimize the system is information about the applications and the users. This step adds roles to the list of profiles for the system to optimize. These roles have more realistic use patterns for the different jobs a user would actually perform on the system. For this example, we will have three client applications with two buttons, nine buttons and three buttons respectively.

Let's assume that there are three different roles the network of computers supports for the user and the following is the use pattern shown in Table 6, and that the buttons call the following server methods shown in Table 7. Method calls that appear in italics in Tables 7 and 8 are complex method calls. They appear in italics to remind us that these methods require special handling when figuring out the objective function.

Table 6. Roles.

ROLE	CALL PATTERN (observation interval is 990 seconds)
Role 1	50 C1.B1 ÷ 1 C1.B2 ÷ 1 C2.B1 ÷ 1 C2.B6
Role 2	10 C1.B1 ÷ 40 C1.B2 ÷ 24 C3.B2
Role 3	50 C2.B5 ÷ 10 C2.B9 ÷ 30 C2.B3 ÷ 1 C2.B2 ÷ 1 C3.B2

Table 7. User interface calls.

Button	Methods Called
C1.B1	A.1
C1.B2	A.2 + B.1
C2.B1	C.1 + C.2
C2.B2	C.3
C2.B3	C.2
C2.B4	C.3
C2.B5	A.1 + *B.2*
C2.B6	*B.2*
C2.B7	A.4
C2.B8	C.3 + A.3
C2.B9	A.1 + A.2 + A.3 + *B.2*
C3.B1	C.1
C3.B2	B.1 + *B.2*
C3.B3	C.2

By substituting the user interface calls into the roles matrix, we get an objective function for optimizing the system shown in Table 8. All other method calls will be ignored.

Table 8. Roles to server calls.

ROLE	Methods Called in Role
Role 1	50 * (A.1) + 1 * (A.2 + B.1) + 1 * (C.1 + C.2) + 1 * (*B.2*)
Role 2	10 * (A.1) + 40 * (A.2 + B.1) + 24 * (B.1 + *B.2*)
Role 3	50 * (A.1 + *B.2*) + 10 * (A.1 + A.2 + A.3 + *B.2*) + 30 * (C.2) + 1 * (C.3) + 1 * (B.1 + *B.2*)

4.7 Model Solutions

All of the information above is run through a LINGO model that varies the location of the object servers on the different machines to find the a solution set that minimizes the value of the objective function. Changing any of these variables will lead to different model outputs [13]. For this example, the LINGO model computes a solution on a 360MHz PC in less than one second.

LINGO is a comprehensive tool designed to make building and solving linear, nonlinear and integer optimization models faster, easier and more efficient. LINGO provides a completely integrated package that includes a powerful language for expressing optimization models, a full featured environment for building and editing problems, and a set of fast built-in solvers. It is a product of LINDO Systems, Inc. and can be found on the web at www.lindo.com

4.8 Model Outputs

This method outputs the following deployment strategies for the different roles when setting different RAM limits and keeping all other variables the same as in the last example. Solving the optimization problem defined in section 4.1 with the parameter values determined in section 4.6 derives these results.

Table 9. Single user. RAM limit set to 1.5.

Machine	Role 1 (1 user)	Role 2 (1 user)	Role 3 (1 user)
SIX	None	None	None
BR733	None	None	None
GIGA	A, B, C	A, B, C	A, B, C

Table 10. Single user. RAM limit set to 1.0.

Machine	Role 1 (1 user)	Role 2 (1 user)	Role 3 (1 user)
SIX	None	None	None
BR733	B	C	A
GIGA	A, C	A, B	B, C

Table 11. Concurrent users. RAM limit set to 1.0.

Machine	Role 1 (28 user)	Role 2 (4 user)	Role 3 (3 user)
SIX	None	A	A
BR733	B, C	C	B
GIGA	A	B	C

From the model output, we can see that when a single user is present and RAM is not a limiting factor, the result is that all the servers migrate to the fastest machine. However, when we start to limit RAM, the servers start to spread out. The first server to leave the fastest machine turns out to be different in each role. Multiple concurrent users also tend to spread the servers across the available machines. The significance of the model is that different roles and different numbers of concurrent users lead to different optimal configurations in most cases for this example. No single static configuration can outperform the ability to change configurations based on perceived changes in the usage of the system.

4.9 Validation Experiments

We tested the validity of the model by experimental measurement. A test bed was created with Windows 2000 machines that match the characteristics of the machines in the above example. Servers were created using JDK 1.3 and RMI as the middleware. Software to simulate the three different users was also created. The user was simulated with a random choice for button selection that has a uniform distribution similar to the roles. This simulation software was instrumented to measure the actual time the software was blocked waiting for an object server method call to response [13]. All 27 different configurations were established and the average response time for each configuration was measured and recorded. Between each simulation, the test bed machines were rebooted.

All 27 configurations were tested twice. One tested the configuration with the object servers using much less than the stated memory needs. Another tested the configuration with the object servers using all of the stated memory needs. Some configurations strained the machines memory limits. These configurations resulted in system failures in the test with the object servers using all of the stated memory needs. These system failures are listed as error in the tables of results. It should be noted that Windows 2000 did a much better job of swapping when memory utilization exceeded 100% than a previously tested operating system, Windows NT.

4.9.1 Experimentation Results

A tabulation of experimental results obtained from measuring the outputs of a test system for all of the test scenarios can be found in the dissertation [13]. The following sections detail some of the results and observations for different test scenarios.

4.9.2 Role 1

The models chose a configuration of pattern 1 when RAM was set at 150% utilization and a configuration of pattern 3 when RAM was limited to 100% utilization. Pattern 3 was the third fastest average response time in the minimal memory run and the fastest average response time in the stated memory run. The fact that pattern 10 was the fastest average response time in the minimal memory run is a result of the variability of the simulation [13]. Pattern 1 was the fourth fastest on both runs even though it was the predicted configuration when RAM usage was set to 150% of physical RAM in the model. More interesting from a software engineering standpoint was the fact that the model proposed a configuration that outperformed most configurations from 10 to 44 percent and that the recommended patterns were free from failures.

4.9.3 Role 2

The models predicted a configuration of pattern 1 when RAM was set at 150% utilization and a configuration of pattern 2 when RAM was limited to 100% utilization. In the two runs, the models predicted configuration of pattern 2 was the second fastest average response time in both runs. Pattern 1 was the fastest average response in both runs, which is the predicted configuration when RAM usage is 150% of physical RAM. Again, the configuration chosen by the model outperformed most configurations from 10 to 38 percent. When 4 concurrent users were present, the model predicted pattern 26 which was the second fastest in testing.

4.9.4 Role 3

The models predicted a configuration of pattern 1 when RAM was set at 150% utilization and a configuration of pattern 5 when RAM was limited to 100% utilization. In the two runs, the models predicted configuration of pattern 5 was the third fastest average response time in the minimal memory run and the second fastest average response time in the stated memory run. Pattern 1, the fastest average response time in both runs, was the predicted configuration when RAM usage was set to 150% of physical RAM. The fact that pattern 12 was the second fastest time in the minimal memory run is a result of the variability of the simulation [13]. Again, the model proposed configuration outperformed most configurations from 10 to 44 percent. When 3 concurrent users were present, the model predicted pattern 27, which was the fastest in testing.

4.9.5 Manning Shifts

Although the model does a good job of predicting performance for a single point, the true strength of this approach is chaining these points together. By taking advantage of changes to the system at predictable points in time, we can do better than any single statically assigned server placement.

Table 12: Shift Changes.

P	ROLE 1	ROLE 2	ROLE 3	R2 (4)	R3 (3)	R1 (28)
2	899.34	5530.33	8266.52	11746.10	13925.95	4964.73
3	5960.81	6417.17	7802.17	11711.42	13066.21	4333.77
4	1079.64	6686.38	9124.94	14333.22	20415.47	5789.35
5	1140.80	5953.02	7411.34	11335.30	14614.58	7005.97
26	1355.59	6752.50	8625.44	10544.92	13839.30	11117.11
27	1306.69	7380.83	8259.05	12569.52	12488.02	12042.34

If we assume that we have a shift schedule that has the following six unique manning requirements over the duration of the schedule, then we can initiate object server re-deployments to coincide with the shift changes. The shaded areas in Table 12 indicate the deployment pattern recommended by the model. The numbers in the matrix are the actual measured values for these deployments.

We are only interested in the six deployment patterns listed in Table 12. If we were to institute a static deployment for our system, then we would be forced to pick just one of the deployment patterns listed above. The system engineer would be forced into some logic that mitigated a worst-case scenario.

However, since we have the ability to reason about different manning schedules, then we can take advantage of this capability. By allowing the system to adjust the location of its object servers at shift changes, we gain substantial improvements to the system.

By comparing the models recommended deployment pattern versus the other six deployment patterns in Table 12, we can quantify this improvement. By dividing the model predicted patterns measured performance by the measured performance of the other patterns in the same column, we get the performance improvement for each shift. Table 13 below contains these values.

Table 13: Shift improvements.

PAT	ROLE 1	ROLE 2	ROLE 3	R2 (4)	R3 (3)	R1 (28)
2	-7%	0%	10%	10%	10%	24%
3	0%	14%	5%	10%	4%	13%
4	11%	17%	18%	26%	39%	0%
5	16%	7%	0%	7%	15%	46%
26	29%	18%	14%	0%	10%	66%
27	26%	25%	10%	16%	0%	68%

Interesting to note is that we are only comparing deployment patterns that are of high probability of actually being used. Only one entry in the table has a negative value, all other entries have a substantial performance improvement. Clearly from Table 13, any organization with known manning schedules that fluctuate would benefit from this approach.

5. CONCLUSIONS

The approach produces useful results even with a simplistic model that doesn't directly address queuing delay. The system responds in a reasonable way with changes in the environment, constraints placed on the system, and different roles that a user might want. Since all of these changes take place on real networks of computers, static deployment strategies will never utilize the assets available to better support the end user. The strategies chosen by our model were robust in the sense that performance was good even when actual loads departed from predicted loads.

Predicting exactly how a user will interact with a system that supports multiple roles will always be an inexact science because of the limitations inherent in modeling users, software, hardware, etc. This system provides an adaptive software engineering approach to a real world problem that currently does not have a better solution. Adaptive reconfiguration of object servers enables systems to automatically grow to the point where collective machine limits are exceeded and hard failures occur. Perhaps the most significant capability added by our model is the ability to use predictive and planning information encoded as profiles to pre-optimize system configuration in preparation for peak loads, staying ahead of events and avoiding reconfiguration overhead during periods of high load.

6. FUTURE WORK

The system needs to be refined to more precisely reflect the workings of the network of computers in more complex topologies. These refinements include allowances for asymmetric communications, latency, and queuing delays. Aggregated tuples of these models will be necessary to better evaluate the impact of RAM utility on processing speed. The CPU constraint in the model could be replaced by a function that more accurately models the variation of queuing delays with arrival rates.

By tying the logic with a tool that automatically generates object servers [15], performance comparisons can be made between spreading object types across multiple object servers and machines versus replicating the original object server on multiple machines.

The approach could also be used to optimize other kinds of systems involving a mixed bag of server types, as long as those servers can be modeled as object servers. This would enable better deployment strategies, especially since many of these non-object servers could be tightly coupled to object servers.

7. REFERENCES

[1] Adler, R., "Distributed Coordination Models for Client/Server Computing," *IEEE Transactions on Computers*, pp. 14-22, April 1995.

[2] Berzins, V. and Luqi, "Software Engineering with Abstractions", chapter 6, Addison-Wesley, ISBN 0-201-08004-4, 1991.

[3] Foster, I., Kesselman, C., Tuecke, S., "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International. Journal on Supercomputer Applications*, 2001.

[4] Foster, I., Roy, A., Sander, V., Winkler, L., "End-to-End Quality of Service for High-End Applications," *IEEE Journal on Selected Areas in Communications Special Issue on QoS in the Internet*, 1999.

[5] Hsiao, C., King, C., "The Thread-Based Protocol Engines for CC-NUMA Multiprocessors," *International Conference on Parallel Processing 2000 Proceedings*, pp. 497-504.

[6] Kim, J., Lee, H. and Lee, S., "Replicated Process Allocation for Load Distribution in Fault-Tolerant Multicomputers," *IEEE Transactions on Computers*, vol. 46, no. 4, pp. 499-505, April 1997.

[7] Loh, P., Hsu, W., Wentong, C. and Sriskanthan, N., "How Network Topology Affects Dynamic Load Balancing," *IEEE Transactions on Parallel and Distributed Technology*, vol. 4, no. 3, pp. 25-35, Fall 1996.

[8] Luckham, D. and Frasca, B., "Complex Event Processing in Distributed Systems," *Computer Systems Laboratory Technical Report CSL-TR-98-754.* Stanford University, Stanford, 1998.

[9] Luckham, D. and Vera, J., "An Event-Based Architecture Definition Language," *IEEE Transactions on Software Engineering*, Vol 21, No 9, pp.717-734. Sep. 1995.

[10] Lui, J., Muntz, R. and Towsley, D., "Bounding the Mean Response Time of the Minimum Expected Delay Routing Policy: An Algorithmic Approach," *IEEE Transactions on Computers.* Vol 44, No. 12, December 1995, pp. 1371-1382.

[11] Mehra, P. and Wah, B., "Synthetic Workload Generation for Load-Balancing Experiments," *IEEE Transactions on Parallel and Distributed Technology*, vol. 3, no. 3, pp. 4-19, Fall 1995.

[12] Perrochon, L., Mann, W., Kasriel, S. and Luckham, D., "Event Mining with Event Processing Networks," *The Third Pacific-Asia Conference on Knowledge Discovery and Data Mining.* April 26-28, 1999. Beijing, China, 5 pages.

[13] Ray, W., "Optimization of Distributed, Object-Oriented Systems," *PhD Dissertation in Software Engineering*, Naval Postgraduate School, September 2001.

[14] Ray, W., Berzins, V. and Luqi, "Adaptive Distributed Object Architectures," *AFCEA Federal Database Colloquium 2000 Proceedings*, pp. 313-330, September 2000.

[15] Ray, W. and Farrar, A., "Object Model Driven Code Generation for the Enterprise," *IEEE RSP 2001*, June 2001.

A Unified Approach to Component Assembly Based on Generative Programming*

Wei Zhao[1] Barrett R. Bryant[1] Rajeev R. Raje[2] Mikhail Auguston[3] Andrew M. Olson[2] Carol C. Burt[1]

The UniFrame project consists of a unified meta-component model (UMM) for distributed component-based systems (DCS), and a Unified Approach (UA) for integrating components [7]. The core parts of the UMM are: *components, service and service guarantees*, and *infrastructure*. A creation of a software solution for a DCS using UA comprises of two levels: a) the component level -- developers create components, test and validate the appropriate functional and non-functional (Quality of Service -- QoS features and deploy the components on the network, and b) the system level -- a collection of components, each with a specific functionality and QoS, are obtained from the network, and a semi-automatic generation of a software solution for a particular DCS is achieved.

It is assumed that different developers will provide on a network a variety of possibly heterogeneous components for specific problem domains. For a specific problem, a search process will identify relevant components from those available on the network. Once these are identified, the task is to integrate these disparate components in a specific solution for the DCS under construction. The UA assumes that the generation environment is built around a generative domain-specific model (GDM) [4, 5] supporting component-based system assembly. The distinctive features of UA are:

- The developer of a distributed application presents to the UA-based system a query, describing the required characteristics, in a structured form of a natural language. The query is processed using the domain knowledge (such as key concepts from a domain) and a knowledge-base containing the UMM descriptions of the components for that domain. The domain knowledge and the knowledge-base are parts of the GDM. From this query a set of search parameters is generated which guides "head-hunters" to perform a component search of the networked environment.. Headhunters are special components responsible for locating components deployed by the developers for the specific domain under consideration [8].
- The headhunters discover a set of applicable components that satisfy the functional and QoS requirements as indicated by the developer of the distributed system. The developer expresses the QoS requirements by selecting an appropriate set of parameters from a catalog of parameters [2]. After the components are fetched, the distributed application is assembled according to the generation rules embedded in the GDM. This assembly requires the creation of glue/wrapper interface between various components. Two-Level Grammar (TLG) [3] is used to specify the generative rules and provides the formal framework for the generation of the glue/wrapper code. This is implemented according to the process of translating TLG specifications into executable code as described in [6].
- QoS parameters are divided into two categories: a) static and b) dynamic. Static QoS parameters (e.g. dependability) are processed during the generation. Dynamic QoS parameters (e.g., response time) result in the instrumentation of generated target code based on event grammars [1], which at run time produce the corresponding QoS dynamic metrics, to be measured and validated.

* This material is based upon work supported by, or in part by, the U. S. Army Research Laboratory and the U. S. Army Research Office under contract/grant numbers DAAD19-00-1-0350 and 40473-MA, and by the U. S. Office of Naval Research under award number N00014-01-1-0746.

[1] Department of Computer and Information Sciences, University of Alabama at Birmingham, Birmingham, AL 35294-1170, U. S. A., *{zhaow, bryant, cburt}@cis.uab.edu.*

[2] Department of Computer and Information Science, Indiana University Purdue University Indianapolis, Indianapolis, IN 46202, U. S. A., *{rraje, aolson}@cs.iupui.edu.*

[3] Computer Science Department, New Mexico State University, Las Cruces, NM 88003, U. S. A., *mikau@cs.nmsu.edu.*

QoS parameters given in the query provide a special dimension to the generated code - the instrumentation necessary for the run-time QoS metrics evaluation. Based on the query, the user has to come up with a representative set of test cases. Next the implementation is tested using the set of test cases to verify that it meets the desired QoS criteria. If it does not, it is discarded. After that, another implementation is chosen from the component collection. This process is repeated until an optimal (with respect to the QoS) implementation is found, or until the collection is exhausted. In the latter case, the process may request additional components or it may attempt to refine the query by adding more information about the desired solution from the problem domain. If a satisfactory implementation is found, it is ready for deployment.

The case study for the generative approach is a bank account management system. It is assumed that different client and server components for a bank domain are available on the network. These components (belonging to a category, i.e., server/client) do offer the same functionality but different QoS features. After requesting the construction of this system, assume that the headhunters found the following components: **JavaAccountClient**, **JavaAccountServer**, and **CorbaAccountServer**. The first two adhere to the Java-RMI model and the third one is developed with CORBA technology. The UMM specifications associated with the components indicate that the two server components have the same functionality but **CorbaAccountServer** has better service guarantees and meets the QoS specified in the system query, thus the final system should assembled from the Java client and the CORBA server. Based on the generation rules embedded in GDM, a proxy server for the Java client component, a proxy client for the CORBA server component, a bridge driver between the two proxies and some other installation helper files can be generated to form an integrated account system. The assembled system will be deployed if it meets the desired QoS criteria. If the system succeeds then a new set of UMM specifications will be generated for the integrated system to insure that it is available for the discovery by other head-hunters, i.e., to act as a component of other possible application systems.

References:

[1] M. Auguston, A. Gates, M. Lujan, *Defining a Program Behavior Model for Dynamic Analyzers,* Proc. SEKE '97, 9th Int. Conf. Software Eng. Knowledge Eng., 1997, pp. 257-262.

[2] G. J. Brahnmath, R. R. Raje, A. M. Olson, M. Auguston, B. R. Bryant, C. C. Burt, *A Quality of Service Catolog for Software Components.* Proc. Southeastern Software Eng. Conf. (to appear), 2002.

[3] B. R. Bryant, B.-S. Lee, *Two–Level Grammar as an Object-Oriented Requirements Specification Language.* Proc. 35th Hawaii Int. Conf. System Sciences, 2002.

[4] J. C. Cleaveland, *Program Generators with XML and Java*, Prentice-Hall, 2001.

[5] K. Czarnecki, U. W. Eisenecker, *Generative Programming: Methods, Tools, and Applications.* Addison-Wesley, 2000.

[6] B.-S. Lee, B. R. Bryant, *Automated Conversion from Requirements Documentation to an Object-Oriented Formal Specification Language.* Proc. ACM Symp. Applied Computing (to appear), 2002.

[7] R. R. Raje, M. Auguston, B. R. Bryant, A. M. Olson, C. C. Burt, *A Unified Approach for the Integration of Distributed Heterogeneous Software Components.* Proc. Monterey Workshop Engineering Automation for Software Intensive System Integration, 2001, pp.109-119.

[8] N. N. Siram, R. R. Raje, B. R. Bryant, A. M. Olson, M. Auguston, C. C. Burt, *An Architecture for the UniFrame Resource Discovery Service.* Submitted for publication, 2002.

Using an Object Oriented Model for Resolving Representational Differences between Heterogeneous Systems

Paul Young, Valdis Berzins, Jun Ge, Luqi

Department of Computer Science
Naval Postgraduate School
Monterey, California 93943, USA

Email: {peyoung, berzins, jge, luqi}@nps.navy.mil

ABSTRACT

One of the major concerns in the study of software interoperability is the inconsistent representation of the same real world entity in various legacy software products. This paper proposes an object-oriented model to provide the architecture to consolidate two legacy schemas in order that corresponding systems may share attributes and methods through use of an automated translator. A Federation Interoperability Object Model (FIOM) is built to capture the information and operations shared between different systems. An automatic wrapper-based translator is discussed that utilizes the model to bridge data representation and operation implementation differences between heterogeneous distributed systems.

Keywords

interoperability, object-oriented, heterogeneous, wrapper-based.

1. INTRODUCTION

In contemporary object-oriented modeling, an object is a software representation of some real-world entity in the problem domain. An object has identity (i.e., it can be distinguished from other objects by a unique identifier of some kind), state (data associated with it), and behavior (things you can do to the object or that it can do to other objects). In the Unified Modeling Language (UML) these characteristics are captured in the name, attributes, and operations of the object, respectively. UML distinguishes an individual object from a set of objects that share the same attributes, operations, relationships, and semantics; termed a *class* in UML [1].

This view of objects and classes has proven valuable in the development of countless systems in various problem domains encompassing all degrees of size and complexity. However, one common characteristic of the majority of these object-oriented developments is that they were produced by a development team that shared common objectives and had a common view of the real-world entities being modeled. Most projects also involve a common architecture implemented on a common target platform, using the same implementation language and operating system. As a result a single scheme for depicting an entity's name, attributes, and operations as well as the means for representing these properties has been the norm. Therefore, capturing the representation of these properties has not been an issue. The software representation of the real-world entity should have the same name, attributes, and operations across all elements of the architecture if the development team enforces consistency.

This is not necessarily the case when integrating independently developed systems. The different perspectives of the real-world entity being modeled by independent development teams will most likely result in the use of different class names as well as differences in the number, definition, and representation of attributes and operations for that same real-world entity. These representation differences must be reconciled if the systems are to interoperate.

This paper proposes an object-oriented model for defining the information and operations shared between systems. The initial use of the model is targeted for integration of legacy systems, which generally have not been developed using the object-oriented paradigm. However, defining the interoperation between systems in terms of an object model provides benefits in terms of increasing the visibility of the information and operations shared between systems, and provides a foundation for easy extension as new systems are added to an existing federation. The object model defined in this paper can be easily constructed from the external interfaces defined for most legacy systems (whether object-oriented or not).

Section 2 categorizes representational differences that exist in autonomously developed systems. Section 3 introduces the Object-Oriented Model for Interoperability (OOMI) as a means for capturing the information required for resolving these representational differences. Section 4 introduces an automated environment for constructing an instance of the interoperability object model for a federation of systems, the OOMI Integrated Development Environment (OOMI IDE). Section 5 presents an overview of the use of this Federation Interoperability Object Model (FIOM) by a wrapper-based translator for enabling interoperability among legacy systems.

SAC 2002, Madrid, Spain
ISBN 1-58113-445-2/02/03

2. CATEGORIZING REPRESENTATIONAL DIFFERENCES

This paper addresses two categories of variations in the representation of a real-world entity on different systems. The first category concerns differences in the information utilized by each component system to represent the entity. Termed *heterogeneity of scope*, this refers to the fact that differing amounts and types of information can be captured by various systems to represent the state and behavior of an entity [10].

For example, suppose a federation of four autonomously developed military systems contained information about an enemy surface-to-surface missile launcher. Because independent development teams created them, each system provides a different perspective on what state and behavior information should be contained in a model of that real-world entity. As can be seen from Figure 1, each system includes different aspects of the entity's state. For instance, systems A and D include information about the missile system's type, position, and time. System B captures position, time and range information on the entity, and System C utilizes type, position, time, and range to describe the missile system. Similarly, each system could capture different aspects of the behavior of an entity. These differences in the state and behavior used by a component system to characterize a real-world entity can be thought of as providing different *views* of the entity by the systems concerned.

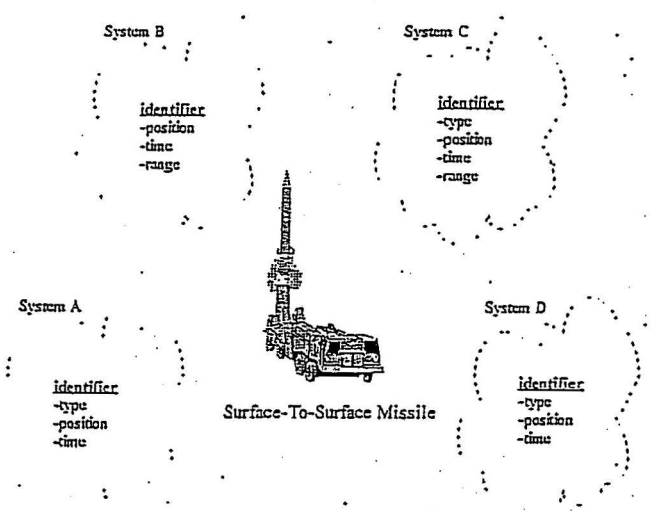

Figure 1. Differing Views of Real-World Entity

Even if two systems provide the same view of the entity being modeled, that is they both contain the same state and behavior information about the entity, there may still be differences in the representation of that information on different systems. This *heterogeneity of representation* [10] refers to differences in the terminology used, format, accuracy, range of values allowed, and structural representation of the included state and behavioral information [5]. This difference in representation is illustrated in Figure 2 by systems A and D. Even though these systems both have the same view of our real-world entity, i.e. both capture the type, position and time for the entity; they each represent the information comprising that view in a different manner. For example, System A refers to our entity as a *SurfaceToSurfaceMissile* and names its type attribute *missileDesignation*. System D refers to our entity as an *SSM* and names its type attribute *missileType*. Additionally, System A captures the entity *position* in latitude/longitude coordinates and *time* using Greenwich Mean Time (GMT) as the reference, whereas System D records entity *location* using Military Grid Reference System (MGRS) coordinates and records *time* using Local Mean Time (LMT). Figure 2 illustrates the different views of our example real-world entity and the various representations provided for each view.

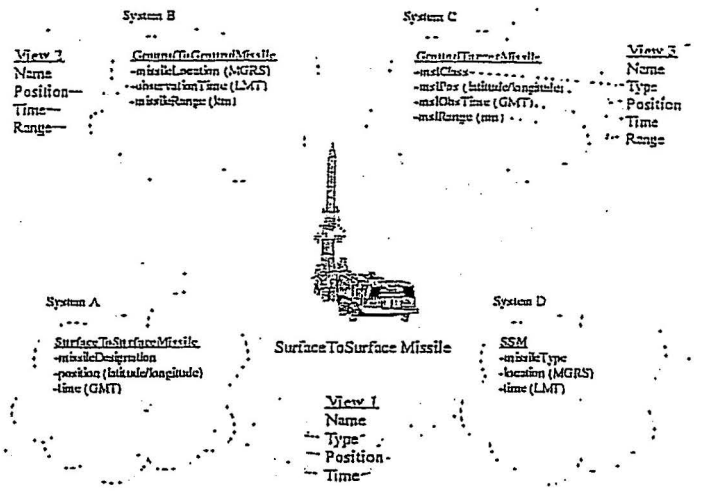

Figure 2. Differing Real-World Entity View Representations

3. OBJECT-ORIENTED MODEL FOR INTEROPERABILITY

The goal of the research presented in this paper is to provide a computer-aided methodology to aid in the resolution of differences in the representation of data between systems targeted for integration in order to enable system interoperability. Pitoura defines interoperability as the capability of systems to exchange information and to jointly execute tasks [8]. The information exchanged between interoperating systems consists of data associated with the real-world entities being modeled by systems of the federation. The joint execution of tasks reflects the capability of an entity on one system to employ the services of an entity on another. Thus, interoperation can be characterized in terms of the real-world entities whose state and behavior are shared between systems in a federation. As stated previously, there can be differences in view and representation of these real-world entities. In order to achieve interoperability, a means for bridging these differences in view and representation is needed.

As the basis for achieving interoperability between systems in a federation, a model was defined for depicting the real-world entities that represent the shared state and behavior [11]. The model captures differences in view and representation of these entities and provides the means for bridging such differences. Principal objectives of the model were to clearly depict the real-world entities whose state and behavior are shared between systems in a federation, to provide computer aid to the process of determining the differences in view and representation of those entities, to provide automation support for defining the translations necessary to resolve representational differences between systems, and to capture the information required to resolve differences in real-world entity scope and representation between federation systems.

In evaluating the objectives outlined above, it was determined that an object-oriented approach offered the greatest promise for satisfying these requirements. Object-oriented analysis and design (OOAD) provides principles of abstraction, information hiding, and inheritance that can be employed to meet the specified goals and objectives [4, 9]. However, conventional use of these object-oriented principles and techniques is not sufficient for resolving representational differences between heterogeneous components of a system federation. Instead, a model-based approach built on OOAD principles is presented to satisfy the requirements for heterogeneous system interoperability. The resulting model, the *Object Oriented Model for Interoperability (OOMI)*, is described below.

3.1 Capturing Real-World Entities and Views

The real-world entities whose state and behavior information are shared among a federation of interoperating systems are modeled in the OOMI using the concept of a *Federation Entity (FE)*. The FE provides an abstract representation of the information being shared while hiding the details of how that information is being represented on different systems. From the example introduced in Section 2, an FE for the *SurfaceToSurfaceMissile* depicted in Figures 1 and 2 is created to represent the real-world entity, as shown in Figure 3.

For each FE, one or more *Federation Entity Views (FEVs)* are used to distinguish the differences in the state and behavior information used for representing the same real-world entity on different systems. Continuing the example, FEVs are created for the three views of the surface-to-surface missile entity illustrated in Figure 2, labeled *SSM*, *GroundToGroundMissile*, and *GroundTargetMissile* for views 1 through 3 in Figure 3.

It is expected that for a federation of heterogeneous systems, a number of real-world entities will be involved in the interoperation between systems. Under the OOMI, the collection of real-world entities used to define the interoperation of a specified federation of systems is termed a *Federation Interoperability Object Model (FIOM)*. Figure 4 provides a representative FIOM containing the SurfaceToSurfaceMissile FE previously introduced as well as other FEs involved in the interoperation of a hypothetical federation. The notation used in Figure 4 is similar to UML, with each FE represented as a UML package containing a list of the different views of that FE, with the details of the SurfaceToSurfaceMissile FE views described in Figures 5 and 7.

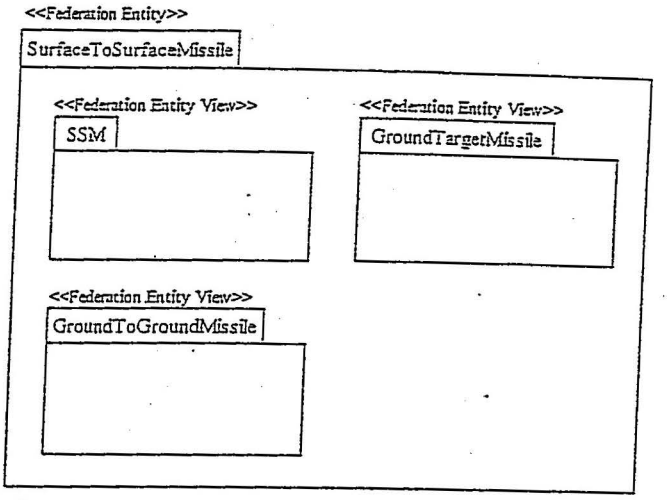

Figure 3. Defining Federation Entity (FE) and Federation Entity Views (FEVs) for Modeled Real-World Entity

All of the normal relationships between classes, packages, interfaces, and other elements used in the OOAD paradigm are available for use with federation entities in the FIOM. For example, in Figure 4 the previously introduced *SurfaceToSurfaceMissile* FE represents a specialization of a *GroundLaunchedWeapon* FE, which in turn has a *part of* relation with an EnemyOrderOfBattle FE. This enables the OOMI to exploit OOAD principles such as inheritance in modeling the entities that define the interoperation between systems.

3.2 Capturing FEV Representations

In addition to providing dissimilar views of a real-world entity defining the interoperation between components, different systems may also provide varied implementations of a view. As discussed earlier, these different implementations may result in

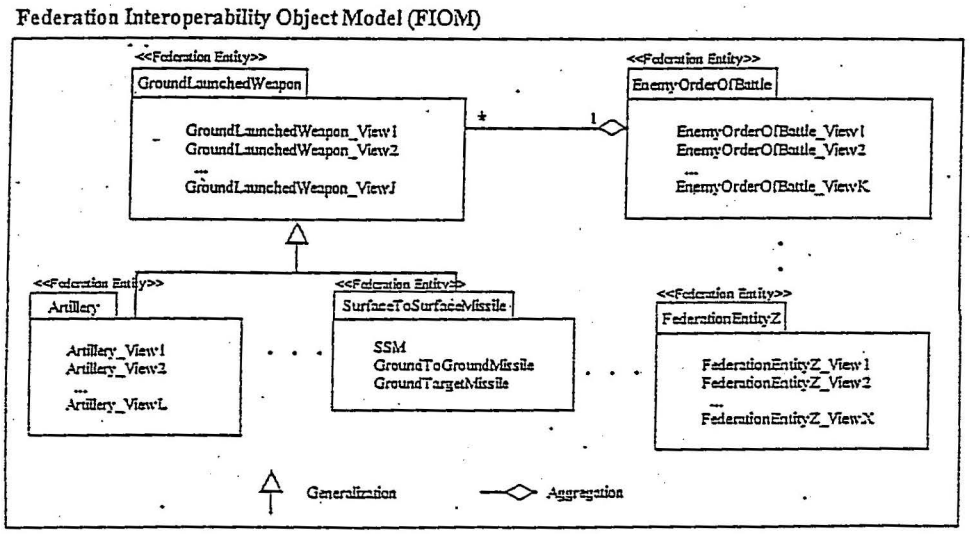

Figure 4. Federation Interoperability Object Model (FIOM) Representation

variations in the terminology, definition, and representation of the attributes and operations defined for the same real-world entity. In order to resolve these differences, the OOMI provides two mechanisms to capture the possible alternative representations of an entity's view. The first mechanism, the *Component Class Representation (CCR)* is a special-purpose class used to capture the alternative ways various component systems may represent a federation entity view.

The typical approach to resolving representational differences between systems involves the use of a number of point-to-point translators between systems to be integrated. For a federation of n systems, this approach requires the specification of n(n-1)/2 translations. An alternative to the use of point-to-point translators involves the use of an intermediate representation where the information and operations being transmitted are converted from the source representation to an intermediate representation and then to the destination representation. The use of an intermediate representation requires specification of 2n translations for a federation of n systems.

In order to take advantage of the reduced number of translators required with the use of the intermediate representation approach, the OOMI adds a second special-purpose class to an FEV, the *Federation Class Representation (FCR)*. The FCR is used to reflect the "standard" (as defined by the interoperability engineer for the specified federation) representation used by the federation for an entity's view. Each FEV will contain exactly one FCR representing this "standard" representation of the view. The FCR serves as the intermediate representation for translation between a source and destination system. Figure 5 illustrates the CCRs and FCRs created for the system A through D representation of the example surface-to-surface missile previously introduced in Figure 2. Note that each FEV contains a single FCR whereas an FEV may contain more than one CCR- the *SSM* FEV includes CCRs *SurfaceToSurfaceMissile* and *SSM* corresponding to System A's and System D's representation of the view, respectively.

The FCR representation is based on an *ontology* containing the federation-sanctioned depiction of an entity's state and behavior. This ontology can be developed specifically for a federation of systems or it can be derived from a domain-specific or industry-wide standard such as the Defense Information Systems Agency's (DISA's) Defense Information Infrastructure (DII) Common Operating Environment (COE) XML Registry or the Defense Modeling and Simulation Office's (DMSO's) Functional Description of the Mission Space (FDMS) namespaces [2, 3].

The FCR and CCR are each actually a composition of related special-purpose classes. These component classes contain information needed to assist the interoperability engineer in identifying the real-world entities that represent the information being shared between systems in the federation as well as define the views and view representations of those entities.

3.2.1 Capturing Information Shared Between Component Systems

The first of these component classes, the *FCR Schema*, is used to characterize the "standard" representation of an entity's view. In general, a schema is a summarized or diagrammatic representation of something. In the OOMI the FCR schema contains the name, attributes, and operations used to represent the "standard" interpretation of an entity's view. The FCR Schema is used to provide an abstract representation of the information being shared between component systems, hiding the details of how that information is represented on different systems.

Similarly, the *CCR Schema* is used to characterize the component system implementation of a federation entity view. The CCR Schema contains the name, attributes, an operations used by a specific component system to model a federation entity.

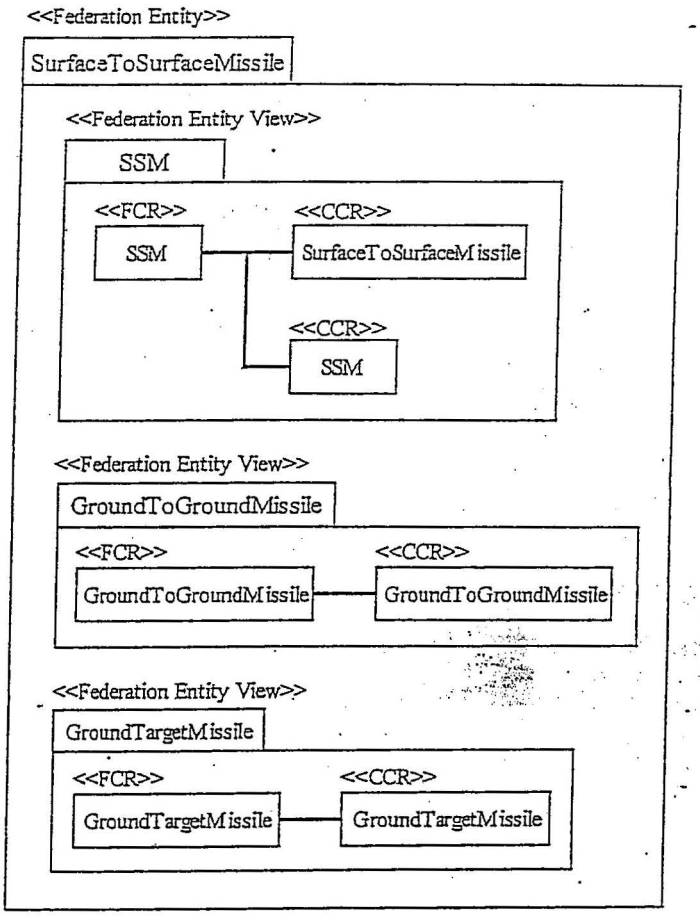

Figure 5. FEV With Component Class Representation (CCR) and Federation Class Representation (FCR)

3.2.2 Identifying Correspondences between Representations

Previous efforts toward integrating heterogeneous databases found that a large part of the effort was consumed by determining whether two entries in related databases represented the same real-world entity [6]. An equivalent situation exists in the integration of heterogeneous system components. When presented with a number of systems to be integrated, the interoperability engineer must determine which classes used to realize the external interfaces of component systems refer to the same entity in the problem environment. Establishing this correspondence is crucial in order for systems to exchange information and operations and is the basis for defining the federation entities involved in systems interoperation. Once determined, this correspondence is captured in the model as an association relating a FEV's FCR and CCR, as depicted in Figure 5.

In order to assist the interoperability engineer in establishing the correspondence between different representations of a federation entity, the FCR and CCR also contain syntactic and semantic information used to correlate the "standard" and various component system representations of the real-world entities defining the interoperation. This information is represented using the special-purpose classes *FCR Syntax* and *CCR Syntax* to capture syntactic information on the "standard" representation and component implementations of a federation entity view, respectively. Similarly, special-purpose classes *FCR Semantics* and *CCR Semantics* capture semantic information about the "standard" and component representations, respectively. This syntactic and semantic information is used to determine the correspondence between component system and interoperability classes in order to construct the entities, views and representations of the FIOM.

Syntactic information is used to capture the composition and structure of a class. Class composition is provided as a list of terms depicting the name, attributes and operations contained in the class. Structural information describes which attributes are included as parameters to which operations, whether attributes and operations are visible outside the class, etc. The composition and structure defines a *signature* for the class that can be used for comparison with other classes. Semantics are used to provide information as to the meaning and behavior of a class, i.e., what does the state information about a class represent and what actions does the class perform? Behavioral information can be captured in terms of a set of conditions an element must satisfy or a set of equations describing the dynamic behavior of the entity.

3.2.3 Capturing the Translations used to Resolve Representational Differences

Finally, the FEV contains the translations required to convert between each component system representation and the "standard" representation of that view. These translations are used to resolve differences in physical representation, accuracy tolerances, range of values allowed, and terminology used in representing a federation entity view. Two translations are defined for each FEV- one to convert an instance of a CCR Schema to an FCR Schema compliant instance, and the other to convert from an FCR to CCR Schema instance. These translations are defined by the interoperability engineer and stored in the FEV as a *CCR-FCR Translation Class* for subsequent use.

3.2.4 Federation Entity View Summary

Figure 6 provides a summary of the contents of an FEV, illustrating *View1* of a hypothetical *FederationEntityA* with the federation representation of that view, *FederationEntityA_View1_FCR*, and a corresponding component system representation, *SystemA_Class1_CCR*. Also depicted are the schema, syntax, and semantics classes that comprise the FCR and CCR, as well as the translation class used to resolve differences between the component and federation schema.

3.3 Resolving Differences in View of an FE

The translations depicted in Figure 6 and described in section 3.2.3 enable the conversion between two different representations of a federation entity view. Rarely will two different systems' view of a federation entity be identical. In order to share information and jointly execute tasks between two systems that have different views of the entity(s) defining the interoperation, these differences in view must be resolved. Fortunately it is just as rare that different systems' views of an entity are mutually exclusive (otherwise they wouldn't be able to interoperate).

Generally, two or more systems' view of the same entity will have some areas of commonality. Two systems' representations may capture the same core state and behavior information of an entity with each including additional characteristics as required by the specific application. In this situation a view could be defined for

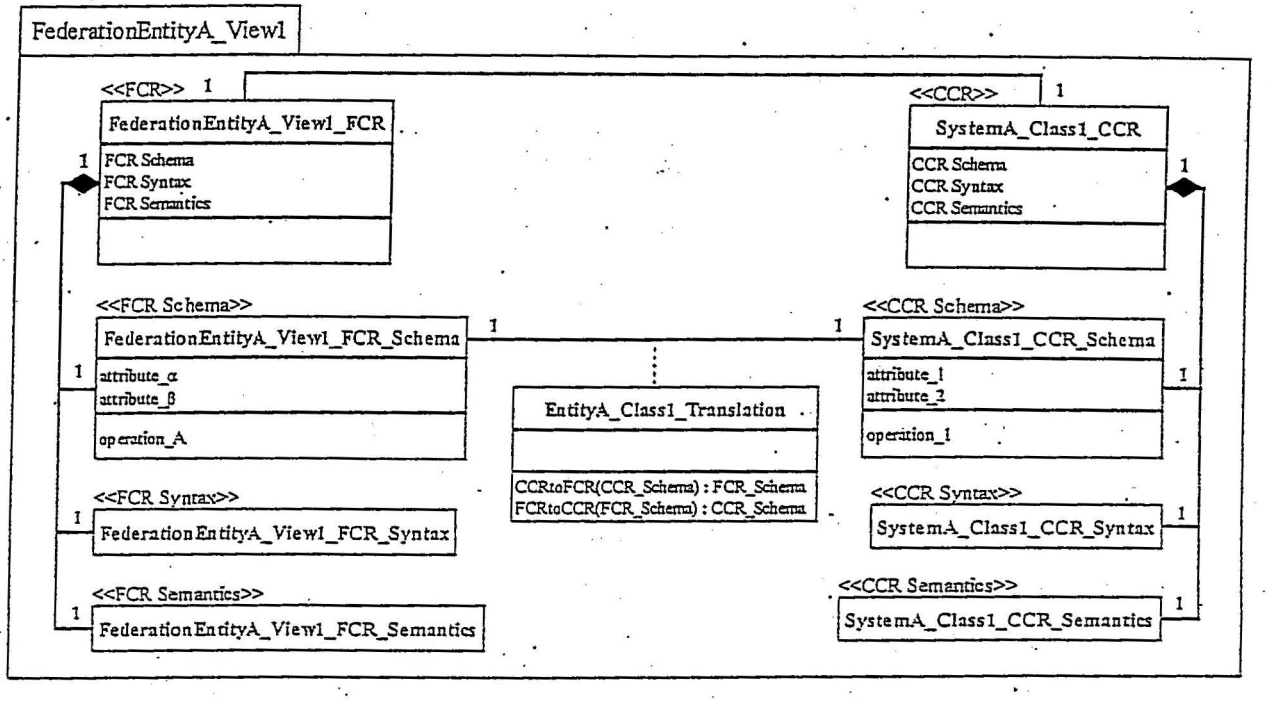

Figure 6. Federation Entity View Archetype

the core state and behavior information, and separate views defined for the extended information. The views containing the extended information can be considered to be subtypes of the view containing the common core information.

By determining the supertype-subtype relationships between entity views, we can construct an inheritance hierarchy that can be used to determine when the information contained in one system's view of an entity is suitable for use by another. This hierarchy is initially constructed by evaluating the attributes and operations contained in the FCR Schema for two views. Figure 7 shows the FEV inheritance hierarchy constructed for the example surface-to-surface missile entity. Due to space considerations, only the FCR and component FCR Schema are shown for each FEV. Details of inheritance hierarchy construction are contained in [11] and are beyond the scope of this paper.

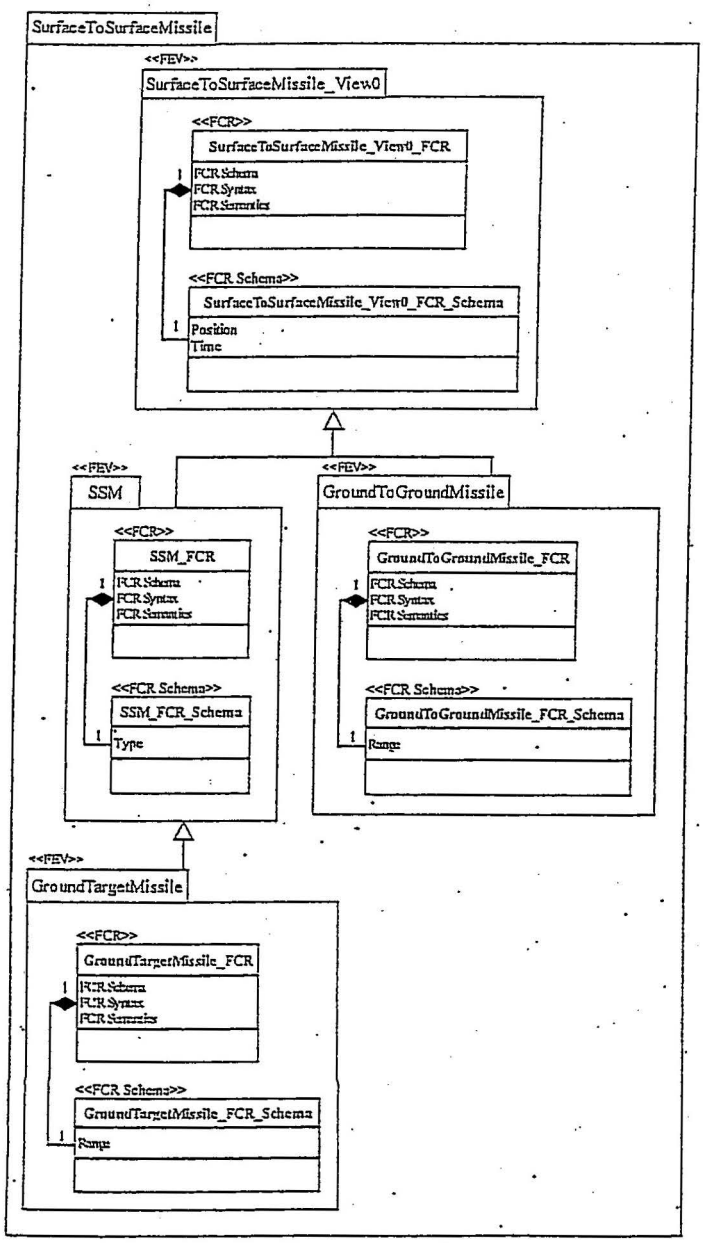

Figure 7. Federation Entity View Inheritance Hierarchy

Then, through exploitation of the Liskov and Wing notion of behavioral subtyping [7], we can determine when the information contained in one system's view of an entity is suitable for use by another. Making this determination is easy when the producer's view of an entity is a subtype of a consumer's view, i.e. when the producer's view extends the consumer's view. By Liskov and Wing's behavioral notion of subtyping, anywhere a supertype can be used a subtype can be substituted without any difference in behavior. Thus, in this instance the consumer will just ignore any additional information provided by the producer.

This determination is not as easy when the producer's view is a supertype of the consumer's view, or when the producer's view is not a direct ancestor or descendent of the consumer's view in the inheritance hierarchy. However, it is possible that the supertype of an entity's view can be substituted for a subtype of the view if the attributes and operations which extend the supertype are either optional for the component system providing a representation of the subtype view, or if default values can be specified for those attributes and operations. Similarly, information can also be shared between component systems that are not direct ancestors or descendents of each other if there is a path in the inheritance hierarchy defined between the producer view and the consumer view, and the previously mentioned restrictions on supertype extension hold.

4. CONSTRUCTING INTEROPERABILITY OBJECT MODEL FOR FEDERATION OF HETEROGENEOUS SYSTEMS

Enabling a collection of related software systems to share information and task execution has the potential for significantly enhancing the capability of the resultant federation of systems over that of the individual components. The previously introduced Object Oriented Model for Interoperability is used to enable information sharing and cooperative task execution among a federation of autonomously developed heterogeneous systems. Using the information contained in the OOMI, computer aid can be applied to the resolution of data representational differences between heterogeneous systems. In order to apply computer aid, a model of the real-world entities involved in the interoperation, termed a Federation Interoperability Object Model (FIOM), is constructed for the specified system federation. Construction of the FIOM is done prior to run-time by an interoperability engineer with the assistance of a specialized toolset, called the Object Oriented Model for Interoperability Integrated Development Environment (OOMI IDE).

The Graphical User Interface (GUI) based OOMI IDE is used to:
1) discover the information and operations shared between federation components,
2) provide assistance in identifying the different representations used for such information and operations by component systems,
3) define the transformations required to translate between different representations, and
4) generate system-specific information used to resolve representational differences between component systems.

The first task in FIOM construction is determining the real-world entities whose state and behavior are shared between systems in the federation. Each resultant federation entity is represented in the FIOM as a package constructed from the classes contained in the component systems' external interface.

Determination of the real-world entities that define the interoperation of a federation is not merely a matter of identifying the classes contained in the external interfaces of the included systems. Because of the independently developed, heterogeneous nature of the systems in the federation, each system may have a different representation for the real-world entities involved. Identifying which of a component system's classes are representations of the same real-world entity is a key step in achieving interoperability between the component systems. Correlation software is included as part of the OOMI IDE in order to assist the interoperability engineer in this effort by providing a small set of proposed correspondences to be reviewed by domain experts.

After identifying the different means used by component systems in the federation to represent the same real-world entity, the transformations required to translate between different representations must be defined. The OOMI IDE assists the interoperability engineer in this task through the use of a GUI-based matching process used to provide computer aid to translation development, and the maintenance of a translation library to enable the reuse of common translation algorithms.

Finally, class transformation and relationship information is extracted from the FIOM for each component system. The system-specific information is used by a wrapper-based translator to resolve representational differences between component systems.

5. USING FIOM TO RESOLVE REPRESENTATIONAL DIFFERENCES BETWEEN HETEROGENEOUS SYSTEMS

As previously mentioned, system interoperability involves both the capability to exchange information between systems and the ability for joint task execution among different systems [8]. Both capabilities involve one or more of the following kinds of actions:

- Send One system transmits a piece of information to another
- Call One system invokes an operation on another
- Return Returns a value to the caller
- Create Creates an object on the called system
- Destroy Destroys an object on the called system [1]

Information exchange is accomplished through means of a *Send* operation, where one system, the producer, exports information that another system, the consumer, imports. Information transmitted by the producer system can be an object of some class defined for the producer, or it can consist of one or more attributes of an object defined for the producer.

Joint task execution is accomplished through the use of a *Call* operation where one system, a client, invokes an operation on another, which acts as a server for the requested action. In invoking an operation on a server, a client system must provide the name of the operation requested as well as any parameters required by the server to perform the operation. Required parameters can be in the form of one or more attributes, operations, or objects. In addition, in response to a client *Call* operation, a server may return a set of attributes, operations, or objects to a client via a *Return* operation. *Create* and *Destroy* actions are special instances of a system call. Care must be exercised in their use due to the security risks they pose- the potential for denial of service attacks and the spread of misinformation through the use of the *Create* operation and the possible loss of vital information through unintended use of the *Destroy* operation.

When information exchange or joint task execution takes place between heterogeneous systems, the interoperability object model constructed during the *pre-runtime* phase for a specified federation of component systems is used to derive a translator. Differences in view and representation of information and tasks shared between interoperating systems are reconciled at *runtime* by the *translator*, which serves as an intermediary between component systems. The translation function is implemented as part of a *software wrapper* enveloping a producer or consumer system (or both) in a message-based architecture, or alternatively as part of the data store (actual or virtual) in a publish/subscribe architecture. A software wrapper is a piece of code used to alter the view provided by a component's external interface without modifying the underlying component code. Figure 8 shows an overview of the use of software wrappers and the involvement of the Federation Interoperability Object Model in the translation process.

The translations required by the wrapper-resident translator for both information exchange and joint task execution are similar. For information exchange, the source system provides the exported information in the form of a set of attributes or objects of a producer class in the native format of the producer. In order to be utilized by a consumer system, the exported information must be converted into the representation expected by the destination system. For joint task execution, a client system provides an operation name and a set of parameter values to a server system in the native format of the producer. The parameters may be attributes, operations, or objects of a client class. Again, this information must be provided to the destination system in a format recognized by that system. Thus the operation name and parameter values must be converted to the server representation.

As indicated above, the translator must be capable of converting instances of a class's attributes and operations (or both attributes and operations in the form of an object of the class) from one representation to another. The information required to effect these translations is captured as part of the FIOM during federation design. Then, at run-time, the translator accesses the information contained in the model to resolve differences in federation entity view and to effect the translation between component and standard representations of a view.

The translator utilizes the FE inheritance hierarchy described in section 3.3 to first resolve differences in the number and type of attributes and operations used to model an entity between two systems in a federation. Then for two systems having the same view of the attributes and operations used to model an entity, the translator resolves differences in representation using the translation operations included in the model with each federation entity view. See [11] for more details.

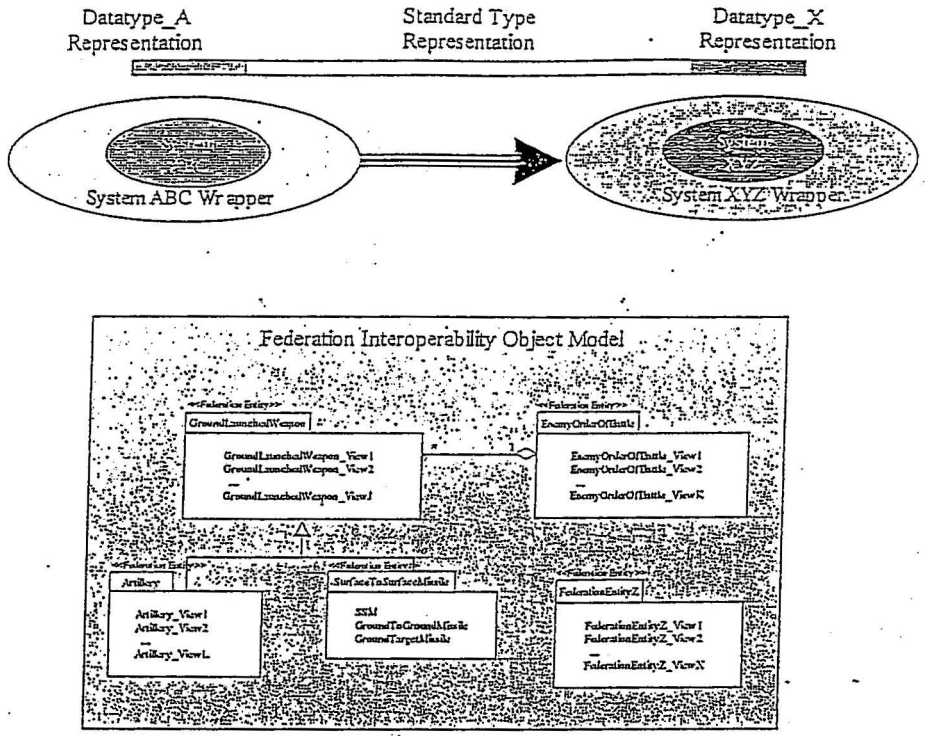

Figure 8. Translator - FIOM Interaction

6. CONCLUSIONS

An Object-Oriented Model for Interoperability (OOMI) is proposed in this paper to solve the data and operation inconsistency problem in legacy systems. A Federation Interoperability Object Model (FIOM) is defined for a specific federation of systems designated for interoperation. A specialized toolset, the Object Oriented Model for Interoperability Integrated Development Environment (OOMI IDE) is used prior to runtime to construct the FIOM for the federation. The FIOM consists of a number of Federation Entities (FEs) that contain the data and operations to be shared between systems. The FIOM also captures the translations required to bridge differences in representation of this data and operations. Then, at runtime, a wrapper-based translator utilizes the information contained in the FIOM to automatically convert instances of real-world entity attributes and operations to the proper representation to enable interoperation between systems.

At this stage, XML-based message translation is being studied for implementation of the proposed model. The capability provided by the XML family of tools coincides nicely with the requirement for data and operation representation capture and translation.

7. REFERENCES

[1] Booch,G., Rumbaugh, J., Jacobson, I., *The Unified Modeling Language User Guide*, Addison-Wesley Longman, Inc., Reading, MA, 1998.

[2] "DII COE Data Emporium." [http:// diides.ncr.disa.mil /xmlreg/user/index.cfm]

[3] "Functional Description of the Mission Space." [http://fdms.msiac.dmso.mil/]

[4] Khoshafian, S., Abnous, R., *Object Orientation*, John Wiley and Sons, Inc., New York, NY, 1995.

[5] Kahng, J., McLeod D., "Dynamic Classificational Ontologies: Mediation of Information Sharing in Cooperative Federated Database Systems", *Cooperative Information Systems, Trends and Directions*, Academic Press, 1998.

[6] Li, W., and Clifton, C., "Semantic Integration in Heterogeneous Databases Using Neural Networks", Proceedings of the 20th VLDB Conference, Santiago, Chile, 1994, pp. 1-12.

[7] Liskov, B., Wing, J., "A Behavioral Notion of Subtyping," ACM Transactions on Programming Languages and Systems, Vol. 16, No. 6, November 1994, pp. 1811-1841.

[8] Pitoura, E., "Providing Database Inter-operability through Object-Oriented Language Constructs", Journal of Systems Integration, Volume 7, No. 2, August 1997, pp. 99-126.

[9] Walsh, A., Couch, J., Steinberg, D., *Java 2 Bible*, IDG Books Worldwide, Inc., Foster City, CA, 2000.

[10] Wiederhold, G., "Intelligent Integration of Information", ACM-SIGMOD 93, Washington, DC, May 1993, pp. 434-437.

[11] Young, P., *Integration of Heterogeneous Software Systems Through Computer-Aided Resolution of Data Representation Differences*, Ph.D. Dissertation, Naval Postgraduate School, Monterey, California, March, 2002.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center 8725 John J. Kingman Rd., STE 0944 Ft. Belvoir, VA 22060-6218	2
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
4.	Dr. John Salasin Defense Advanced Research Projects Agency 3701 North Fairfax Drive Arlington, VA. 22203-1714	1
5.	Dr. Valdis Berzins, CS/Be Computer Science Department Naval Postgraduate School Monterey, CA 93943	1
6.	Dr. Luqi, CS/Lq Computer Science Department Naval Postgraduate School Monterey, CA 93943	7

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center 8725 John J. Kingman Rd., STE 0944 Ft. Belvoir, VA 22060-6218	2
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
4.	Dr. John Salasin Defense Advanced Research Projects Agency 3701 North Fairfax Drive Arlington, VA. 22203-1714	1
5.	Dr. Valdis Berzins, CS/Be Computer Science Department Naval Postgraduate School Monterey, CA 93943	1
6.	Dr. Luqi, CS/Lq Computer Science Department Naval Postgraduate School Monterey, CA 93943	7